

NTIS

PB96-148762

NTIS
Information is our business.

TEMPORAL SPECIFICATION AND VERIFICATION OF REAL-TIME SYSTEMS

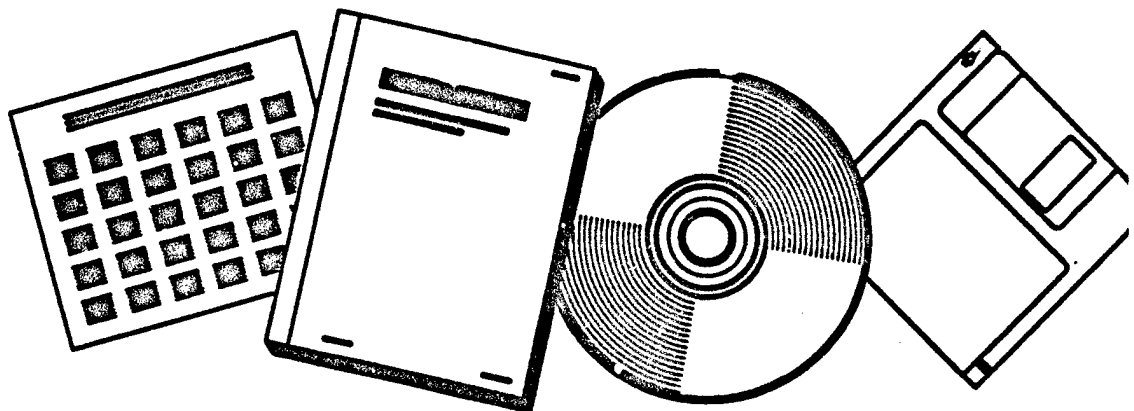
STANFORD UNIV., CA

DISTRIBUTION STATEMENT A

Approved for public release;
Distribution Unlimited

AUG 91

19970821 055



U.S. DEPARTMENT OF COMMERCE
National Technical Information Service

100% QUALITY INSPECTED 3

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and reviewing the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE 8/30/91	3. REPORT TYPE AND DATES COVERED
----------------------------------	---------------------------	----------------------------------

4. TITLE AND SUBTITLE THE TEMPORAL SPECIFICATION AND VERIFICATION OF REAL-TIME SYSTEMS	5. FUNDING NUMBERS
---	--------------------

6. AUTHOR(S) THOMAS A. HENZINGER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) DEPARTMENT OF COMPUTER SCIENCE STANFORD UNIVERSITY STANFORD, CA 94305	8. PERFORMING ORGANIZATION REPORT NUMBER STAN-CS-91-1380
---	---

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) DARPA ARLINGTON, VA	10. SPONSORING/MONITORING AGENCY REPORT NUMBER N00039-84C-0211
---	---

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT UNLIMITED	12b. DISTRIBUTION CODE
---	------------------------

13. ABSTRACT (Maximum 200 words) <p>We extend the specification language of temporal logic, the corresponding verification framework, and the underlying computational model to deal with real-time properties of reactive systems.</p> <p>Semantics We introduce the abstract computational model of <i>timed transition systems</i> as a conservative extension of traditional transition systems: qualitative fairness requirements are superseded by quantitative real-time constraints on the transitions. <i>Digital clocks</i> are introduced as observers of continuous real-time behavior. We justify our semantical abstractions by demonstrating that a wide variety of concrete real-time systems can be modeled adequately.</p> <p>Specification We present two conservative extensions of temporal logic that allow for the specification of timing constraints: while <i>timed temporal logic</i> provides access to time through a novel kind of time quantifier, <i>metric temporal logic</i> refers to time through time-bounded versions of the temporal operators. We justify our choice of specification languages by developing a general framework for the classification of real-time logics according to their complexity and expressive power.</p> <p>Verification We develop tools for determining if a real-time system that is modeled as a timed transition system meets a specification that is given in timed temporal logic or in metric temporal logic. We present both <i>model-checking algorithms</i> for the automatic verification of finite-state real-time systems and <i>proof methods</i> for the deductive verification of real-time systems.</p>

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
---------------------------------------	--	---	----------------------------

**THE TEMPORAL
SPECIFICATION AND VERIFICATION
OF REAL-TIME SYSTEMS**

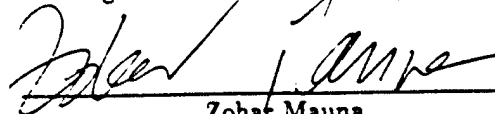
**A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY**

**By
Thomas A. Henzinger
August 1991**

DEAN QUALITY INSPECTED 3

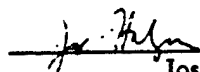
© Copyright 1991 by Thomas A. Henzinger
All Rights Reserved

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.



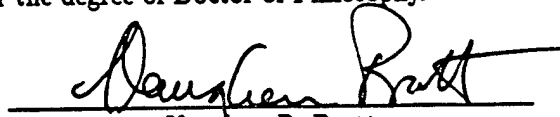
Zohar Maana
(Principal Advisor)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.



Joseph Y. Halpern

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.



Vaughan R. Pratt

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.



Richard Waldinger

Approved for the University Committee on Graduate Studies:

Dean of Graduate Studies

Abstract

We extend the specification language of temporal logic, the corresponding verification framework, and the underlying computational model to deal with real-time properties of reactive systems.

Semantics We introduce the abstract computational model of *timed transition systems* as a conservative extension of traditional transition systems: qualitative fairness requirements are superseded by quantitative real-time constraints on the transitions. *Digital clocks* are introduced as observers of continuous real-time behavior. We justify our semantical abstractions by demonstrating that a wide variety of concrete real-time systems can be modeled adequately.

Specification We present two conservative extensions of temporal logic that allow for the specification of timing constraints: while *timed temporal logic* provides access to time through a novel kind of time quantifier, *metric temporal logic* refers to time through time-bounded versions of the temporal operators. We justify our choice of specification languages by developing a general framework for the classification of real-time logics according to their complexity and expressive power.

Verification We develop tools for determining if a real-time system that is modeled as a timed transition system meets a specification that is given in timed temporal logic or in metric temporal logic. We present both *model-checking algorithms* for the automatic verification of finite-state real-time systems and *proof methods* for the deductive verification of real-time systems. Both techniques are conservative generalizations of the corresponding conventional approaches to program verification, which abstract time from consideration.

◇□ *Her story all over* □◇

*There will be time, there will be time
To prepare a face to meet the faces that you meet;
There will be time to murder and create,
And time for all the works and days of hands
That lift and drop a question on your plate;
Time for you and time for me,
And time yet for a hundred indecisions,
And for a hundred visions and revisions,
Before the taking of a toast and tea.*

— T.S. Eliot

Foreword

This dissertation deals with a relatively new — and thus perhaps necessarily somewhat controversial — topic of research. We are concerned with the question

How can we formally prove that a computing system meets time deadlines, where "time" is not measured asymptotically by a function on the size of the input, but by a number of days, hours, minutes, seconds, and milliseconds?

While the crucial importance of this so-called *real-time* verification problem has long been realized in practice, the challenge of formal reasoning about physical time has stirred the interest of the theoretical computer science community only recently, and only halfheartedly at that. The two main arguments that usually are brought forward to justify this negligence on the side of the theoreticians run as follows:

1. *There is no real-time verification problem.* The real-time behavior of a program depends on myriad factors such as the operating system and the underlying hardware. Worse, many of these factors generally are hidden from the programmer. If your program misses a real-time deadline by a constant factor, use (or wait for) a faster machine.

While this analysis of the problem is correct, the conclusion is not. Indeed, the real-time verification problem is far messier than the convenient abstraction of time by traditional program verification techniques. This observation only goes to show how clever these techniques are — and that they ought to be used whenever possible — but it does not make the need for, say, flight control systems disappear. Tampering with the speed of the machine is a naive nonsolution for the most important real-time systems, which are embedded in real-time environments whose speed is not under our control.

2. Granted, the verification of timing constraints is a real concern, but *there is nothing special about the real-time verification problem per se*. Time is but another parameter of the system state — treat it as such. That is, use the standard, established program verification techniques.

There is an obvious reply: no. Time is fundamentally different from the state components of a computing system. For all we know, time is continuous, monotonic, and divergent, and program variables generally happen not to have any of these characteristics. Only if we recognize the special status of time will we be able to find and exploit the intricacies of proving timing properties and avoid pitfalls like “Zeno” behaviors, which allow time to converge.

It is the purpose of this dissertation to provide technical arguments that substantiate the gut response. As a case in point, observe that time ranges over an infinite domain and, therefore, the verification of real-time systems escapes all of the widely used tools that have been developed for the verification of finite-state systems. We will show that this need not be the case: with some effort we can make finite-state techniques work if a program variable ranges over all natural or real numbers, provided at most one variable does so and provided the value of the variable is never decreased. Luckily, time happens to move in one direction only.

Acknowledgments

Chapters 2 and 6 of this dissertation present joint work with Zohar Manna and Amir Pnueli; Chapters 3 and 4 present joint work with Rajeev Alur. Yet the influence of any one of my three coauthors can be felt throughout the entire thesis, far beyond the particular sections that have evolved from joint papers. Zohar has been my advisor at Stanford for five years; he brought me to Stanford, he made possible two extremely productive visits to the Weizmann Institute in Israel, and he has been the source of the unlimited spiritual, technical, and financial support that has carried me through the doctoral program. Amir is one of those rare people who can almost single-handedly open an entire new area of research; it has been an immeasurable experience and pleasure to collaborate with him in a field he pioneered fifteen years ago. To Rajeev, let me plainly say: thank you! For whatever particulars I were to point out about his technical ingenuity and about how much fun he is to work with, I would not nearly do justice to his many contributions to my education as a researcher. Without any one of the three, this thesis would simply not exist.

That the thesis does exist in the present form is due also to the enthusiastic interest, support, and criticism I have received from colleagues at Stanford, at the Weizmann Institute, and at scientific meetings around the world. I am particularly grateful for innumerable discussions, of both highly technical and almost philosophical points, with the members of my reading committee, Joe Halpern, Vaughan Pratt, and Richard Waldinger, as well as Martín Abadi, Dave Dill, Allen Emerson, Tomás Feder, Sol Feferman, Tim Fernando, Adam Grove, Leslie Lamport, John Mitchell, Serge Plotkin, Martin Rinard, Vijay Saraswat, Fred Schneider, Natarajan Shankar, Yoav Shoham, Moshe Vardi, and Pierre Wolper. Special thanks go to Eddie Chang, Daphne Koller, Alex Wang, Liz Wolf, and Howard Wong-Toi for their detailed comments on various drafts of my thesis.

I would also like to take this opportunity to thank the four teachers who, up to this

point in my life, have been the most profound influences: Bruno Buchberger in Linz, who introduced me to logic and galvanized my interest in formal methods; Don Knuth at Stanford, whose Problem Seminar of 1987 inspired my (unreachable) ideal of what it means to "do research"; and, above all, my parents, who have never failed to believe in me.

Finally, I gratefully acknowledge the support I have received for the past three years from the IBM Corporation in the form of a graduate fellowship.

Tom Henzinger
Stanford, California
August 1991

Chronology

Much of Chapters 3 and 4 is based on material that was first presented at the *30th Annual IEEE Symposium on the Foundations of Computer Science* in October, 1989 [10], and at the *Fifth Annual IEEE Symposium on Logic in Computer Science* in June, 1990 [11]. A preliminary version of Chapter 5 appeared at the *Ninth Annual ACM Symposium on Principles of Distributed Computing* in August, 1990 [56]. The bulk of Chapters 2 and 6 has evolved from work that was first published at the *18th Annual ACM Symposium on Principles of Programming Languages* in January, 1991 [58].

Contents

Abstract	v
Foreword	viii
Acknowledgments	x
Chronology	xii
0 Introduction	1
0.1 The Real-time Verification Problem	2
0.2 The Temporal Methodology	3
I Specification	7
1 An Interleaving Model for Real Time	9
1.1 Timed State Sequences	10
1.1.1 Actual versus observed behavior	14
1.1.2 Analog-clock semantics	17
1.1.3 Digital-clock semantics	21
1.2 Real-time Properties	25
1.2.1 Refinability	26
1.2.2 Digitizability	30

1.2.3	Safety, liveness, and operability	34
1.3	Real-time Systems, Specifications, and Verification	45
1.3.1	Implementation languages	46
1.3.2	Specification logics	47
1.3.3	Analog versus digital verification	49
2	Real-time Systems	53
2.1	Abstract Model: Timed Transition Systems	53
2.1.1	Closure properties	56
2.1.2	Operability	60
2.2	Concrete Model: Multiprocessing Systems	67
2.2.1	Syntax: Timed transition diagrams	67
2.2.2	Semantics: Timed transition systems	69
2.2.3	Examples: Time-out and timely response	71
2.2.4	Message passing	75
2.3	Concrete Model: Multiprogramming Systems	78
2.3.1	Syntax and semantics	80
2.3.2	Scheduling strategies	82
2.3.3	Processor allocation	86
2.3.4	Priorities and interrupts	87
3	Real-time Logics	93
3.1	From Temporal to Real-time Logics	94
3.1.1	Linear temporal logic	95
3.1.2	Bounded invariance and bounded response	97
3.1.3	Real-time temporal logics	99
3.2	The Classical Theory of Timed State Sequences	101
3.2.1	The classical theory of state sequences	101

3.2.2	Adding time to state sequences	103
3.2.3	Decidability and expressibility	104
3.3	Timed Temporal Logic	110
3.3.1	Syntax and semantics	111
3.3.2	Expressive completeness	116
3.3.3	Timed extended temporal logic	119
3.3.4	Nonelementary extensions	123
3.4	Metric Temporal Logic	126
3.4.1	Syntax and semantics	126
3.4.2	Expressive completeness	130
3.5	Real-time Properties That Cannot Be Verified	132
3.5.1	Timed temporal logic revisited: Undecidable extensions	132
3.5.2	The classical theory revisited: Undecidable extensions	137
3.5.3	Decidability versus undecidability in real-time logics	139

II Verification **143**

4	Finite-state Verification	145
4.1	Deciding Timed Temporal Logic	146
4.1.1	Timed tableaux	146
4.1.2	Complexity of timed temporal logic	157
4.1.3	Timed extended temporal logic	160
4.2	Deciding Metric Temporal Logic	162
4.2.1	Metric tableaux	163
4.2.2	Complexity of metric temporal logic	166
4.3	Model Checking	167
4.3.1	Finite-state real-time systems	168

4.3.2	Model-checking algorithms	172
4.3.3	Complexity of model checking	173
5	Deductive Verification: General Part	177
5.1	Half-order Modal Logic	178
5.1.1	Syntax and semantics	179
5.1.2	Proof system	181
5.1.3	Completeness	185
5.1.4	Syntactic and semantic extensions	191
5.2	Half-order Temporal Logic	196
5.2.1	Syntax and semantics	197
5.2.2	Proof system	198
5.2.3	Completeness	203
5.2.4	Nonaxiomatizable extensions	210
5.3	Proving Timed Transition Systems Correct	212
5.3.1	Reasoning about explicit programs	213
5.3.2	Reasoning about one implicit program	215
6	Deductive Verification: Program Part	217
6.1	Properties of Timed Transition Systems	219
6.1.1	State properties	219
6.1.2	Temporal properties	222
6.2	Explicit-clock Reasoning	226
6.2.1	Explicit-clock temporal logic	226
6.2.2	Example: Race condition	229
6.3	Bounded-operator Reasoning	231
6.3.1	Deterministic rules	232
6.3.2	Conditional rules	241
6.3.3	Relative completeness	245

7 Discussion	253
7.1 Some Connections with Related Research	255
7.1.1 Semantic alternatives: Real-time models	255
7.1.2 Syntactic alternatives: Real-time languages	258
7.2 Some Directions for Future Research	261
Bibliography	267

Chapter 0

Introduction

real time: (software) (A) Pertaining to the processing of data by a computer in connection with another process outside the computer according to the time requirements imposed by the outside process. This term is also used to describe systems operating in conversational mode and processes that can be influenced by human intervention while they are in progress. (B) Pertaining to the actual time during which a physical process transpires, for example, the performance of a computation during the actual time that the related physical process transpires, in order that results of the computation can be used in guiding the physical process.

— IEEE Standard Dictionary of Electrical and Electronics Terms [66]

Many software and hardware components meet the tasks for which they have been designed only if they relate properly to the passage of time. Examples of such time-critical, or real-time, systems abound: embedded controllers have to oversee the operation of physical plants in physical time; the correctness of circuits and communication protocols often depends on gate delays and message delays, respectively. The behavior of computing systems in physical time is particularly difficult to predict by "inspection." This is why real-time systems are prime targets for a formal approach to system specification, verification, and development.

Logical formalisms and techniques have provided valuable aids for understanding and proving how complex computing systems behave. *Temporal logic*, in particular, has been

established as a working tool for the design and analysis of concurrent programs. One shortcoming of conventional temporal logic, however, is that it admits only the treatment of qualitative timing requirements, such as the demand that an event occurs "eventually." Because of this limitation, standard temporal logic is inadequate for the study of *real-time systems*, whose correctness depends crucially on the actual times at which events occur. In this thesis, we generalize the *temporal* methodology to encompass the analysis of *real-time* behavior.

0.1 The Real-time Verification Problem

A formal approach to the specification and verification of real-time systems must

1. assume a particular mathematical model C of *computation*,
2. assume a particular mathematical model T of *time*,
3. use a formal language \mathcal{L}_I — the *implementation* language — for the description of systems and their behavior over time,
4. use a formal language \mathcal{L}_S — the *specification* language — for the description of qualitative as well as quantitative timing properties of systems,
5. present algorithms and/or proof rules that facilitate a formal argument that a particular system has a particular property under the assumed semantics of computation and time. We call this question — whether a system S , given as an expression of \mathcal{L}_I , meets a specification ϕ , given as an expression of \mathcal{L}_S , with respect to the semantical assumption (C, T) — the *real-time verification problem* for $(C, T, \mathcal{L}_I, \mathcal{L}_S)$:

$$S \stackrel{?}{\models}_{(C, T)} \phi.$$

6. justify the adequacy of the semantical assumption (C, T) by showing how an answer to the real-time verification problem for $(C, T, \mathcal{L}_I, \mathcal{L}_S)$, which asks a question about an *abstract mathematical* domain, relates to the behavior of *concrete* systems in *physical* time.

This thesis addresses several instances of the real-time verification problem. Some are solved by algorithms, some are served by proof systems, and others are shown to be so hard that neither method can succeed. In particular, we will study the real-time verification problem, and demonstrate its practical relevance, for the following values of the parameters C , T , \mathcal{L}_I , and \mathcal{L}_S :

Computational model As we are primarily interested in the issues that are particular to time, we concentrate on a single simple and established model of computation. Throughout this thesis, we shall represent computation by *linear, interleaved* traces. Both systems and properties will be modeled as sets of traces. Those aspects of computation that can be represented only by branching or partial-order structures do not concern us.

Time model We investigate two models of time. A *continuous* time domain is required to represent the exact physical time of events; if the time of events is recorded by a fictitious digital clock, a *discrete* time domain suffices.

Implementation language A real-time implementation language should, from a practical point of view, be able to describe such real-time phenomena as time-outs and interrupts. We will argue that from a theoretical point of view, system descriptions should be executable, refinable, and independent of the time model. With *timed transition systems* we will introduce a language that satisfies these criteria.

Specification language A real-time specification language should, from a practical point of view, support natural and verifiable specifications of such real-time properties as time-bounded response, time-bounded invariance, and periodicity. We will argue that from a theoretical point of view, the language should strike a particular intrinsic balance between expressive power and complexity of the associated verification problem. With *timed temporal logic* and *metric temporal logic* we will introduce two orthogonal syntactic extensions of temporal logic that satisfy these criteria.

0.2 The Temporal Methodology

Let us briefly review the temporal-logic approach to system specification and verification and, simultaneously, use this opportunity to outline the organization of this thesis, which

advances the introduction of time into the temporal framework. The use of temporal logic for the formal analysis of reactive systems was first advocated by Pnueli [106]. The complete temporal methodology consists of four elements [91], all of which are affected by the addition of time. These four components structure the thesis:

Systems — *syntax and semantics.* The temporal approach is based on a trace model of computation. The semantics of a system as a set of traces is defined in four steps:

1. A concrete system P typically is described in a graphical language, by a transition diagram, although the methodology is applicable to a wide variety of programming languages as well as hardware description languages.
2. The system P is modeled as an abstract mathematical object S_P called a transition system.
3. The formal semantics of the transition system S_P is defined as a set $\Pi(S_P)$ of state sequences.
4. The set $\Pi(S_P)$ of state sequences represents the set $Beh(\Pi(S_P))$ of possible behaviors (i.e., traces) of the system P .

We incorporate time into this model by

1. describing a *real-time* system P by a *timed* transition diagram,
2. modeling P by a *timed* transition system S_P ,
3. defining the semantics of S_P as a set $\Pi(S_P)$ of *timed* state sequences, which associate a time with every state, and
4. having $\Pi(S_P)$ represent the possible behaviors $Beh(\Pi(S_P))$ of P in *physical time*.

Chapter 1 introduces timed state sequences — a formal abstraction of real-time behavior and the cornerstone of our approach. Chapter 2 introduces timed transition systems — a formal abstraction of real-time systems — and demonstrates their ability to model a wide variety of real-time phenomena that are encountered in practice.

Specifications — *syntax and semantics.* The temporal approach uses temporal logic to specify qualitative timing properties of systems. A formula ϕ of temporal logic defines a set $\Pi(\phi)$ of state sequences, which represents the set $Beh(\Pi(\phi))$ of traces that are

admitted by the specification ϕ . Consequently, a system P meets the specification ϕ , denoted by $P \models \phi$, iff all possible behaviors of P are admitted by ϕ :

$$\text{Beh}(\Pi(S_P)) \subseteq \text{Beh}(\Pi(\phi)).$$

We present two extensions of temporal logic that are interpreted over *timed* state sequences and allow the specification of quantitative timing properties. While *timed* temporal logic provides access to time through a novel kind of time quantifier, *metric* temporal logic refers to time through time-bounded versions of temporal operators. Chapter 3 introduces both languages and analyzes their expressive power and the complexity of their decision problems.

The preceding two elements of the temporal methodology fix the parameters of the verification problem and formalize the verification task as one of checking containment of trace sets. The following two elements present two fundamentally different techniques — a semantic one and a syntactic one — to solve this task.

Algorithmic verification — model checking. Model checking is a powerful automatic technique for the verification of systems with a *finite* state space. It relies on algorithms that check if a transition system meets a temporal-logic specification. Chapter 4 presents model-checking algorithms for timed transition systems and both timed and metric temporal logics. The real-time model-checking problem is shown to be exponentially harder than the corresponding untimed problem.

Deductive verification — proof rules. A deductive calculus is necessary for the verification of systems that escape model-checking methods because of the size of the state space. Such a proof system typically consists of three parts:

1. The *general* part axiomatizes temporal logic.
2. The *program* part provides proof rules for reasoning about structural properties of a particular system.
3. The *domain* part provides proof rules for reasoning about various data domains, if such a need arises.

Chapter 5 axiomatizes timed temporal logic. Chapter 6 introduces the program part of two proof methodologies for verifying timed transition systems. The proof methods are shown to be complete relative to reasoning about data domains.

Part I
Specification

Chapter 1

An Interleaving Model for Real Time

We study *reactive systems* — discrete systems that maintain an ongoing interaction with their environment [108]. Typical examples of reactive systems are distributed processes, which interact with each other, and real-time processes, whose interaction is prompted and constrained by the passage of time. In order to develop and apply a formal methodology for the specification and verification of a class of systems, the members of the class have to be modeled by mathematical objects. A model should be both *adequate*, in that it distinguishes between systems whose behaviors differ in one of the aspects under consideration, and *abstract*, in that it omits unnecessary detail by identifying systems without such disagreements.

One well-established approach to the modeling of reactive systems uses the paradigm of *interleaving* to represent concurrent activity. Whenever the simultaneous occurrence of several actions causes a distinct effect, such as process synchronization, the interleaving approach introduces joint “metaactions.” Independent concurrent actions, however, are simply nondeterministically sequentialized — as if they are performed in any order. It is this bold abstraction of representing the behavior of a system by a *linear* sequence of actions that brings about a major economic benefit, namely, that at any point only one action can occur and has to be analyzed. This astoundingly simple model turns out to be adequate for the study of many important properties of reactive systems. In particular, if the grain of atomicity of actions is chosen to be fine enough, it allows us to reason effectively about

the correctness of concurrent programs, independent of whether they are implemented in multiprogramming or multiprocessing environments.

The standard interleaving model is, however, abstract with respect to physical time; it identifies systems that admit the same sequences of actions, even if they do so at radically different speeds. While this abstraction facilitates the analysis of speed-independent systems, it is blatantly inadequate for *real-time* systems. To handle the quantitative timing requirements of these systems, we have to refine our model of computation by adding a time dimension. It has been claimed, however, that the interleaving model is intrinsically unsuited for the adjunction of time and that a more realistic modeling of concurrent activity is needed for the study of real-time systems [73]. For, it is argued, two actions executed truly in parallel surely take less time than any sequential execution of both actions. One of the main points that we demonstrate in this thesis is a refutation of this claim. We show that by a careful incorporation of time into the interleaving model, we can still model adequately most of the phenomena that occur in the timed execution of systems and yet retain the important economic advantages of interleaving models.

The key insight for combining interleaving and real time is to confine all actions to happen "instantaneously." This restriction proves workable, because any action a that takes $\delta > 0$ time units can be modeled by two instantaneous actions, *begin-a* and *end-a*, as well as a timing constraint that forces the latter action to happen (and only happen) δ time units after the former. Exploiting this observation, we may proceed to represent two independent simultaneous actions by nondeterministically interleaving the corresponding *begin* and *end* components as usual.

Having motivated an interleaving model of timed computation, we follow the temporal logic tradition of modeling the behavior of reactive systems by sequences of states rather than actions. Actions will, in our model, be represented implicitly, through their effects on the states of a system. The requirement of instantaneousness of actions translates, therefore, into instantaneousness of state changes.

1.1 Timed State Sequences

We study how reactive systems behave over time. The behavior of a system will be formalized as a sequence of snapshots of the global system state at certain times. The kinds of

systems and phenomena that can be captured adequately in this fashion depend on *when* and *how often* snapshots can be taken and on *how accurately* they can be timed:

When We do not restrict the times at which the state of a system can be observed, snapshots may be taken at any real point in physical, continuous time. This enables us to model both *synchronous* (clocked) processes, all of whose state changes occur in lock-step with a digital clock, and more general *asynchronous* processes, which change their state at arbitrary real points in time.

In fact, we allow several snapshots to be taken at the same time. Since state changes are instantaneous, such contemporaneous observations may yield different results. The possibility of several contemporaneous, yet ordered, snapshots is imperative for modeling simultaneous state changes by interleaving, as has been pointed out above.

How often We do require the number of snapshots to be countable; that is, we cannot observe the state of a system at every real point in time. Furthermore, we permit only finitely many snapshots between any two points in time; thus we prohibit infinite "Zeno" sequences of snapshots, whose times converge towards a real point in time. These restrictions are entirely adequate for modeling *discrete* processes, which change their state only finitely often between any two points in time and, thus, can be described completely by a divergent ω -sequence of state changes. We make no attempt to model *continuous* processes other than through discrete approximations.

How accurately The times of snapshots are recorded by a global (fictitious) clock. We distinguish between different manifestations of the clock. An *analog clock* gauges the time of every snapshot with infinite precision, while a *digital clock* records only the number of clock ticks, with respect to some time unit chosen a priori, between any two snapshots.

We emphasize that the type of clock that is used to time-stamp observations is completely independent of whether we model a system that is synchronous or asynchronous, discrete or continuous: even if the time of a snapshot is recorded by a digital clock, it may occur at any real point in time; even if we employ an analog clock, we can take at most countably many snapshots.

Formally, an observation is a pair that consists of a state showing the result of a snapshot and a time-stamp recording the time of the snapshot. A finite or infinite sequence

$$\rho: (\sigma_0, T_0) \longrightarrow (\sigma_1, T_1) \longrightarrow (\sigma_2, T_2) \longrightarrow (\sigma_3, T_3) \longrightarrow \dots$$

of $|\rho| \in \mathbb{N} \cup \{\omega\}$ observations is called a *timed state sequence* $\rho = (\sigma, T)$; it consists of a sequence σ of $|\rho|$ states σ_i , where $0 \leq i < |\rho|$, and a sequence T of corresponding time-stamps $T_i \in TIME$. At this point, we do not commit to any particular time domain $TIME$; we only assume that there is a total ordering $<$ on $TIME$ and demand that

Monotonicity Time does not decrease:

$$T_{i-1} \leq T_i \text{ for all } 0 < i < |\rho|.$$

This condition ensures that the global logical order of snapshots is consistent with the temporal order. It permits, however, adjacent observations with the same time-stamp. Thus a timed state sequence can be viewed as imposing a two-level ordering on snapshots: the macro-order is determined by the time-stamps, and within each time-stamp, there is a local logical order of observations.

Progress Time progresses:

either ρ is finite,
or for all $\delta \in TIME$, there is some $i \geq 0$ such that $T_i \geq \delta$.

This condition ensures that the time-stamps of observations do not converge.

We consider three types of fictitious clocks. In the *analog-clock* model we take $TIME$ to be the nonnegative real numbers \mathbb{R} ; in the *digital-clock* model, the nonnegative integers \mathbb{N} — thus assuming that the ticks of a digital clock are exactly one time unit apart. If time is immaterial, we let $TIME$ be any trivial one-element domain 1 ; this case is referred to as the *untimed* model. We call timed state sequences of the analog-clock model *precisely timed*; of the digital-clock model, *digitally timed*; and of the untimed model, *untimed*. Untimed state sequences are often identified with their state components. In both the analog-clock and the digital-clock model, we assume the standard definitions of customary functions such as addition on the time domain; in the untimed model, the definition of any total function on the time domain is uniquely determined. We write TSS_{TIME} for the set of timed state

sequences over the time domain *TIME*; whenever the value of the parameter *TIME* is unspecified (i.e., arbitrary), the subscript is suppressed. The set of all infinite timed state sequences is denoted by TSS^∞ .

The remainder of this thesis can be viewed as a contribution to the development of a formal theory of timed state sequences. We will propose, analyze, and relate various methods for defining (sets of) timed state sequences — methods that range from abstract specification languages, such as real-time logics, to concrete implementations of real-time systems.

Operations on timed state sequences

We routinely use the following operations on timed state sequences. For any timed state sequence $\rho = (\sigma, T)$, let ρ^- be its state component σ ; applied to a set Π of timed state sequences, this "untime" operation yields the corresponding set Π^- of state components. By $\rho^i = (\sigma^i, T^i)$, for $0 \leq i < |\rho|$, we denote the timed state sequence that results from ρ by deleting the first i observations. If $\delta \in TIME$, then $\rho + \delta = (\sigma, T + \delta)$ stands for the timed state sequence that is obtained from ρ by adding δ to all times in T . Given two infinite timed state sequences $\rho = (\sigma, T)$ and $\rho' = (\sigma', T')$ and $\delta \in TIME$, we say that ρ' is a δ -suffix of ρ iff for some $i \geq 0$,

- (1) $\sigma' = \sigma^i$,
- (2) either $i = 0$ or $T_{i-1} \leq \delta$, and
- (3) $T' + \delta = T^i$.

Note that ρ has at least one δ -suffix for every $\delta \in TIME$, but it may have several different δ -suffixes: if ρ contains n observations with the time-stamp δ , then there are $n + 1$ distinct δ -suffixes of ρ . For example, the timed state sequence

$$(a, 1) \longrightarrow (b, 3) \longrightarrow (c, 3) \longrightarrow (d, 4) \longrightarrow (e, 5) \longrightarrow \dots$$

has one 2-suffix,

$$(b, 1) \longrightarrow (c, 1) \longrightarrow (d, 2) \longrightarrow (e, 3) \longrightarrow \dots,$$

and three 3-suffixes:

$$(b, 0) \longrightarrow (c, 0) \longrightarrow (d, 1) \longrightarrow (e, 2) \longrightarrow \dots,$$

$$(c,0) \longrightarrow (d,1) \longrightarrow (e,2) \longrightarrow \dots,$$

$$(d,1) \longrightarrow (e,2) \longrightarrow \dots.$$

Modeling system behaviors by timed state sequences

We will define the formal semantics of a reactive system S as a set of timed state sequences that represents the possible behaviors of S . To do this properly, we have to address the following two issues:

1. Given a set Π of timed state sequences, which set Ψ of system behaviors does Π represent? We take the view that a timed state sequence ρ is obtained by observing a system and, consequently, represents any behavior that may result in the observation sequence ρ .
2. Given a set Ψ of system behaviors, is there a set Π of timed state sequences that represents Ψ ? If there are several such sets Π , is there a preferred one? Indeed, we are limited to a certain class of reactive systems that can be represented by a timed state sequence semantics in both the analog-clock model and the digital-clock model. Moreover, for any system S in this class, we will select a particular set of timed state sequences to represent the possible behaviors of S , because we shall require that the semantics of S satisfies certain closure conditions.

Both of these issues are discussed in the rest of this chapter. While the remainder of Section 1.1 provides a formal answer to the first question, Section 1.2 addresses the second point. The practically oriented reader who is not interested in the details of modeling reactive systems by sets of timed state sequences may proceed immediately to Chapter 2, which defines and illustrates the timed state sequence semantics of concrete systems.

1.1.1 Actual versus observed behavior

A timed state sequence is the result of *observing* the behavior of a reactive system. We say that a sequence of snapshots is *state-complete* iff every system state is observed; it is *time-complete* iff every real point in time is observed. While we demand state-completeness when observing discrete systems, no countable number of snapshots can be time-complete.

Thus, unlike a state sequence when time is of no interest, a timed state sequence may not uniquely define the *actual* behavior of a system in physical time; observing different system behaviors can result in the same state-complete timed state sequence. There are two reasons for this ambiguity:

1. Two *contemporaneous* observations are necessary to obtain complete information about a state change — one snapshot showing the state immediately before and the other showing the state immediately after the instantaneous state change. If a timed state sequence fails to observe a state change in this fashion, then it does not define a unique system behavior. For example, if given the (state-complete) precisely timed observation sequence

$$(a, 5) \longrightarrow (b, 6),$$

all we know is that the system state changes from a to b between time 5 and time 6; the actual state change may occur as early as at time 5.1 or as late as at time 5.9.

2. In the digital-clock model, there is a second source of ambiguity, which is caused by the inaccuracy of the fictitious clock that records the times of snapshots. For example, if given the (state-complete) digitally timed observation sequence

$$(a, 5) \longrightarrow (b, 5),$$

all we know is that the system state changes from a to b between the fifth and the sixth tick of a digital clock; assuming that the first clock tick happens at time 0, the state change may, again, occur as early as at time 5.1 or as late as at time 5.9.

Both sources of timing ambiguities need to be resolved. In other words, so far we have given only the syntax of timed state sequences; we still need to agree on what a timed state sequence, in either clock model, "means."

In the following two subsections, we will define the semantics of timed state sequences, first for the analog-clock model and then for the digital-clock model. To eliminate both sources of timing ambiguities, we introduce, on top of interleaving, two additional layers of abstraction in modeling *real-time* behavior by ω -sequences of observations:

1. *Precisely timed observation of actual behavior* — from behaviors of reactive systems to timed state sequences over R .

2. *Digital timing of observed behavior* — from timed state sequences over R to timed state sequences over N .

The untimed model requires neither layer; the analog-clock model, the first layer only; the digital-clock model, both layers. In both steps, we use the convention that a timed state sequence does not denote a single behavior, but rather that a *set* of timed state sequences stands for a *set* of behaviors of a reactive system. For this purpose, we need to formalize the notion of "(actual) behavior" of a system.

Real-time behavior

Not surprisingly, we employ timed state sequences that contain no timing ambiguities to serve as representatives of system behaviors. A *behavior* is a timed state sequence $\rho = (\sigma, T)$ such that

Determinism Time advances only if the state does not change:

$$\text{for all } 0 < i < |\rho|, \text{ either } \sigma_{i-1} = \sigma_i \text{ or } T_{i-1} = T_i.$$

This property ensures that every state change is observed by two contemporaneous snapshots; a timed state sequence that satisfies it is called *deterministic*. For a set $\Pi \subseteq TSS$, let $Det(\Pi) \subseteq \Pi$ be the set of all deterministic timed state sequences in Π . A set $\Pi \subseteq TSS$ is called *deterministic* iff $Det(\Pi) = \Pi$.

Stutter-freedom Time advances to the next state change:

$$\begin{aligned} &\text{for all } 0 < i < |\rho|, \text{ not both } \sigma_{i-1} = \sigma_i \text{ and } T_{i-1} = T_i, \text{ and} \\ &\text{for all } 0 \leq i < |\rho|, \text{ either } i > 0 \text{ and } \sigma_{i-1} \neq \sigma_i, \text{ or } i < |\rho| - 1 \text{ and } \sigma_i \neq \sigma_{i+1}. \end{aligned}$$

This property ensures that a minimal number of snapshots is taken; a timed state sequence that satisfies it is called *stutter-free*. Note that according to our definition, if $\rho \in TSS$ is stutter-free, then $|\rho| > 1$.

We write $Beh(\Pi) \subseteq \Pi$ for the set of all behaviors in $\Pi \subseteq TSS$.

A behavior of the analog-clock model is called a *real-time* behavior. *Determinism* by itself guarantees that a precisely timed state sequence uniquely defines the behavior of a system in physical time, because any state-complete deterministic timed state sequence

over R contains enough information to determine the precise time of every state change. However, depending on how many repetitious snapshots of a system state are taken, the same system behavior may result in different deterministic sequences of observations. The requirement of *stutter-freedom* has been added to obtain a bijection between actual and observed system behaviors.

Another operation on timed state sequences

It will turn out to be convenient to have an operation on timed state sequences that removes redundant observations. Given a timed state sequence $\rho = (\sigma, T)$, we define the timed state sequence $\natural\rho$ to be a maximal stutter-free subsequence of ρ ; that is, $\natural\rho$ results from ρ by deleting some observations. In particular, the observation (σ_i, T_i) , for $0 \leq i < |\rho|$, is deleted unless either the previous observation or the subsequent observation shows a state different from σ_i ; in addition, all observations but one in any subsequence of successive identical observations are deleted. For example, if ρ is the timed state sequence

$$(a, 0) \longrightarrow (a, 1) \longrightarrow (b, 1) \longrightarrow (c, 1) \longrightarrow (c, 1) \longrightarrow (d, 1) \longrightarrow (d, 2) \longrightarrow (d, 3) \longrightarrow (e, 5),$$

then $\natural\rho$ is the timed state sequence

$$(a, 1) \longrightarrow (b, 1) \longrightarrow (c, 1) \longrightarrow (d, 1) \longrightarrow (d, 3) \longrightarrow (e, 5).$$

Note that a timed state sequence ρ is deterministic iff $\natural\rho$ is deterministic. Consequently, if ρ is deterministic, then $\natural\rho$ is a behavior. For $\Pi \subseteq TSS$, let $\natural\Pi = \{\natural\rho \mid \rho \in \Pi\}$. Thus, for any set Π of timed state sequences, the set

$$\natural Det(\Pi) = Det(\natural\Pi)$$

is a set of behaviors.

1.1.2 Analog-clock semantics

We define our interpretation of timed state sequences in the analog-clock model in such a way that every precisely timed state sequence ρ denotes a set $[\rho]$ of real-time behaviors. Informally, we take $[\rho]$ to contain all real-time behaviors that are consistent with the observation sequence ρ under the assumption of state-completeness; that is, at any point in

time, the system state is assumed to be equal to one of the two states shown by neighboring observations. This interpolation of states induces a closure operation on sets of timed state sequences.

Stuttering

A set Π of timed state sequences is *weakly closed under stuttering* iff whenever the sequence

$$\dots \longrightarrow (\sigma_i, T_i) \longrightarrow (\sigma_{i+1}, T_{i+1}) \longrightarrow \dots$$

is in Π , then so are the sequences

$$\dots \longrightarrow (\sigma_i, T_i) \longrightarrow (\sigma_i, \delta) \longrightarrow (\sigma_{i+1}, T_{i+1}) \longrightarrow \dots$$

and

$$\dots \longrightarrow (\sigma_i, T_i) \longrightarrow (\sigma_{i+1}, \delta) \longrightarrow (\sigma_{i+1}, T_{i+1}) \longrightarrow \dots$$

for all $T_i \leq \delta \leq T_{i+1}$; whenever the sequence

$$(\sigma_0, T_0) \longrightarrow \dots$$

is in Π , then so is the sequence

$$(\sigma_0, \delta) \longrightarrow (\sigma_0, T_0) \longrightarrow \dots$$

for all $\delta \leq T_0$; and whenever the finite sequence

$$\dots \longrightarrow (\sigma_n, T_n)$$

is in Π , then so are all infinite timed state sequences of the form

$$\dots \longrightarrow (\sigma_n, T_n) \longrightarrow (\sigma_n, T_{n+1}) \longrightarrow (\sigma_n, T_{n+2}) \longrightarrow \dots$$

The set $\Pi \subseteq TSS$ is called *strongly closed under stuttering* iff

- (1) Π is weakly closed under stuttering and
- (2) $\Pi \subseteq \Pi$.

While closing a set Π under weak stuttering can only add observations to timed state sequences in Π , closing Π under strong stuttering may require the deletion of redundant observations. Given a set Π of timed state sequences, we define the *weak stuttering closure* $\Gamma(\Pi)$ of Π to be the smallest set of timed state sequences that contains Π and is weakly closed under stuttering; the (*strong*) *stuttering closure* $\Gamma_h(\Pi)$ of Π is the smallest set of timed state sequences that contains Π and is strongly closed under stuttering. If $\rho \in TSS$, we simply write $\Gamma(\rho)$ and $\Gamma_h(\rho)$ for the weak and strong stuttering closures of $\{\rho\}$.

We now list a few useful properties of stuttering closures. Clearly, $\Gamma(\Pi) \subseteq \Gamma_h(\Pi)$ for all $\Pi \subseteq TSS$. If Π is deterministic, then so are both $\Gamma(\Pi)$ and $\Gamma_h(\Pi)$; thus, if Π is (weakly) closed under stuttering, then so is $Det(\Pi)$.

Lemma 1.1 (Stuttering) *For every set Π of timed state sequences, $h\Gamma(h\Gamma(\Pi)) = h\Gamma(\Pi)$.*

Proof of Lemma 1.1 We assume that $\rho \in h\Gamma(h\Gamma(\Pi))$ — that is, $\rho = h\rho_1$ and $\rho_1 \in \Gamma(h\Gamma(\rho_2))$ for some $\rho_2 \in \Pi$ — and show that $\rho \in h\Gamma(\Pi)$. Let $\rho'_1 \in \Gamma(\rho_1) \cap \Gamma(\rho_2)$ be the timed state sequence that results from merging both ρ_1 and ρ_2 . It is not hard to see that $\rho = h\rho'_1$ and, therefore, $\rho \in h\Gamma(\rho_2)$. ■

This lemma implies that strong closure under stuttering can be obtained by a single application of the “unstutter” operator h :

$$\Gamma_h(\Pi) = \Gamma(h\Gamma(\Pi)).$$

It follows that $\Gamma(\Pi) = \Gamma_h(\Pi)$ iff Π is a set of behaviors.

Stuttering semantics

In any clock model, we let a timed state sequence stand for the *set* of all sequences in its (strong) stuttering closure. Accordingly, a set Π of timed state sequences is said to *specify under stuttering* the set

$$[\Pi] = Beh(\Gamma_h(\Pi))$$

of behaviors; if $\rho \in TSS$, we abbreviate $[\{\rho\}]$ to $[\rho]$. Note that if $\Pi' \subseteq TSS$ is strongly closed under stuttering, then $h\Pi'$ selects exactly the stutter-free sequences in Π' . Hence we can characterize the set $[\Pi]$ of behaviors in $\Gamma_h(\Pi)$ as follows:

$$[\Pi] = hDet(\Gamma_h(\Pi)).$$

Since \natural and Det commute, by Lemma 1.1 we have, equivalently, that

$$[\Pi] = \natural Det(\Gamma(\Pi)).$$

Also observe that $Beh(\Pi) \subseteq [\Pi]$ and

$$[\Pi] = \bigcup_{\rho \in \Pi} [\rho]$$

for all $\Pi \subseteq TSS$. If Π is closed under stuttering, then $[\Pi] = Beh(\Pi)$; if Π is deterministic, then $[\Pi] = \natural\Pi$; if Π is a set of behaviors, then $[\Pi] = \Pi$.

A precisely timed state sequence ρ specifies, under stuttering, a set of real-time behaviors $[\rho]$; we agree that the *analog denotation* of a set Π of precisely timed state sequences is $[\Pi]$. This interpretation of observation sequences has the properties we desire:

1. A deterministic precisely timed observation sequence ρ denotes the real-time behavior $\natural\rho$.
2. A nondeterministic precisely timed observation sequence ρ denotes the set of all real-time behaviors that correspond to deterministic sequences that are obtained from ρ by interpolating states.
3. A set Π of precisely timed observation sequences denotes the set of all real-time behaviors that correspond to sequences in Π .

Two sets of timed state sequences are called *equivalent under stuttering* iff they specify under stuttering the same set of behaviors. For example, the singleton set that contains the timed state sequence

$$(a, 5) \longrightarrow (b, 6)$$

is, under stuttering, equivalent to the set that contains all timed state sequences of the form

$$(a, \delta) \longrightarrow (b, \delta)$$

with $5 \leq \delta \leq 6$, which is infinite in the analog-clock model. Also, any two sets of timed state sequences with the same stuttering closure are equivalent under stuttering. Two sets of behaviors are equivalent under stuttering iff they are equal.

We have defined equivalence in the analog-clock model as equivalence under stuttering. It is worth pointing out that, in addition, our definitions of weak stuttering closure, stutter-freedom, and equivalence under stuttering properly generalize the corresponding untimed notions for state sequences [1]: if $TIME = 1$, then $\Gamma(\rho^-) = \Gamma(\rho)^-$ and $\mathfrak{h}(\rho^-) = (\mathfrak{h}\rho)^-$. There is no untimed equivalent to determinism, because every untimed state sequence is trivially deterministic. We conform with the usual convention that, in the untimed model, a set Π of untimed state sequences denotes the set $[\Pi] = \mathfrak{h}\Pi$ of untimed behaviors.

Transparency

We will reserve a special role for sets of precisely timed state sequences whose analog denotation is immediately apparent. A set Π of timed state sequences is called *transparent* iff

$$[\Pi] = Beh(\Pi);$$

that is, a transparent set Π specifies under stuttering precisely the behaviors in Π . Note that all sets of timed state sequences that are closed under stuttering are transparent.

1.1.3 Digital-clock semantics

Recall that we assume that successive clock ticks of a digital clock are precisely one time unit apart. In other words, we scale time so that a time unit corresponds to the distance between two ticks of a digital clock. Still, there are infinitely many "different" digital clocks that meet this requirement — one for every real number $0 \leq \epsilon < 1$, which determines the absolute times at which the clock ticks occur. It will be useful to have all of these clocks available. Thus we shall henceforth identify digital clocks with reals in the interval $[0, 1)$. For example, the digital clock 0.5 ticks first at time 0.5 , then at time 1.5 , then at time 2.5 , and so on.

Digitization

If the times of a sequence $\rho \in TSS_R$ of snapshots are recorded by a digital clock ϵ , we obtain a timed state sequence, denoted by $[\rho]_\epsilon$, over N . For any timed state sequence $\rho = (\sigma, T)$ and digital clock $0 \leq \epsilon < 1$, let the *digitization* $[\rho]_\epsilon = (\sigma, [T]_\epsilon)$ of ρ with respect to ϵ be the

timed state sequence

$$(\sigma_0, [\tau_0]_\epsilon) \longrightarrow (\sigma_1, [\tau_1]_\epsilon) \longrightarrow (\sigma_2, [\tau_2]_\epsilon) \longrightarrow (\sigma_3, [\tau_3]_\epsilon) \longrightarrow \dots,$$

where $[x]_\epsilon = \lfloor x \rfloor$ if $x \leq \lfloor x \rfloor + \epsilon$, else $[x]_\epsilon = \lfloor x \rfloor$. In words, the digitization $[\rho]_\epsilon$ results from ρ by rounding, with respect to ϵ , all times to integers. For example, if ρ is the timed state sequence

$$(a, 0.1) \longrightarrow (b, 0.5) \longrightarrow (c, 0.6) \longrightarrow (d, 1.4) \longrightarrow (e, 5) \longrightarrow (f, 6.9),$$

then $[\rho]_{0.5}$ is the timed state sequence

$$(a, 0) \longrightarrow (b, 0) \longrightarrow (c, 1) \longrightarrow (d, 1) \longrightarrow (e, 5) \longrightarrow (f, 7).$$

For a set Π of timed state sequences, let $[\Pi] \subseteq TSS_N$ be the set of all digitizations of sequences in Π :

$$[\Pi] = \{[\rho]_\epsilon \mid \rho \in \Pi \text{ and } 0 \leq \epsilon < 1\};$$

if $\rho \in TSS$, then $\{[\rho]\}$ is abbreviated to $[\rho]$. Note that if Π is a set of timed state sequences over N , then $[\Pi] = \Pi$.

Digitization semantics

Now let us define our interpretation of digitally timed state sequences. Recall that every digitally timed state sequence ρ stands, in fact, for its stuttering closure $\Gamma_b(\rho) \subseteq TSS_N$. We have already pointed out that the digital-clock model involves *two* layers of abstractions:

1. First we will associate with a set $\Pi \subseteq TSS_N$ of digitally timed observation sequences a set $[\Gamma_b(\Pi)]^{-1} \subseteq TSS_R$ of precisely timed observation sequences.
2. Then we will use the analog-clock semantics of $[\Gamma_b(\Pi)]^{-1}$ to define the set $[[\Pi]]_N$ of real-time behaviors denoted by Π .

If we have only a particular — or, worse yet, an unknown — digital clock to time-stamp observations, there is a large number of precisely timed state sequences that may result in a given sequence of digitally timed snapshots. To reduce the degree of ambiguity, we assume that *all* digital clocks are, simultaneously, available when observing the behavior of a system. Thus we obtain *infinitely many* — though not necessarily distinct — digitally

timed state sequences from every precisely timed state sequence. For example, the timed state sequence

$$(a, 0.5) \longrightarrow (b, 0.5)$$

results, depending on which digital clock is used, in one of the two digitally timed state sequences in the set

$$\Pi = \{(a, 0) \longrightarrow (b, 0), (a, 1) \longrightarrow (b, 1)\}.$$

Moreover, a subset of Π is obtained by simultaneous digitization with all digital clocks from (and only from) precisely timed state sequences of the form

$$(a, \delta) \longrightarrow (b, \delta)$$

with $0 \leq \delta \leq 1$. These are the precisely timed observation sequences that are, in our interpretation of the digital-clock model, taken to be consistent with the set Π of digitally timed observation sequences.

Formally, a set Π of timed state sequences over N is said to *specify under digitization* the set

$$[\Pi]^{-1} = \{\rho \in TSS_R \mid [\rho] \subseteq \Pi\}$$

of timed state sequences over R all of whose digitizations are contained in Π . It is not hard to see that, in particular, $[\{\rho\}]^{-1} = \{\rho\}$ for all $\rho \in TSS_N$. Moreover,

$$[\Pi_1]^{-1} \cup [\Pi_2]^{-1} \subseteq [\Pi_1 \cup \Pi_2]^{-1}$$

for all $\Pi_1, \Pi_2 \subseteq TSS_N$. In general this subset relationship, and thus the subset relationship $\Pi \subseteq [\Pi]^{-1}$ for all $\Pi \subseteq TSS_N$, is a proper one. The "inverse" notation for $[\Pi]^{-1}$ stems from the observation that $[[\Pi]^{-1}] = \Pi$. Inverse digitization preserves the following properties of sets of timed state sequences:

Lemma 1.2 (Inverse digitization) *For every set Π of digitally timed state sequences:*

- (1) $Det([\Pi]^{-1}) = [Det(\Pi)]^{-1}$.
- (2) If Π is weakly closed under stuttering, then so is $[\Pi]^{-1}$.
- (3) If Π is closed under stuttering, then so is $[\Pi]^{-1}$.

Proof of Lemma 1.2 (1) To see that $Det([\Pi]^{-1}) \subseteq [Det(\Pi)]^{-1}$, observe that whenever a precisely timed state sequence ρ is deterministic, then so is the set $[\rho]$ of digitizations. To see that $[Det(\Pi)]^{-1} \subseteq Det([\Pi]^{-1})$, note that if ρ is not deterministic, then there is some $0 \leq \epsilon < 1$ such that the digitization $[\rho]_\epsilon$ is not deterministic either.

(2) We suppose that Π is weakly closed under stuttering and show that $[\Pi]^{-1}$ is weakly closed under stuttering. Assume that $\rho \in [\Pi]^{-1}$ — that is, $[\rho] \subseteq \Pi$ — and that $\rho' \in \Gamma(\rho)$ for two precisely timed state sequences ρ and ρ' . Then

$$[\rho'] \subseteq [\Gamma(\rho)] \subseteq \Gamma([\rho]) \subseteq \Gamma(\Pi) \subseteq \Pi,$$

which implies that $\rho' \in [\Pi]^{-1}$.

(3) We suppose that Π is closed under stuttering; since part (2) implies that $[\Pi]^{-1}$ is weakly closed under stuttering, it suffices to show that $\mathfrak{h}[\Pi]^{-1} \subseteq [\Pi]^{-1}$. Assume that $\rho \in \mathfrak{h}[\Pi]^{-1}$; that is, $\rho = \mathfrak{h}\rho'$ for some precisely timed state sequence ρ' with $[\rho'] \subseteq \Pi$. Then $\mathfrak{h}[\rho'] \subseteq \mathfrak{h}\Pi$ and, as Π is closed under stuttering, $[\mathfrak{h}\rho'] \subseteq \Pi$, which implies that $\rho \in [\Pi]^{-1}$. ■

We let a set Π of digitally timed state sequences represent the set $[\Gamma_{\mathfrak{h}}(\Pi)]^{-1}$ of precisely timed state sequences all of whose digitizations are contained in the stuttering closure of Π . It follows that Π denotes in the digital-clock model the set

$$[\Pi]_{\mathfrak{N}} = [[\Gamma_{\mathfrak{h}}(\Pi)]^{-1}]$$

of real-time behaviors; we refer to this set as the *digital denotation* of Π . By Lemma 1.2, the digital denotation of Π can be characterized as follows:

$$[\Pi]_{\mathfrak{N}} = Beh([\Gamma_{\mathfrak{h}}(\Pi)]^{-1}) = \mathfrak{h}[Det(\Gamma_{\mathfrak{h}}(\Pi))]^{-1}.$$

We point out that, unlike in the precisely timed case, *weak* closure under stuttering is insufficient to define the denotation of a set of digitally timed state sequences. For example, we want the set that contains all digitally timed state sequences of the form that $n \in \mathfrak{N}$ identical observations $(a_0, 0)$ are followed by the observation sequence

$$\begin{aligned} (a_0, 0) &\longrightarrow (a_1, 0) \longrightarrow \cdots \longrightarrow (a_{n-1}, n-1) \longrightarrow (a_n, n-1) \longrightarrow \\ (a_n, n+1) &\longrightarrow (a_{n+1}, n+1) \longrightarrow (a_{n+1}, n+2) \longrightarrow (a_{n+2}, n+2) \longrightarrow \cdots \end{aligned}$$

to denote, among others, the real-time behavior

$$(a_0, 0) \longrightarrow (a_1, 0) \longrightarrow (a_1, 1\frac{1}{2}) \longrightarrow (a_2, 1\frac{1}{2}) \longrightarrow (a_2, 2\frac{2}{3}) \longrightarrow (a_3, 2\frac{2}{3}) \longrightarrow \cdots$$

Two sets of timed state sequences N are called *digitally equivalent* iff they have the same digital denotations. For example, the singleton set that contains the timed state sequence

$$(a, 0) \longrightarrow (b, 1)$$

is digitally equivalent to the set that contains the two timed state sequences

$$(a, 0) \longrightarrow (b, 0) \quad \text{and} \quad (a, 1) \longrightarrow (b, 1).$$

Also, any two sets of digitally timed state sequences with the same stuttering closure are digitally equivalent.

1.2 Real-time Properties

We will define the formal semantics of a reactive system to be a set of infinite timed state sequences. Such a set is called a (*real-time*) *property*. In accordance with different clock models, we distinguish analog, digital, and untimed properties. Recall that both analog and digital properties denote sets of real-time behaviors: an analog property $\Pi \subseteq TSS_{\mathbb{R}}^{\omega}$ represents its analog denotation $\llbracket \Pi \rrbracket$; a digital property $\Pi \subseteq TSS_{\mathbb{N}}^{\omega}$ represents its digital denotation $\llbracket \Pi \rrbracket_{\mathbb{N}}$. An untimed property Π denotes the set $\llbracket \Pi \rrbracket$ of untimed behaviors.

Let $\llbracket S \rrbracket$ be the set of possible real-time behaviors of the reactive system S . The *analog semantics* $\Pi_{\mathbb{R}}(S)$ of S ought to be an analog property that denotes $\llbracket S \rrbracket$; the *digital semantics* $\Pi_{\mathbb{N}}(S)$ of S , a digital property that denotes $\llbracket S \rrbracket$. Note that there are reactive systems without a suitable digital semantics; that is, not every set of real-time behaviors is represented by some digital property. For example, no set of digitally timed state sequences denotes the singleton set that contains the real-time behavior

$$(a, 0.5) \longrightarrow (b, 0.5);$$

in other words, our digital-clock model is not expressive enough to model a reactive system whose only possible behavior is the given one. We refer to the reactive systems for which there is a suitable digital semantics as *digitizable*. Thus, the digital-clock model restricts us to the study of digitizable systems.

In the analog-clock model, no such limitation arises, because every set Ψ of real-time behaviors is denoted by some analog property — the set Ψ itself, with all finite sequences

in Ψ being extended by infinite stuttering of the final observation. Yet we shall not permit all analog properties as being a suitable semantics of a system. This is because we want descriptions of reactive systems to have certain properties, which put four additional demands on the analog semantics of a system and its presentation:

Transparency We want a system description to be *transparent*; that is, we want the behaviors in the analog semantics of a system S to be exactly the possible real-time behaviors of S :

$$\llbracket S \rrbracket = \llbracket \Pi_R(S) \rrbracket = Beh(\Pi_R(S)).$$

Refinability We want a system description to be *refinable*. This is the case if its analog semantics is weakly closed under stuttering.

Digitizability When using the digital clock model — and we shall use it extensively for verification — we want a system description to be uniform, *independent of the clock model*. We will show that this is the case if its analog semantics is closed under a condition we call “digitizability.”

Operationality We want a system description to be *executable*. We will show that this is the case if its analog semantics is presented in a way that we call “operational.”

We shall restrict ourselves to system descriptions that satisfy all four criteria. Although this restriction will greatly influence our choice of languages to describe reactive systems, it does not limit the set of systems under consideration — the digitizable ones, which can be modeled adequately in both clock models. This is because we will show that the set of real-time behaviors of every digitizable system is denoted by an analog property that is closed under stuttering, digitizable, and given operationally. Note that closure under stuttering implies both the requirement of *transparency* and the requirement of *refinability*. In the following three subsections, we discuss the three issues of *refinability*, *digitizability*, and *operationality* one by one.

1.2.1 Refinability

So far, we have implicitly associated a fixed set of global states with every reactive system. This static view prohibits the study of large systems for managerial reasons. In particular, with every step in the hierarchical specification, design, and verification of a complex

reactive system, the state space is refined by enlarging its visible portion. The vertical decomposition of systems can be formalized by refinement mappings between increasingly detailed system descriptions. Since expansion of the visible portion of the state space can increase the frequency at which state changes are observed, the formal semantics of a system description needs to be weakly closed under stuttering to guarantee the existence of refinement mappings [78].

Let Ψ be a set of behaviors. We say that the property Π is a *refinable specification* of Ψ iff

- (1) Π specifies Ψ under stuttering and
- (2) Π is weakly closed under stuttering.

For example, the stuttering closure $\Gamma_1(\Psi) = \Gamma(\Psi)$ is a refinable specification of Ψ , which implies that every set of behaviors has a refinable specification. Moreover, two distinct sets of behaviors cannot have the same refinable specification.

In the analog-clock model, a refinable specification $\Pi \subseteq TSS_R$ of a set Ψ of real-time behaviors is called an *analog specification* of Ψ . We have argued that the analog semantics $\Pi_R(S)$ of a reactive system S ought to be an analog specification of the set $\llbracket S \rrbracket$ of possible real-time behaviors of S . Although the formal semantics of a reactive system has to be sufficiently discriminative, we like it to be not unnecessarily discriminative either; that is, we look for a one-to-one correspondence between properties that represent the formal semantics of reactive systems and properties that represent the possible behaviors of reactive systems. There are, however, in general infinitely many refinable specifications of a set of behaviors. Thus, for every set Ψ of real-time behaviors, we designate a particular refinable specification of Ψ as the analog semantics of all reactive systems S with $\llbracket S \rrbracket = \Psi$. There are two obvious candidates for $\Pi_R(S)$, which will be discussed in turn.

Interval semantics

One option is to agree that the set of all infinite sequences in the stuttering closure $\Gamma(\Psi)$ serves as the analog semantics of a system whose possible real-time behaviors are Ψ . Since the stuttering closure of Ψ is deterministic, this amounts to restricting ourselves to deterministic timed state sequences when modeling reactive systems. Note that a deterministic timed state sequence can, alternatively, be viewed as a *state interval sequence*, in which a

closed time interval is associated with every state and adjacent intervals have overlapping end-points; for example, the deterministic timed state sequence

$$(a, 1) \longrightarrow (b, 1) \longrightarrow (b, 3) \longrightarrow (b, 4) \longrightarrow (c, 4) \longrightarrow (d, 4) \longrightarrow (d, 5) \longrightarrow (e, 5)$$

corresponds to the state interval sequence

$$(a, [0, 1]) \longrightarrow (b, [1, 3]) \longrightarrow (b, [3, 4]) \longrightarrow (c, [4, 4]) \longrightarrow (d, [4, 5]) \longrightarrow (e, [5, \omega])$$

(which is not stutter-free). Recall that the point-wise overlap of intervals is necessary to accommodate interleaving. We took this approach of modeling real-time behavior by state interval sequences at a different opportunity [57].

We find it more convenient to allow arbitrary — not necessarily deterministic — timed state sequences when defining the formal semantics of reactive systems; they permit us, for example, to characterize a reactive system that changes its state from a to b , nondeterministically, at any time between 0 and 1 simply by the stuttering closure of the timed state sequence

$$(a, 0) \longrightarrow (b, 1),$$

rather than the stuttering closure of an infinite set of deterministic timed state sequences. Hence we will take the analog semantics of a reactive system with the set Ψ of possible real-time behaviors to contain more observation sequences than $\Gamma(\Psi)$.

Maximal refinable semantics

It is not hard to see that the refinable specifications of any set Ψ of behaviors are closed under arbitrary unions (as well as under finite intersections). It follows that Ψ has a unique maximal refinable specification — the union of all refinable specifications of Ψ , which can, alternatively, be defined as follows. A set Π of timed state sequences is said to be *maximally closed under stuttering* iff for all timed state sequences ρ ,

$$[\rho] \subseteq [\Pi] \text{ implies } \rho \in \Pi.$$

The *refinement closure* $\Gamma_{\max}(\Pi)$ of Π is the smallest set of timed state sequences that contains Π and is maximally closed under stuttering. It is not hard to see that

$$\Gamma(\Pi) \subseteq \Gamma_t(\Pi) \subseteq \Gamma_{\max}(\Pi)$$

for every set Π of timed state sequences. However, unlike ordinary closure under stuttering, maximal closure under stuttering does not preserve determinism — if Π is deterministic, $\Gamma_{\max}(\Pi)$ need not be. For example, the refinement closure of the deterministic set that contains all timed state sequences of the form

$$(a, \delta) \longrightarrow (b, \delta)$$

with $0 \leq \delta \leq 1$ contains the nondeterministic timed state sequence

$$(a, 0) \longrightarrow (b, 1).$$

The following proposition shows that the refinement closure of Π is the largest set that specifies $[\Pi]$ under stuttering.

Proposition 1.1 (Largest refinable specification) *For any set Π of timed state sequences, the refinement closure $\Gamma_{\max}(\Pi)$ is a refinable specification of the set $[\Pi]$ of behaviors. Moreover, if the set Π' of timed state sequences is equivalent to Π under stuttering, then $\Pi' \subseteq \Gamma_{\max}(\Pi)$.*

Proof of Proposition 1.1 (1) First, we show that $\Gamma_{\max}(\Pi)$ specifies $[\Pi]$ under stuttering; that is, $[\Gamma_{\max}(\Pi)] = [\Pi]$. Since $\Pi \subseteq \Gamma_{\max}(\Pi)$, we have that $[\Pi] \subseteq [\Gamma_{\max}(\Pi)]$. To show the converse containment, suppose, to the contrary, that there is a behavior $\rho \in [\Gamma_{\max}(\Pi)]$ such that $\rho \notin [\Pi]$. Since $\Gamma_{\max}(\Pi)$ is maximally closed under stuttering, $\rho \in \Gamma_{\max}(\Pi)$. Now it is not hard to see that the set

$$\Gamma_{\max}(\Pi) - \{\rho' \in TSS \mid \rho \in \Gamma(\rho')\}$$

is a proper subset of $\Gamma_{\max}(\Pi)$ that contains Π and is maximally closed under stuttering, which contradicts the definition of $\Gamma_{\max}(\Pi)$.

(2) Secondly, we show that every set that specifies $[\Pi]$ under stuttering is contained in $\Gamma_{\max}(\Pi)$. Consider an arbitrary timed state sequence ρ such that $\rho \notin \Gamma_{\max}(\Pi)$ and an arbitrary set $\Pi' \subseteq TSS$ with $[\Pi'] = [\Pi]$; we show that $\rho \notin \Pi'$. Since $\Gamma_{\max}(\Pi)$ is maximally closed under stuttering, there is a behavior $\rho' \in [\rho]$ such that $\rho' \notin [\Pi]$. It follows that $\rho' \notin [\Pi']$ and, therefore, that $\rho \notin \Pi'$.

(3) Thirdly, recall that $\Gamma_{\max}(\Pi)$ and its stuttering closure are equivalent under stuttering. Hence parts (1) and (2) put together imply that $\Gamma_{\max}(\Pi)$ is closed under stuttering and, thus, is a refinable specification of $[\Pi]$. ■

Corollary 1.1 (Refinement closure) For all timed state sequences ρ and sets Π of timed state sequences, $\rho \in \Gamma_{\max}(\Pi)$ iff $[\rho] \subseteq [\Pi]$.

It follows that the refinement closure of a set Ψ of behaviors is the maximal refinable specification of Ψ . Moreover,

$$\Gamma(\Psi) = \Gamma_h(\Psi) = \text{Det}(\Gamma_{\max}(\Psi))$$

for every set Ψ of behaviors. The refinement closure induces a bijection between all sets Ψ of behaviors and all sets Π of timed state sequences that are maximally closed under stuttering:

$$\Pi \begin{array}{c} \xrightarrow{\text{Beh}} \\ \xleftarrow{\Gamma_{\max}} \end{array} \Psi.$$

We shall take the infinite sequences in $\Gamma_{\max}(\Psi)$ as the analog semantics of a reactive system with the set Ψ of possible real-time behaviors. This decision is consistent with the untimed use of the stuttering closure, because ordinary and maximal closure under stuttering collapse in the untimed model: $\Gamma_{\max}(\Psi) = \Gamma(\Psi)$ for every set Ψ of untimed behaviors.

1.2.2 Digitizability

Let Ψ be a set of real-time behaviors. We say that a digital property Π is a *digital specification* of Ψ iff Ψ is the digital denotation of Π . We have argued that the digital semantics $\Pi_N(S)$ of a reactive system S ought to be a digital specification of the set $[\![S]\!]$ of possible real-time behaviors of S . However, as we have already pointed out, not every set of real-time behaviors has a digital specification. Let us now give a necessary and sufficient condition for a set of real-time behaviors to have a digital specification.

Digital specifiability

For this purpose we need the following definitions. A set Π of timed state sequences is called *closed under digitization* iff $[\Pi] \subseteq \Pi$; it is *inversely closed under digitization* iff

$[\rho] \subseteq \Pi$ implies $\rho \in \Pi$ for all timed state sequences ρ over R ; it is *digitizable* iff it is both closed and inversely closed under digitization. Clearly, Π is digitizable iff, for all $\rho \in TSS_R$,

$$\rho \in \Pi \text{ iff } [\rho] \subseteq \Pi.$$

Note that if Π is digitizable, then $[[\Pi]]^{-1} = \Pi$.

Proposition 1.2 (Digital specifiability) *Given a set Ψ of real-time behaviors, the following conditions are equivalent:*

- (1) Ψ has a digital specification.
- (2) Ψ has a digitizable analog specification.
- (3) The refinement closure $\Gamma_{\max}(\Psi)$ is digitizable.
- (4) The stuttering closure $\Gamma_h(\Psi) = \Gamma(\Psi)$ is digitizable.

Proof of Proposition 1.2 (1) \Rightarrow (2) Suppose that Ψ has a digital specification Π ; that is, $\text{Beh}([\Gamma_h(\Pi)]^{-1}) = \Psi$. We show that $[\Gamma_h(\Pi)]^{-1}$ is a digitizable analog specification of Ψ . Lemma 1.2 implies that $[\Gamma_h(\Pi)]^{-1}$ is closed under stuttering (although it is in general not maximally closed under stuttering). To see that $[\Gamma_h(\Pi)]^{-1}$ is digitizable, observe that

$$\rho \in [\Gamma_h(\Pi)]^{-1} \text{ iff } [\rho] \subseteq \Gamma_h(\Pi) \text{ iff } [[\rho]] \subseteq \Gamma_h(\Pi) \text{ iff } [\rho] \subseteq [\Gamma_h(\Pi)]^{-1}$$

for all $\rho \in TSS_R$.

(2) \Rightarrow (3) Suppose that $\Pi \subseteq TSS_R$ is digitizable and closed under stuttering. By Corollary 1.1, we have that $\Gamma_{\max}([\Pi]) = \Gamma_{\max}(\Pi)$; thus it suffices to show that $\Gamma_{\max}(\Pi)$ is digitizable. We show that $\rho \in \Gamma_{\max}(\Pi)$ iff $[\rho] \subseteq \Gamma_{\max}(\Pi)$ for an arbitrary $\rho \in TSS_R$. By Corollary 1.1, it suffices to show that $[\rho] \subseteq [\Pi]$ iff $[[\rho]] \subseteq [\Pi]$.

First we suppose that $[\rho] \subseteq [\Pi]$ and show that $[[\rho]] \subseteq [\Pi]$. Consider an arbitrary $0 \leq \epsilon < 1$ and an arbitrary real-time behavior $\rho' \in [[\rho]]_\epsilon$; we show that $\rho' \in [\Pi]$. It suffices to show that $\rho' \in \Pi$ or, because Π is digitizable, that $[\rho']_\epsilon \in \Pi$ for an arbitrary $0 \leq \epsilon' < 1$. It is not hard to see that there is some real-time behavior $\rho'' \in [\rho]$ such that $[\rho'']_{\epsilon'} = [\rho']_{\epsilon'}$. Since $[\rho] \subseteq [\Pi]$, we have that $\rho'' \in [\Pi]$ and, as Π is closed under stuttering, $\rho'' \in \Pi$. Since Π is digitizable, also $[\rho'']_{\epsilon'} \in \Pi$, as we wanted to show.

Now we suppose that $[\rho] \not\subseteq [\Pi]$ — that is, $\rho' \in [\rho]$ and $\rho' \notin [\Pi]$ for some real-time behavior ρ' — and show that $[[\rho]] \not\subseteq [\Pi]$. In this case, $\rho' \notin \Pi$ and, as Π is digitizable,

$[\rho']_\epsilon \notin \Pi$ for some $0 \leq \epsilon < 1$. Since Π is closed under stuttering, $h[\rho']_\epsilon \notin \Pi$ and, therefore, $h[\rho']_\epsilon \notin [\Pi]$. Hence $[[\rho']_\epsilon] \not\subseteq [\Pi]$.

(3) \Rightarrow (4) Recall that $\Gamma(\Psi) = \text{Det}(\Gamma_{\text{max}}(\Psi))$. Assume that $\Pi \subseteq TSS$ is digitizable; we show that then $\text{Det}(\Pi)$ is digitizable as well. To see that $\rho \in \text{Det}(\Pi)$ implies $[\rho] \subseteq \text{Det}(\Pi)$, observe that whenever $\rho \in TSS_R$ is deterministic, then so is $[\rho]$. To see that $[\rho] \subseteq \text{Det}(\Pi)$ implies $\rho \in \text{Det}(\Pi)$, note that if ρ is not deterministic, then there is some $0 \leq \epsilon < 1$ such that $[\rho]_\epsilon$ is not deterministic either.

(4) \Rightarrow (1) Suppose that $\Gamma(\Psi) = \Gamma_h(\Psi)$ is digitizable. To show that $[\Gamma_h(\Psi)]$ is a digital specification of Ψ , it suffices to show that $[\Gamma_h([\Gamma_h(\Psi)])]^{-1} = \Gamma_h(\Psi)$, because $\Gamma_h(\Psi)$ specifies Ψ under stuttering. First observe that the digitizability of $\Gamma_h(\Psi)$ implies that $[\Gamma_h(\Psi)] \subseteq \Gamma_h(\Psi)$ and, therefore, that $\Gamma_h([\Gamma_h(\Psi)]) \subseteq \Gamma_h(\Psi)$. Now consider an arbitrary $\rho \in TSS_R$; then

$$\rho \in [\Gamma_h([\Gamma_h(\Psi)])]^{-1} \text{ iff } [\rho] \subseteq \Gamma_h([\Gamma_h(\Psi)]) \text{ iff } [\rho] \subseteq \Gamma_h(\Psi) \text{ iff } \rho \in \Gamma_h(\Psi),$$

because $\Gamma_h(\Psi)$ is digitizable. ■

It follows that a reactive system has a suitable digital semantics iff its analog semantics is digitizable. We shall restrict ourselves to these systems — the *digitizable* ones.

Clock-independent specifiability

To define the digital semantics of digitizable systems, we look for a one-to-one correspondence between digital properties and sets of possible behaviors of digitizable systems. There are, however, in general infinitely many digital specifications of the set of possible behaviors of a digitizable system. Thus we designate, for every set Ψ of real-time behaviors that has a digital specification, a particular digital specification of Ψ as the digital semantics of all reactive systems S with $[S] = \Psi$. We like the formal semantics of a reactive system S to be independent of the clock model; that is, we want to define the digital semantics $\Pi_N(S)$ of S simply as the set of all timed state sequences over N in the analog semantics $\Pi_R(S)$ of S :

$$\Pi_N(S) = \Pi_R(S) \cap TSS_N.$$

This convention would allow us to describe the formal semantics of any particular reactive system uniformly — independent of the clock model — as the set of all timed state sequences that satisfy certain requirements. A look at the system with the single behavior

$$(a, 0.5) \longrightarrow (b, 0.5)$$

shows that such a definition is in general highly inappropriate; yet it proves workable for all digitizable systems. The following proposition shows that for all digitizable systems S , the set of timed state sequences over N in the analog semantics $\Gamma_{max}([S])$ of S is indeed a digital specification of $[S]$. We find it convenient to write Π_N for $\Pi \cap TSS_N$, for any set Π of timed state sequences.

Proposition 1.3 (Clock-independent specifiability) *If the set Π of precisely timed state sequences is digitizable and closed under stuttering, then $[\Pi] = \Pi_N$ and this set $[\Pi]$ of digitally timed state sequences is a digital specification of the set $[\Pi]$ of real-time behaviors.*

Proof of Proposition 1.3 (1) First observe that $\Pi_N \subseteq [\Pi]$ for all $\Pi \subseteq TSS_R$. Now suppose that Π is closed under digitization. Then

$$[\Pi] = [\Pi]_N \subseteq \Pi_N.$$

(2) To see that $[\Pi]$ is a digital specification of $[\Pi]$ if Π is both digitizable and closed under stuttering, confer part (4) \Rightarrow (1) of the proof of Proposition 1.2. ■

Recall that if Π is closed under stuttering, then $[\Pi] = Beh(\Pi)$, and if Π is digitizable, then $\Pi = [[\Pi]]^{-1}$. It follows that the set $[\Pi] = \Pi_N$ of the previous proposition specifies, in the digital-clock model, the set $Beh([\Pi]^{-1})$ of real-time behaviors. This observation reveals the following bijection between all sets Ψ of real-time behaviors with digital specifications and some sets Π of digitally timed state sequences:

$$\Pi \begin{array}{c} \xrightarrow{Beh([\]^{-1})} \\ \xleftarrow{[\Gamma_{max}(\)] = \Gamma_{max}(\)_N} \end{array} \Psi.$$

We take the infinite sequences in $\Gamma_{max}(\Psi)_N$ as the digital semantics of a reactive system with the set Ψ of possible real-time behaviors. Proposition 1.3 gives us also another way of looking at our definition of digital semantics. Consider a digitizable system S with the set $[S] = \Psi$ of possible real-time behaviors, the analog semantics $\Pi_R(S)$ of S — a digitizable analog specification of Ψ — and the digital semantics $\Pi_N(S)$ of S — a digital specification of Ψ . Since $\Pi_N(S) = [\Pi_R(S)]$,

$$\rho \in \Pi_R(S) \text{ iff } [\rho] \subseteq \Pi_N(S)$$

for all precisely timed state sequences ρ . This property of the digital semantics of S can be intuitively interpreted as the following two requirements on $\Pi_N(S)$. Given a precisely timed observation sequence ρ ,

1. If ρ observes a real-time behavior of S , then all digital clocks think so. Formally,

$$[\Pi_R(S)] \subseteq \Pi_N(S).$$

2. If all digital clocks think that ρ observes a real-time behavior of S , then they are correct (or, equivalently, if ρ does not observe a real-time behavior of S , then some digital clock thinks so). Formally, $[\rho] \subseteq \Pi_N(S)$ implies $\rho \in \Pi_R(S)$. This demand asserts, in particular, that $\Pi_N(S) \subseteq \Pi_R(S)$ and, consequently, that

$$\Pi_N(S) \subseteq [\Pi_R(S)].$$

It is not hard to see that since the analog semantics $\Pi_R(S)$ of a system S is closed under stuttering, so is the digital semantics $\Pi_N(S) = (\Pi_R(S))_N$ in the digital-clock model. The following proposition implies that the digital semantics is, in fact, maximally closed under stuttering; it shows that our choice of digital semantics is, consistent with the analog and untimed cases, again a maximal one.

Proposition 1.4 (Largest digital specification) *For every set Ψ of real-time behaviors and every digital specification $\Pi \subseteq TSS_N$ of Ψ , we have $\Pi \subseteq \Gamma_{max}(\Psi)_N$.*

Proof of Proposition 1.4 We suppose that $[\Pi]_N = \Psi$, consider an arbitrary digitally timed state sequence $\rho \in \Pi$, and show that $\rho \in \Gamma_{max}(\Psi)$. By Corollary 1.1, it suffices to show that $[\rho] \subseteq \Psi$. Consider an arbitrary real-time behavior $\rho' \in [\rho]$; we show that $\rho' \in \Psi$. Since $\rho \in \Pi$, we have that $\rho' \in \Gamma_h(\Pi)$. Then,

$$[\rho'] \subseteq [\Gamma_h(\Pi)] \subseteq \Gamma_h([\Pi]) = \Gamma_h(\Pi),$$

which implies that $\rho' \in [\Gamma_h(\Pi)]^{-1}$. As $Beh([\Gamma_h(\Pi)]^{-1}) = \Psi$, also $\rho' \in \Psi$ as desired. ■

1.2.3 Safety, liveness, and operationality

Reactive systems define real-time properties. In the analog-clock model, the system S defines the analog property $\Pi_R(S)$; in the digital-clock model, S defines the digital property

$\Pi_N(S) = (\Pi_R(S))_N$; in the untimed model, the untimed property $\Pi_1(S)$. If the time domain is immaterial, we omit the subscript as usual and say that the system S defines (specifies) the property $\Pi(S) \subseteq TSS^w$. On the other hand, S is said to satisfy (implement) a property $\Pi \subseteq TSS^w$ iff

$$\Pi(S) \subseteq \Pi.$$

Verification of S is the task of proving that S satisfies certain properties. It is useful to classify properties of reactive systems into two categories — safety properties and liveness properties — because they require fundamentally different means for their specification and verification [76]. Thus we extend the standard safety-liveness classification of untimed properties to real-time properties.

Real-time safety and liveness

- A *safety* property stipulates that “nothing bad” will happen, ever, during the execution of a system. If “something bad” were to happen during the execution, it would have to happen within a finite number of observations. Thus we can formalize safety as follows:

A set Π of infinite timed state sequences is a (*real-time*) *safety* property iff for all infinite timed state sequences ρ , whenever every finite prefix of ρ can be extended to a sequence in Π , then $\rho \in \Pi$.

- A *liveness* property stipulates that “something good” will happen, eventually, during the execution of a system. If “nothing good” were to happen during the execution, an irremediable situation would have to be reached within a finite number of observations. Thus we can formalize liveness as follows:

A set Π of infinite timed state sequences is a (*real-time*) *liveness* property iff every finite timed state sequence can be extended to a sequence in Π .

These definitions generalize the corresponding untimed notions of safety, as defined by Alpern, Demers, and Schneider [4], and liveness, as defined by Alpern and Schneider [5]: in the untimed model, Π is a real-time safety property iff Π^- is a safety property, and Π is a real-time liveness property iff Π^- is a liveness property. In fact, we show that this correlation holds, independent of the clock model, for all time-invariant properties. A set Π

of timed state sequences is called *time-invariant* iff, for all timed state sequences $\rho = (\sigma, T)$ and $\rho' = (\sigma, T')$, if $\rho \in \Pi$ then $\rho' \in \Pi$; that is,

$$\rho \in \Pi \text{ iff } \rho^- \in \Pi^-$$

for all $\rho \in TSS$. Note that in the untimed model, every property is time-invariant.

Proposition 1.5 (Time-invariant safety and liveness) *If Π is a time-invariant real-time property, then (1) Π is a real-time safety property iff Π^- is a safety property, and (2) Π is a real-time liveness property iff Π^- is a liveness property.*

Proof of Proposition 1.5 (1) To show that the safety of Π implies the safety of Π^- , consider an arbitrary infinite state sequence σ and suppose that every finite prefix σ^i , for $i \geq 0$, of σ can be extended to a sequence $\sigma_i \in \Pi^-$; we show that $\sigma \in \Pi^-$. Let T be any infinite time sequence that satisfies the *monotonicity* and *progress* conditions. Since Π is time-invariant, $(\sigma_i, T) \in \Pi$ for all $i \geq 0$ and, as Π is safe, also $(\sigma, T) \in \Pi$, which implies that $\sigma \in \Pi^-$.

To show that the safety of Π^- implies the safety of Π , consider an arbitrary infinite timed state sequence ρ and suppose that every finite prefix ρ^i of ρ can be extended to a sequence $\rho_i \in \Pi$; we show that $\rho \in \Pi$. Since Π is time-invariant, it suffices to show that $\rho^- \in \Pi^-$. Since Π^- is safe, it suffices to show that every finite prefix of ρ^- can be extended to a sequence in Π^- , which follows from our assumption (extend $(\rho^-)^i$ to ρ_i^-).

(2) To show that the liveness of Π implies the liveness of Π^- , consider an arbitrary finite state sequence σ ; we show that σ can be extended to a sequence in Π^- . Let ρ be any finite timed state sequence with $\rho^- = \sigma$. Since Π is live, ρ can be extended to an infinite timed state sequence $\rho' \in \Pi$. Clearly, $\rho'^- \in \Pi^-$ is an extension of σ .

To show that the liveness of Π^- implies the liveness of Π , consider an arbitrary finite timed state sequence ρ ; we show that ρ can be extended to a sequence in Π . Since Π^- is live, ρ^- can be extended to an infinite state sequence $\sigma \in \Pi^-$. Let ρ' be any infinite timed state sequence extending ρ such that $\rho'^- = \sigma$. Since Π is time-invariant, $\rho' \in \Pi$ as desired.

■

Moreover, the safety and liveness of any digital property that is defined by a reactive system S is determined by the analog semantics of S . This is shown by the following

proposition, which states that the safety and liveness of digitizable properties is invariant under digitization.

Proposition 1.6 (Digitizable safety and liveness) *Let Π be a digitizable analog property.*

- (1) Π is a safety property iff Π_N is a safety property.
- (2) If Π is a liveness property, then so is Π_N .

Proof of Proposition 1.6 (1) It is not hard to see that the safety of Π implies the safety of Π_N for all $\Pi \subseteq TSS_{\mathbb{R}}^{\omega}$.

To see that the safety of Π_N implies the safety of Π , consider an arbitrary infinite sequence $\rho \notin \Pi$; we show that there is a finite prefix ρ^i of ρ that cannot be extended to an infinite sequence in Π . Since Π is digitizable, there is some $0 \leq \epsilon < 1$ such that $[\rho]_{\epsilon} \notin \Pi_N$. Since Π_N is safe, there is some finite prefix $[\rho]_{\epsilon}^i$ of $[\rho]_{\epsilon}$ that cannot be extended to an infinite sequence in Π_N . It follows ρ^i cannot be extended to an infinite sequence $\rho' \in \Pi$: otherwise the extension $[\rho']_{\epsilon}$ of $[\rho]_{\epsilon}^i$ would be in Π_N , because Π is digitizable.

(2) To see that the liveness of Π implies the liveness of Π_N , consider an arbitrary finite sequence $\rho \in TSS_N^{\omega}$. Since Π is live, ρ can be extended into an infinite sequence $\rho' \in \Pi$ and, as Π is closed under digitization, the extension $[\rho']_{\epsilon}$ of ρ is in Π_N for any $0 \leq \epsilon < 1$. ■

In general there is, however, no obvious correspondence between timed and untimed safety and liveness. Consider, for example, the bounded-response property Π_{δ}^{\diamond} that contains an infinite timed state sequence (σ, T) iff for all $i \geq 0$, whenever $\sigma_i = a$, then $\sigma_j = b$ and $T_j \leq T_i + \delta$ for some $j \geq i$; that is, every observation that shows state a is followed by an observation that shows state b within time δ . Then $\Pi_{\delta}^{\diamond-}$ is the untimed, unbounded response property that specifies that every state a is, eventually, followed by a state b ; it contains an infinite sequence σ of states iff for all $i \geq 0$, whenever $\sigma_i = a$, then $\sigma_j = b$ for some $j \geq i$. Since every finite prefix of a state sequence can be extended to contain a state b , the response property $\Pi_{\delta}^{\diamond-}$ is a liveness property. On the other hand, in both the analog-clock and the digital-clock model the bounded-response property Π_{δ}^{\diamond} is not live, because the finite prefix

$$(a, 0) \longrightarrow (b, \delta + 1)$$

cannot be extended to a timed state sequence in Π_δ^\diamond . Note that Π_δ^\diamond is, in fact, safe in both the analog-clock and the digital-clock model: if an infinite timed state sequence $\rho = (\sigma, T)$ is not in Π_δ^\diamond , then it contains an observation $\rho_i = (a, T_i)$ and, because time progresses, a later observation ρ_j , for $j \geq i$, such that $T_j > T_i + \delta$ and $\sigma_k \neq b$ for all $i \leq k < j$; moreover, the finite prefix ρ^j cannot be extended to an infinite sequence in Π_δ^\diamond . The “bad thing” that is not supposed to happen is that, after observing state a , time δ passes without an observation of state b .

It follows that the traditional safety-liveness classification of properties, which considers only state components, does not fit for real-time properties. Time cannot be ignored, because its implicit “liveness,” as guaranteed by the *progress* condition on timed state sequences, shifts the spectrum of real-time properties towards the safety side — an observation that has been made repeatedly [79, 85, 115, 116]. It is precisely this phenomenon that has been captured formally by our definitions of *real-time* safety and liveness: instead of looking at all infinite sequences of observations, we have restricted ourselves, a priori, to *timed state sequences*, which satisfy the *monotonicity* and *progress* conditions on time.

In the following segment, we relate the timed and untimed classifications of properties by giving a general topological characterization of safety and liveness under unspecified assumptions about states and time. For this purpose we need to consider arbitrary infinite sequences of observations rather than just timed state sequences. We use the following local conventions for the remainder of this subsection. Let O be a set of observations. If Φ is a subset of the set O^ω of all infinite observation sequences, we say that every set $\Pi \subseteq \Phi$ is a Φ -*property*. Thus TSS^ω -properties are *real-time* properties; we refer to O^ω -properties as *unconditional*. Whenever the value of the parameter Φ is not mentioned explicitly, it is unspecified, rather than TSS^ω as in the other parts of this thesis. For example, in the remainder of this subsection, a “property” may be any subset of O^ω .

Relative safety and liveness

Suppose that, for whatever reason, all legal observation sequences satisfy certain requirements on states and time. These requirements can be characterized by a property $\Phi \subseteq O^\omega$, which contains exactly the legal observation sequences. If Φ is a condition on the times of observations, we refer to it as a *timing assumption*. We have encountered the following timing assumptions:

Time domain The timing assumption $O_{TIME}^w \subseteq O^w$ contains the observation sequences over the time domain $TIME$.

Monotonicity We say that an observation sequence is *monotonic* iff it satisfies the *monotonicity* condition on time. The timing assumption $O_{mon}^w \subseteq O^w$ contains the monotonic observation sequences.

Progress We say that an observation sequence is *divergent* iff it satisfies the *progress* condition on time. The timing assumption $O_{div}^w \subseteq O^w$ contains the divergent observation sequences.

Real time The timing assumption $TSS^w = O_{mon}^w \cap O_{div}^w$ contains the infinite timed state sequences.

Analog clock The timing assumption $TSS_R^w = TSS^w \cap O_R^w$ contains the infinite precisely timed state sequences.

Digital clock The timing assumption $TSS_N^w = TSS^w \cap O_N^w$ contains the infinite digitally timed state sequences.

If we restrict our consideration to infinite sequences from Φ , we obtain the following notions of safety and liveness *relative* to the property Φ :

- $\Pi \subseteq \Phi$ is a *safety* property *relative* to $\Phi \subseteq O^w$ iff for all $\sigma \in \Phi$, whenever every finite prefix of σ can be extended to a sequence in Π , then $\sigma \in \Pi$.
- $\Pi \subseteq \Phi$ is a *liveness* property *relative* to $\Phi \subseteq O^w$ iff every finite prefix of a sequence in Φ can be extended to a sequence in Π .

Thus *real-time* safety and liveness are safety and liveness relative to the timing assumption TSS^w , and conventional untimed safety and liveness are safety and liveness relative to TSS^w . We refer to safety and liveness relative to O^w as *unconditional* safety and liveness; it is not hard to see that a property Π is unconditionally safe (live) iff the untimed property Π^- is safe (live).

Now we can classify the timing assumptions given above:

Time domain O_{TIME}^w is an unconditional safety property for every time domain $TIME$.

Monotonicity O_{mon}^ω is an unconditional safety property.

Progress O_{div}^ω is an unconditional liveness property.

Real time TSS^ω is neither unconditionally safe nor unconditionally live. It is safe relative to O_{div}^ω and live relative to O_{mon}^ω .

Analog clock TSS_R^ω is neither unconditionally safe nor unconditionally live; it is safe relative to TSS^ω .

Digital clock TSS_N^ω is neither unconditionally safe nor unconditionally live; it is safe relative to TSS_R^ω .

There is a natural topology on O^ω , the Cantor topology on infinite strings, in which the unconditional safety properties are exactly the closed sets, and the unconditional liveness properties are exactly the dense sets [4]. It follows immediately that only O^ω itself is both an unconditional safety and an unconditional liveness property. The Cantor topology on O^ω induces a topological subspace on $\Phi \subseteq O^\omega$, which is called the *relativization* of the Cantor topology on O^ω to Φ [70]: the open sets of the relative topology are taken to be the intersections of Φ with the open sets of the topology on O^ω . The following lemma shows that the properties that are safe relative to Φ are exactly the closed sets of the relative topology, and the properties that are live relative to Φ are exactly the dense sets of the relative topology. Note that since unconditional safety properties are closed under arbitrary intersections, we can define the *closure* $\bar{\Pi}$ of a property Π as the smallest unconditional safety property containing Π .

Lemma 1.3 (Relative safety and liveness) *Let $\Pi \subseteq \Phi \subseteq O^\omega$.*

- (1) Π is a safety property relative to Φ iff $\bar{\Pi} \cap \Phi \subseteq \Pi$.
- (2) Π is a liveness property relative to Φ iff $\Phi \subseteq \bar{\Pi}$.

Proof of Lemma 1.3 First observe that an infinite sequence $\rho \in O^\omega$ is in the closure of a property $\Pi \subseteq O^\omega$ (that is, $\rho \in \bar{\Pi}$) iff every finite prefix of ρ can be extended to an infinite sequence in Π . Then apply this observation to the definitions of relative safety and relative liveness. ■

It follows that Π is safe relative to Φ iff $\Pi = \Pi_s \cap \Phi$ for some unconditional safety property Π_s . In particular, if the property $\Pi = \Pi_s \cap \Pi_l$ is given as the intersection of an unconditional safety property Π_s and an unconditional liveness property Π_l , then Π is safe relative to Π_l .

It is convenient to extend the notions of safety and liveness relative to a property Φ to properties that are not necessarily subsets of Φ : we say that $\Pi \subseteq O^\omega$ is a safety (liveness) property relative to $\Phi \subseteq O^\omega$ iff $\Pi \cap \Phi$ is safe (live) relative to Φ . Clearly, unconditional safety properties are, in this sense, safe relative to any property Φ . More generally:

Proposition 1.7 (Downward preservation of safety) *Suppose that $\Phi_1 \subseteq \Phi_2 \subseteq O^\omega$. If $\Pi \subseteq O^\omega$ is a safety property relative to Φ_2 , then it is also a safety property relative to Φ_1 .*

Proof of Proposition 1.7 Let $\Phi_1 \subseteq \Phi_2$. First observe that the closure operator is monotonic; that is, $\Pi \subseteq \Phi$ implies $\overline{\Pi} \subseteq \overline{\Phi}$ for all $\Pi, \Phi \subseteq O^\omega$. In particular, we have that $\overline{\Pi \cap \Phi_1} \subseteq \overline{\Pi \cap \Phi_2}$.

By part (1) of Lemma 1.3, we may assume that

$$\overline{(\Pi \cap \Phi_2)} \cap \Phi_2 \subseteq \Pi \cap \Phi_2$$

and need to show that, then,

$$\overline{(\Pi \cap \Phi_1)} \cap \Phi_1 \subseteq \Pi \cap \Phi_1.$$

The derivation is simple. ■

This proposition shows that every unconditional safety property remains safe relative to any timing assumption. The converse of Proposition 1.7 holds only in a very restricted case:

Proposition 1.8 (Upward preservation of safety) *Suppose that $\Pi \subseteq \Phi_1 \subseteq \Phi_2 \subseteq O^\omega$. If Π is a safety property relative to Φ_1 and Φ_1 is a safety property relative to Φ_2 , then Π is a safety property relative to Φ_2 .*

Proof of Proposition 1.8 Again, use part (1) of Lemma 1.3 and the monotonicity of the closure operator. ■

In general, properties become "safer" if they are viewed relative to stronger (i.e., more restrictive) properties: a property that is not an unconditional safety property may be safe relative to another property. We have already given an interesting example of such a property that is shifted "towards safety": the bounded-response property Π_b^\diamond is unconditionally live, but safe relative to TSS^ω . Note that relative to the weaker timing assumption O_{mon}^ω , the bounded-response property Π_b^\diamond is neither live nor safe; it is not safe relative to O_{mon}^ω , because it contains all monotonic observation sequences of the form

$$(a, 0) \longrightarrow \dots \longrightarrow (a, 0) \longrightarrow (b, 0) \longrightarrow \dots,$$

without containing the monotonic sequence

$$(a, 0) \longrightarrow (a, 0) \longrightarrow (a, 0) \longrightarrow \dots.$$

Our timing assumption TSS^ω causes properties to shift towards safety, because it includes the liveness condition O_{liv}^ω . The class of properties that are safe relative to TSS^ω includes many other important real-time properties that are unconditional liveness properties; that is, all the liveness they stipulate is subsumed by the progress of time. We will use this fact extensively in the following way. Suppose that Π is an unconditional liveness property and safe relative to TSS^ω . Since any description of the unconditional liveness property Π defines, under our assumptions of monotonicity and progress of time, the real-time safety property $\Pi \cap TSS^\omega$, we can

1. Specify Π by methods for specifying liveness properties: liveness-type specifications of, say, bounded response are often more intuitive than safety-type specifications.
2. Verify Π by methods for verifying safety properties: safety-type arguments are often simpler than liveness-type arguments.

Operationality

Recall that the analog semantics $\Pi_R(S)$ of any reactive system S is required to be transparent — that is, $Beh(\Pi_R(S))$ is the set of possible behaviors of S — and closed under stuttering. Since the system S is essentially a machine that may, at any point in time, either change its state or wait and do nothing, the presentation of its semantics $\Pi_R(S)$ ought

to give a recipe for the incremental generation of all possible execution sequences of S — the stuttering closure of $Beh(\Pi_R(S))$:

$$\Gamma(Beh(\Pi_R(S))) = Det(\Pi_R(S)).$$

In other words, we want to be able to interpret the description of the reactive system S operationally. For this purpose, we have to isolate the safety and liveness components of $Det(\Pi_R(S))$. For suppose that $Det(\Pi_R(S))$ is defined as the intersection of an unconditional safety property Π_s and an unconditional liveness property Π_l :

$$Det(\Pi_R(S)) = \Pi_s \cap \Pi_l \cap Det(TSS_R^w).$$

Then we may be able to view the system S as a machine that generates, step by step, safe execution sequences in $\Pi_s \cap Det(TSS_R^w)$ and “eventually” satisfies the liveness requirement Π_l . However, not every pair (Π_s, Π_l) allows this interpretation. The problem is that the safety part may permit the generation of deterministic finite timed state sequences from which either the liveness requirement cannot be satisfied or time cannot advance.

We formalize the notion of operability for arbitrary timing assumptions Φ . A pair (Π_s, Π_l) is said to define the Φ -property $\Pi \subseteq \Phi$ *congruously relative to* $\Phi \subseteq O^w$ iff

- (1) $\Pi = \Pi_s \cap \Pi_l \cap \Phi$,
- (2) Π_s is safe relative to Φ and Π_l is live relative to Φ , and
- (3) every finite prefix of a sequence in $\Pi_s \cap \Phi$ can be extended to an infinite sequence in Π .

Condition (3) ensures that the safety part of a congruous definition is complete: the liveness part does not preclude any safe prefixes. The definition of a property is called *unconditionally congruous* iff it is congruous relative to O^w ; it is (*real-time*) *congruous* iff it is congruous relative to $Det(TSS^w)$. Congruity generalizes an untimed concept that has been named repeatedly: a pair (Π_s, Π_l) is congruous in the untimed model iff the pair (Π_s^-, Π_l^-) is *machine closed* according to Abadi and Lamport [1]; *feasible* according to Apt, Francez, and Katz [13]; or Π_l^- is *live with respect to* Π_s^- according to Dederichs and Weber [32].

Congruous definitions of deterministic properties describe reactive systems, because they can be executed: if (Π_s, Π_l) is congruous relative to $Det(\Phi)$, then a machine that incrementally generates safe execution sequences in $\Pi_s \cap Det(\Phi)$ will never reach an irremediable

situation from which the liveness conditions in $\Pi_l \cap Det(\Phi)$ cannot be satisfied. This is because the defined property $\Pi = \Pi_s \cap \Pi_l \cap Det(\Phi)$ is live relative to $\Pi_s \cap Det(\Phi)$. On the other hand, a machine trying to execute an incongruous definition without look-ahead may "paint itself into a corner" from which no legal continuation is possible. In particular, the real-time congruity of a system description ensures that the system cannot prevent time from advancing.

Thus we say that the formal semantics $\Pi(S)$ of a reactive system S is given *operationally* iff its deterministic part $Det(\Pi(S)) \subseteq Det(TSS^\omega)$ is defined congruously — by a pair (Π_s, Π_l) that defines the deterministic real-time property

$$Det(\Pi(S)) = \Pi_s \cap \Pi_l \cap Det(TSS^\omega)$$

congruously. We require that the analog semantics of every reactive system is given operationally. This demand does not restrict the real-time properties under consideration: we will prove that for every assumption $\Phi \subseteq O^\omega$, every property $\Pi \subseteq \Phi$ can be defined congruously relative to Φ . Alpern and Schneider showed that every untimed property is the intersection of an untimed safety property and an untimed liveness property [5]. It is well-known that they have given a construction that actually proves the stronger result that every untimed property has a congruous definition. We generalize this main result about the untimed safety-liveness classification to safety, liveness, and congruity relative to any timing assumption.

Theorem 1.1 (Existence of congruous definitions) *For all $\Phi \subseteq O^\omega$, every property $\Pi \subseteq \Phi$ has a definition that is congruous relative to Φ .*

Proof of Theorem 1.1 Let $\Pi_s = \bar{\Pi}$ and $\Pi_l = \neg((\Pi_s \cap \Phi) - \Pi)$; then Π_s is unconditionally safe. Alternatively, let $\Pi_s = \bar{\Pi} \cap \Phi$ and $\Pi_l = \neg(\Pi_s - \Pi)$; then $\Pi_s \subseteq \Phi$. We show that (Π_s, Π_l) defines Π congruously relative to Φ in either case; in fact, Π_l is unconditionally live.

It is not hard to see that $\Pi = \Pi_s \cap \Pi_l \cap \Phi$ and that $\Pi_s \cap \Phi \subseteq \bar{\Pi}$ — that is, every finite prefix of a sequence in $\Pi_s \cap \Phi$ can be extended to a sequence in $\bar{\Pi}$. Proposition 1.7 implies that $\Pi_s = \bar{\Pi}$, and thus also $\Pi_s = \bar{\Pi} \cap \Phi$, is safe relative to Φ . It remains to be shown that Π_l is live relative to Φ or, by part (2) of Lemma 1.3, that

$$\Phi \subseteq \overline{\neg((\bar{\Pi} \cap \Phi) - \Pi)} \cap \Phi.$$

Since $\Pi \subseteq \Phi$, this condition is equivalent to

$$\Phi \subseteq \overline{\Pi \cup (\Phi - \overline{\Pi})}.$$

We can derive both containments

$$\begin{aligned} \overline{\Pi} \cap \Phi &\subseteq \overline{\Pi \cup (\Phi - \overline{\Pi})}, \\ \neg \overline{\Pi} \cap \Phi &\subseteq \overline{\Pi \cup (\Phi - \overline{\Pi})}, \end{aligned}$$

using the monotonicity of the closure operator. ■

To complete our discussion of the semantics of reactive systems, we show that if the description of a system S gives its analog semantics $\Pi_R(S) = \Pi_s \cap \Pi_t \cap TSS_R^\omega$ operationally, then it gives its digital semantics $\Pi_N(S) = (\Pi_R(S))_N$ operationally as well. It follows from the following proposition that for digitizable systems S , the pair (Π_s, Π_t) is a congruous definition of the set

$$Det(\Pi_N(S)) = Det(\Pi_R(S))_N$$

of digitally timed execution sequences of S .

Proposition 1.9 (Digitizable operability) *Let $\Pi_s, \Pi_t \subseteq O^\omega$ such that (Π_s, Π_t) is a congruous definition of the analog property $\Pi = \Pi_s \cap \Pi_t \cap TSS_R^\omega$. If Π is digitizable, then (Π_s, Π_t) defines the digital property Π_N congruously.*

Proof of Proposition 1.9 By Proposition 1.6, it suffices to show that every finite prefix of a digitally timed state sequence in Π_s can be extended to an infinite digitally timed state sequence in $\Pi_s \cap \Pi_t$. Consider the finite prefix $\rho^i \in TSS_N$ of $\rho \in \Pi_s$. Since (Π_s, Π_t) defines Π congruously, there is an infinite precisely timed extension ρ' of ρ^i with $\rho' \in \Pi$. Since Π is closed under digitization, also $[\rho']_\epsilon \in \Pi$ for any $0 \leq \epsilon < 1$. Note that $[\rho']_\epsilon$ is an infinite digitally timed extension of ρ^i in $\Pi_s \cap \Pi_t$. ■

1.3 Real-time Systems, Specifications, and Verification

Let S be a reactive system whose set of possible real-time behaviors is $\llbracket S \rrbracket$ and let ϕ be a specification that is satisfied by the real-time behaviors in $\llbracket \phi \rrbracket$. Verification of S with respect to ϕ amounts to checking the containment

$$\llbracket S \rrbracket \stackrel{?}{\subseteq} \llbracket \phi \rrbracket$$

of sets of real-time behaviors. Systems will be given as expressions of an implementation language; specifications as expressions of a specification language. Now, at the end of this foundational chapter, we can summarize our demands on the implementation language and the specification language, and justify both the analog-clock model and the digital-clock model for proving containment of sets of real-time behaviors. Then, in the remainder of this thesis, we will introduce concrete languages that meet our constraints and concrete techniques that solve instances of the verification problem.

1.3.1 Implementation languages

We have put very stringent demands on the *implementation* language. Let Ψ be the set of real-time behaviors of a digitizable system S ; then $\Psi_1 = h(\Psi^-)$ is the set of untimed behaviors of S . Since we require system descriptions to be transparent, refinable, independent of the clock model, and executable, we have agreed that an expression of the implementation language that describes the system S ought to define, in the analog-clock model, the analog property

$$\Pi_R(S) = \Gamma_{max}(\Psi)$$

operationally; in the digital-clock model, it ought to define the digital property

$$\Pi_N(S) = (\Pi_R(S))_N;$$

in the untimed model, the untimed property

$$\Pi_1(S) = \Gamma(h(\Psi^-)) = \Gamma_h(\Psi^-) = \Gamma_{max}(\Psi^-).$$

The following diagram gives the complete semantical picture for a digitizable system S whose possible real-time behaviors are Ψ :

$$\begin{array}{ccccc}
 \Pi_N(S) & \begin{array}{c} \xrightarrow{[]^{-1}} \\ \xleftarrow{[]=()_N} \end{array} & \Pi_R(S) & \begin{array}{c} \xrightarrow{Beh} \\ \xleftarrow{\Gamma_{max}} \end{array} & \Psi \\
 & & \downarrow \Gamma_h(-) & & \downarrow h(-) \\
 & & \Pi_1(S) & \begin{array}{c} \xrightarrow{Beh} \\ \xleftarrow{\Gamma=\Gamma_{max}} \end{array} & \Psi_1
 \end{array}$$

Similar conditions can be required of the *specification* language. Indeed, this approach of using high-level system descriptions as specifications lends itself to many useful verification

techniques such as stepwise refinement [77, 86]: verification methods that are based on refinement mappings show that one system S_1 — the *implementation* — implements another system S_2 — the *specification*. These methods can be used for the hierarchical verification of a chain of system descriptions

$$[[S_1]] \subseteq [[S_2]] \subseteq \dots \subseteq [[S_n]],$$

in which every implementation contains more detail than its specification. We shall, however, study an alternative — or rather, complementary — approach that considers specifications to be logical propositions instead of system descriptions: we require that specifications can be combined by boolean operators rather than that they are refinable and executable. You may think of the logical approach as addressing the final link

$$[[S_n]] \subseteq [\phi]$$

in a verification chain, where the top-level specification ϕ is given as a logical formula.

1.3.2 Specification logics

While only certain real-time properties pass as a suitable semantics of a reactive system, we allow any real-time property as the semantics of a specification. In particular, we do not require that the analog properties that are defined by specifications are weakly closed under stuttering, digitizable, or given operationally. The only requirements that we put on the interpretation of specification languages are the following:

Transparency Every expression ϕ of the specification language defines an analog property $\Pi_R(\phi)$ — the analog semantics of ϕ . The real-time behaviors that satisfy the specification ϕ are exactly the real-time behaviors in $\Pi_R(\phi)$:

$$[\phi] = Beh(\Pi_R(\phi)).$$

Logicity If ϕ_1 and ϕ_2 are expressions of the specification language, then so are $\phi_1 \wedge \phi_2$, $\phi_1 \vee \phi_2$, and $\neg\phi_1$. Moreover, any boolean combination of expressions defines the corresponding boolean combination of properties:

$$\begin{aligned} \Pi_R(\phi_1 \wedge \phi_2) &= \Pi_R(\phi_1) \cap \Pi_R(\phi_2), \\ \Pi_R(\phi_1 \vee \phi_2) &= \Pi_R(\phi_1) \cup \Pi_R(\phi_2), \\ \Pi_R(\neg\phi) &= TSS_R^* - \Pi_R(\phi). \end{aligned}$$

Clock independence The specification ϕ defines a set of infinite timed state sequences uniformly, independent of the clock model:

$$\Pi_N(\phi) = (\Pi_R(\phi))_N$$

for the digital semantics $\Pi_N(\phi)$ of ϕ .

Note that the *logicality* of the analog semantics together with the condition of *clock independence* ensures that the digital semantics of boolean combinations behaves "logically" as well:

$$\begin{aligned}\Pi_N(\phi_1 \wedge \phi_2) &= \Pi_N(\phi_1) \cap \Pi_N(\phi_2), \\ \Pi_N(\phi_1 \vee \phi_2) &= \Pi_N(\phi_1) \cup \Pi_N(\phi_2), \\ \Pi_N(\neg\phi) &= TSS_N^w - \Pi_N(\phi).\end{aligned}$$

We emphasize that our interpretation of specifications differs from our interpretation of system descriptions: while a precisely timed state sequence that results from observing the behavior of a system stands for the set of real-time behaviors that are consistent with the observation sequence, *a timed state sequence that is admitted by a specification stands only for itself*; while the analog semantics $\Pi \subseteq TSS_R^w$ of a system represents the set $[\Pi]$ of real-time behaviors, the analog semantics Π of a specification represents the set $Beh(\Pi)$ of real-time behaviors. For example, a specification that admits only nondeterministic timed state sequences cannot be satisfied by any system. Our choice to model real-time behavior by timed state sequences forces us to make such a grave distinction in the interpretation of system descriptions, which ought to be refinable, and specifications, which ought to be logical:

1. $Beh(\Pi)$ is an unsuitable denotation of refinable properties, because it is generally different from $Beh(\Gamma(\Pi))$.
2. $[\Pi]$ is an unsuitable denotation of complementable properties, because it is generally not disjoint from $[TSS^w - \Pi]$.

We remark that a restriction to *deterministic* timed state sequences (i.e., state interval sequences) offers a clean solution to this dilemma at the expense of having more complicated basic semantical objects.

1.3.3 Analog versus digital verification

Given a reactive system S and a specification ϕ , we discuss two approaches to the verification problem:

Analog verification The analog approach checks, in the analog-clock model, iff

$$Det(\Pi_R(S)) \subseteq \Pi_R(\phi).$$

Digital verification The digital approach checks, in the digital-clock model, iff

$$Det(\Pi_N(S)) \subseteq \Pi_N(\phi).$$

The analog approach of checking containment of analog properties can be extremely difficult; we will show the analog verification problem to be undecidable for many implementation and specification languages. This is why we have introduced the digital-clock model, which has often a simpler verification problem. Even though recently there have been some surprising successes in analog verification [9], we shall concentrate in this thesis on methods for digital verification.

Note that neither of the two approaches directly solves the original verification problem, which poses a question about real-time behaviors,

$$[S] \stackrel{?}{\subseteq} [\phi],$$

not about analog or digital properties. So what, if anything, is achieved if either the analog or the digital verification problem is solved? For a verification method to be meaningful, obviously its result ought to give at least some insight about the original verification problem. For a particular system S and specification ϕ , we say that an application of analog (digital) verification is *sound* iff

$$Det(\Pi(S)) \subseteq \Pi(\phi) \text{ implies } [S] \subseteq [\phi];$$

it is *complete* iff

$$[S] \subseteq [\phi] \text{ implies } Det(\Pi(S)) \subseteq \Pi(\phi).$$

Soundness ensures that verification does not claim that an incorrect system is correct, although it may discard a correct system as incorrect; soundness and completeness together

guarantee that verification gives the right answer. Since we require the analog semantics of both system descriptions and specifications to be transparent, analog verification is always sound. Digital verification, on the other hand, is sound only for checking certain specifications:

Proposition 1.10 (Soundness of digital verification) *Digital verification is sound for all (systems S and) specifications ϕ with an analog semantics $\Pi_R(\phi)$ that is inversely closed under digitization.*

Proof of Proposition 1.10 Assume that $\Pi_R(\phi)$ is inversely closed under digitization and that $Det(\Pi_N(S)) \subseteq \Pi_N(\phi)$; we show that $Beh(\Pi_R(S)) \subseteq Beh(\Pi_R(\phi))$. Consider an arbitrary real-time behavior $\rho \in \Pi_R(S)$. Then $\rho \in Det(\Pi_R(S))$, which implies that

$$[\rho] \subseteq Det(\Pi_R(S))_N = Det(\Pi_N(S)) \subseteq \Pi_N(\phi) \subseteq \Pi_R(\phi).$$

Since $\Pi_R(\phi)$ is inversely closed under digitization, $\rho \in Beh(\Pi_R(\phi))$. ■

We will show that the criterion for soundness of digital verification is indeed satisfied by our specifications of the most important real-time properties. In addition, it is trivially satisfied by all specifications of time-invariant properties, which are obviously digitizable. We also point out that we may gain some information about a system even from the positive result of an unsound application of digital verification. In that case we have proved something about all digitizations of system behaviors, rather than the system behaviors themselves. For examples, if we are able to show that a system is in state a whenever the digital clock shows 1, then we know that the system is in state a throughout the time interval $(0, 2)$. Thus, to make sure that a system satisfies its specification, we may try to prove a modified assertion by digital verification.

Both analog and digital verification turn out to be complete for the same restricted set of verification problems:

Proposition 1.11 (Completeness of verification) *Analog verification is complete for all (systems S and) specifications ϕ with $\Gamma(Beh(\Pi_R(\phi))) \subseteq \Pi_R(\phi)$. Digital verification is complete under the same condition.*

Proof of Proposition 1.11 Let us assume that both $\Gamma(Beh(\Pi_R(\phi))) \subseteq \Pi_R(\phi)$ and $Beh(\Pi_R(S)) \subseteq Beh(\Pi_R(\phi))$.

(1) To see that analog verification is complete, observe that

$$\text{Det}(\Pi_R(S)) = \Gamma(\text{Beh}(\Pi_R(S))) \subseteq \Gamma(\text{Beh}(\Pi_R(\phi))) \subseteq \Pi_R(\phi).$$

(2) To see that digital verification is complete, use part (1) and the definitions of $\Pi_N(S)$ and $\Pi_N(\phi)$. ■



Chapter 2

Real-time Systems

We identify the possible real-time behaviors of a reactive system in two steps. First, we introduce the *abstract* notion of a timed transition system and define its execution sequences over time. Then, we consider *concrete* real-time systems and show how the concrete constructs can be interpreted within the abstract model. We demonstrate that our framework can model a wide variety of phenomena that routinely occur in conjunction with concurrent real-time processes. Our treatment covers both processes that are executed in parallel on separate processors, and processes that time-share a limited number of processors under a given scheduling policy. Often it is this scheduling policy that determines if a system meets its real-time requirements. Thus we explicitly address such questions as time-outs, interrupts, static and dynamic priorities.

2.1 Abstract Model: Timed Transition Systems

As conceptual model of reactive systems we use discrete transition systems [69, 106], which we generalize by imposing timing constraints on the transitions. Qualitative fairness requirements for transitions are replaced (and superseded) by quantitative lower-bound and upper-bound real-time requirements.

A *transition system* $S = (\Sigma, \Theta, \mathcal{T})$ consists of three components:

1. a (possibly infinite) set Σ of *states*.
2. a subset $\Theta \subseteq \Sigma$ of *initial states*.

3. a finite set \mathcal{T} of *transitions*, including the idle transition τ_I . Every transition $\tau \in \mathcal{T}$ is a binary relation on Σ ; that is, it defines for every state $\sigma \in \Sigma$ a (possibly empty) set of τ -successors $\tau(\sigma) \subseteq \Sigma$. We say that the transition τ is *enabled* on a state σ iff $\tau(\sigma) \neq \emptyset$. In particular, the *idle* (stutter) transition

$$\tau_I = \{(\sigma, \sigma) \mid \sigma \in \Sigma\}$$

is enabled on every state.

An infinite sequence σ of states is a *computation* (execution sequence, run) of the transition system $S = \langle \Sigma, \Theta, \mathcal{T} \rangle$ iff it satisfies the following two requirements:

Initiality $\sigma_0 \in \Theta$.

Consecution For all $i \geq 0$ there is a transition $\tau \in \mathcal{T}$ such that $\sigma_{i+1} \in \tau(\sigma_i)$ (which is also denoted by $\sigma_i \xrightarrow{\tau} \sigma_{i+1}$). We say that τ is *taken* at position i and *completed* at position $i+1$. The case of the idle transition τ_I being taken is called a *stuttering step*.

Let $\Pi(S)$ be the set of all computations of S .

We incorporate time into the transition system model by assuming that all transitions happen “instantaneously,” while real-time constraints restrict the times at which transitions occur. The timing constraints are classified into two categories: *lower-bound* and *upper-bound* requirements. They ensure that transitions occur neither too early nor too late, respectively. All of our time bounds are nonnegative integers. The absence of a lower-bound requirement is modeled by a lower bound of 0; the absence of an upper-bound requirement by an upper bound of ∞ . For notational convenience, we assume that $\infty \geq n$ for all $n \in \mathbb{N}$.

A *timed transition system* $S = \langle \Sigma, \Theta, \mathcal{T}, l, u \rangle$ consists of an underlying transition system $S^- = \langle \Sigma, \Theta, \mathcal{T} \rangle$ as well as

4. a *minimal delay* $l_\tau \in \mathbb{N}$ for every transition $\tau \in \mathcal{T}$. We require that $l_{\tau_I} = 0$.
5. a *maximal delay* $u_\tau \in \mathbb{N} \cup \{\infty\}$ for every transition $\tau \in \mathcal{T}$. We require that $u_\tau \geq l_\tau$ for all $\tau \in \mathcal{T}$, and that $u_\tau = \infty$ if τ is enabled on any initial state in Θ . In particular, $u_{\tau_I} = \infty$.

An infinite timed state sequence $\rho = (\sigma, T)$ is a *computation* of the timed transition system $S = \langle \Sigma, \Theta, \mathcal{T}, l, u \rangle$ iff the state sequence σ is a computation of the underlying transition system S^- , and

Lower bound For every transition $\tau \in \mathcal{T}$ and all positions $i \geq 0$ and $j > i$ such that

$$T_j < T_{i+1} + l_\tau,$$

if τ is taken at position j ,

then τ is enabled on σ_i .

In other words, once enabled, τ is delayed for at least l_τ time units; it can be taken only after being continuously enabled for l_τ time units.

Upper bound For every transition $\tau \in \mathcal{T}$ and position $i > 0$ such that τ is enabled on σ_i , there is some position $j > i$ with $T_j \leq T_{i-1} + u_\tau$ such that

either τ is completed at position j ,

or τ is not enabled on σ_j .

In other words, once enabled, τ is delayed for at most u_τ time units; it cannot be continuously enabled for more than u_τ time units without being completed.

Let $\Pi(S)$ be the set of all computations of S . In the analog-clock model, the system S defines the analog property $\Pi_R(S)$; in the digital-clock model, S defines the digital property $\Pi_N(S) = (\Pi_R(S))_N$; in the untimed model, the untimed property $\Pi_1(S)$. Note that we consider all computations of S to be infinite; finite (terminating as well as deadlocking) computations are represented by infinite extensions that add only stuttering steps. In fact, every computation of S must contain infinitely many stuttering steps.

The timing constraints of a timed transition system S can be viewed as filters that prohibit certain execution sequences of the underlying untimed transition system S^- :

$$\Pi(S)^- \subseteq \Pi(S^-).$$

Special cases of timing constraints are a minimal delay 0 and a maximal delay ∞ for a transition τ . While the former does not rule out any computations of S^- , the latter adds to S^- a *weak-fairness* (justice) assumption [88]: τ cannot be continuously enabled without being taken. The untimed semantics $\Pi_1(S)$ of S results from adding these weak-fairness requirements to the underlying untimed transition system S^- , thus obtaining the *weakly-fair* transition system S_W^- . The analog and digital semantics of S add further constraints.

In the following two subsections we show that the analog semantics $\Pi_R(S)$ of any timed transition system S satisfies our requirements of being maximally closed under stuttering,

digitizable, and, under certain restrictions, given operationally. These attributes assure that timed transition systems can be refined, verified by digital methods, and executed, respectively.

2.1.1 Closure properties

We show that the analog properties that are definable by our abstract machine model of timed transition systems are

1. closed under shifting the origin of time,
2. maximally closed under refinement of time, and
3. both closed and inversely closed under digitization of time.

Linear transformation of time

First let us show that our choice to restrict both maximal and minimal delays of transitions to natural numbers, rather than allowing arbitrary rational numbers, does not limit the real-time phenomena that can be modeled by timed transition systems. The unit of the clock can always be scaled appropriately, because the computations of a timed transition system are, in the following sense, invariant under linear transformations of time. Let $\alpha \neq 0$ and β be arbitrary real numbers. Given a timed transition system $S = \langle \Sigma, \Theta, \mathcal{T}, l, u \rangle$, by $\alpha S = \langle \Sigma, \Theta, \mathcal{T}, \alpha l, \alpha u \rangle$ we denote the timed transition system (if any) that results from S by multiplying all minimal and (finite) maximal delays with α . Similarly, given a timed state sequence $\rho = (\sigma, \mathcal{T})$, we write $\alpha\rho + \beta = (\sigma, \alpha\mathcal{T} + \beta)$ for the timed state sequence (if any)

$$(\sigma_0, \alpha\mathcal{T}_0 + \beta) \longrightarrow (\sigma_1, \alpha\mathcal{T}_1 + \beta) \longrightarrow (\sigma_2, \alpha\mathcal{T}_2 + \beta) \longrightarrow (\sigma_3, \alpha\mathcal{T}_3 + \beta) \longrightarrow \dots$$

Note that neither αS nor $\alpha\rho + \beta$ may be defined.

Proposition 2.1 (Scaling of the clock) *Suppose that both αS and $\alpha\rho + \beta$ are defined for a timed transition system S , an infinite timed state sequence ρ , and real numbers $\alpha \neq 0$ and β . If ρ is a computation of S , then $\alpha\rho + \beta$ is a computation of αS .*

Proof of Proposition 2.1 The proposition follows immediately from the form of the *lower-bound* and *upper-bound* requirements on computations. ■

In particular, the set of computations of a timed transition system is closed under shifting the origin of time (take $\alpha = 1$). In other words, timed transition systems cannot refer to absolute time. Thus we will often assume, without loss of generality, that the time of the first state change of a computation is 0.

Refinement of time

We show that the care we have taken in the definition of *lower-bound* and *upper-bound* requirements for timed transition systems has succeeded to make computations robust under stuttering: given a timed transition system S and a computation ρ of S , the addition of finitely many stuttering steps to ρ yields again a computation of S . In fact, we have the following stronger result.

Proposition 2.2 (Maximal closure under stuttering) *The set of computations of a timed transition system is maximally closed under stuttering.*

Proof of Proposition 2.2 Let S be a timed transition system. First observe that the infinite timed state sequence

$$\dots \longrightarrow (\sigma_i, T_i) \xrightarrow{\tau_i} (\sigma_i, T_{i+1}) \xrightarrow{\tau_{i+1}} (\sigma_i, T_{i+2}) \longrightarrow \dots$$

is a computation of S iff the sequence

$$\dots \longrightarrow (\sigma_i, T_i) \xrightarrow{\tau_i} (\sigma_i, T_{i+2}) \longrightarrow \dots$$

is a computation of S ; and the infinite timed state sequence

$$(\sigma_0, T_0) \xrightarrow{\tau_0} (\sigma_0, T_1) \longrightarrow \dots$$

is a computation of S iff the sequence

$$(\sigma_0, T_1) \longrightarrow \dots$$

is a computation of S , because the maximal delay of every transition that is enabled on σ_0 is ∞ . Then observe that the infinite timed state sequence

$$\dots \longrightarrow (\sigma_i, T_i) \xrightarrow{\tau} (\sigma_{i+1}, T_{i+1}) \longrightarrow \dots$$

is a computation of S iff the two sequences

$$\begin{aligned} \dots &\longrightarrow (\sigma_i, T_i) \xrightarrow{\tau_i} (\sigma_i, T_{i+1}) \xrightarrow{\tau} (\sigma_{i+1}, T_{i+1}) \longrightarrow \dots, \\ \dots &\longrightarrow (\sigma_i, T_i) \xrightarrow{\tau} (\sigma_{i+1}, T_i) \xrightarrow{\tau_i} (\sigma_{i+1}, T_{i+1}) \longrightarrow \dots \end{aligned}$$

are computations of S . It follows that the set of computations of S is maximally closed under stuttering. ■

This proposition confirms that the *lower-bound* and *upper-bound* requirements on computations have the intended meaning under our interpretation of a precisely timed state sequence as a set of real-time behaviors. For the computations of a timed transition system $S = \langle \Sigma, \Theta, \mathcal{T}, l, u \rangle$ can be alternatively characterized as follows: an infinite timed state sequence ρ is a computation of S iff every behavior $\rho' = (\sigma', T')$ that is specified by ρ under stuttering (that is, $\rho' \in [\rho]$) satisfies the *initiality* and *consecution* requirements as well as

Deterministic lower bound For every transition $\tau \in \mathcal{T}$ and all positions $i \geq 0$ and $j \geq i$ such that $T_j < T_i + l_\tau$,

if τ is taken at position j ,
then τ is enabled on σ_i .

Deterministic upper bound For every transition $\tau \in \mathcal{T}$ and position $i \geq 0$ such that τ is enabled on σ_i , there is some position $j \geq i$ with $T_j \leq T_i + u_\tau$ such that

either τ is taken at position j ,
or τ is not enabled on σ_j .

Maximal closure under stuttering allows us to generalize the untimed notion of refinement mappings between transition systems [1]. Suppose that S_1 and S_2 are two timed transition systems with the sets Σ_1 and Σ_2 of states, respectively. Let $f: \Sigma_1 \rightarrow \Sigma_2$ be a mapping between states that is typically many-to-one; that is, f hides some information about the states of S_1 . We say that f is a *refinement mapping* from S_1 to S_2 iff the image of every computation of S_1 is a computation of S_2 :

$$(\sigma, T) \in \Pi(S_1) \text{ implies } (f(\sigma), T) \in \Pi(S_2).$$

The system S_1 is said to *refine* (implement) the system S_2 iff there exists a refinement mapping from S_1 to S_2 . Note that if S_1 refines S_2 , then it may "refine" both state and time information:

State refinement S_1 may contain more state information than S_2 , in which case nondle transitions are mapped to stuttering steps. This is analogous to untimed refinement.

Time refinement S_1 may contain more information about the time of state changes than S_2 , in which case any mapping f between computations is not onto (even if f is onto between the state components of computations). For example, the timed transition system whose computations are the infinite sequences in the stuttering closure of the timed state sequence

$$(a, 0) \longrightarrow (b, 5)$$

is refined by the timed transition system whose computations are contained in the stuttering closure of

$$(a, 2) \longrightarrow (b, 3)$$

(take the identity mapping between states), but not vice versa.

Digitization of time

The following proposition ensures that the analog semantics $\Pi_R(S)$ and the digital semantics $\Pi_N(S)$ of a timed transition system S denote the same set of real-time behaviors.

Proposition 2.3 (Digitizability) *The set of computations of a timed transition system is digitizable.*

Proof of Proposition 2.3 (1) To see that the set of computations of a timed transition system is closed under digitization, observe that

$$T_j \geq T_i + l \text{ implies } [T_j]_\epsilon \geq [T_i]_\epsilon + l$$

and

$$T_j \leq T_i + u \text{ implies } [T_j]_\epsilon \leq [T_i]_\epsilon + u$$

for all $T_i, T_j \in \mathbb{R}$, $l, u \in \mathbb{N} \cup \{\infty\}$, and $0 \leq \epsilon < 1$. These observations guarantee that whenever any digitization $[\rho]_\epsilon$ of a timed state sequence ρ violates a *lower-bound* requirement for the positions $i < j$, then so does ρ ; and whenever ρ satisfies an *upper-bound* requirement for position i at position $j > i$, then so does every digitization $[\rho]_\epsilon$.

(2) To see that the set of computations of a timed transition system is inversely closed under digitization, observe that there is, for all $T_i, T_j \in \mathbb{R}$ and $l, u \in \mathbb{N} \cup \{\infty\}$, some $0 \leq \epsilon < 1$ such that

$$T_j < T_i + l \text{ implies } [T_j]_\epsilon < [T_i]_\epsilon + l$$

and some $0 \leq \epsilon < 1$ such that

$$T_j > T_i + u \text{ implies } [T_j]_\epsilon > [T_i]_\epsilon + u.$$

These observations guarantee that whenever a timed state sequence ρ violates a *lower-bound* requirement for the positions $i < j$, then so does some digitization $[\rho]_\epsilon$ of ρ ; and whenever every digitization $[\rho]_\epsilon$ satisfies an *upper-bound* requirement for position i first at position $j > i$, then so does ρ . ■

There are two immediate ramifications of this result. First, every timed transition system S specifies the same untimed property in analog-clock model and the digital-clock model:

$$\Pi_{\mathbb{R}}(S)^- = \Pi_{\mathbb{N}}(S)^- \subseteq \Pi_1(S)^- \subseteq \Pi(S)^-.$$

Secondly, recall the definition of *lower-bound* and *upper-bound* requirements for timed transition systems. The precise meaning of the timing constraints seems to depend on the time domain. Consider, for example, the *upper-bound* requirement that a continuously enabled transition τ must be completed within its maximal delay of, say, $u_\tau = 5$. While in the analog-clock model this requirement ensures, as intended, that τ is completed within 5 *time units*, in the digital-clock model the same condition appears to guarantee only that τ is not continuously enabled for more than 5 *clock ticks*, allowing the actual difference between τ becoming enabled and τ being completed be as much as, say, 5.9 time units. Closure under digitization implies that this is not the case and that all timing constraints preserve their intended meaning in the digital-clock model.

2.1.2 Operationality

Any particular execution sequence of a timed transition system S is a deterministic computation of S . We say that $Det(\Pi(S))$ is the set of *runs* of S . The run fragments of a timed transition system are obtained by closing the set of runs under suffixes: an infinite timed state sequence ρ' is a *run fragment* of S iff $\rho' = \rho^i$ for some run ρ of S and $i \geq 0$. Note that

run fragments are infinite deterministic timed state sequences; during a run (fragment) time can advance only with the idle transition. In the untimed model, no such distinction between computations and runs is necessary, because all computations are deterministic. Thus the run fragments of an untimed transition system S are all suffixes of the computations of S .

We like our abstract machine model to be operational; that is, the description of a timed transition system should give a prescription of how (all of) its runs can be generated monotonically, by adding a state at a time. Untimed transition systems specify unconditional *safety* properties and are, therefore, trivially executable:

Start with an initial state, and at any point during the stepwise construction of a computation take, nondeterministically, any of the enabled transitions.

At every step during the incremental generation of a timed run, we have to choose either a transition that is taken (without incrementing time) or an amount of time that passes (while the idle transition is taken). A timed transition system S , however, contains *liveness* requirements. Thus it may not be obvious how to choose successive transitions or time increments such that

1. At any point it is possible to extend the generated finite deterministic timed state sequence to a run of S ; that is, all *upper-bound* requirements and the *progress* condition on time can, at some later point, be satisfied.
2. Any run of S can be generated in this fashion.

Fortunately, the safety and liveness components of S can be easily separated: a timed transition system without infinite maximal delays specifies a real-time safety property; infinite maximal delays add real-time liveness requirements. If this decomposition is *congruous* relative to deterministic timed state sequences, it suggests a simple procedure for executing the system S :

Start with an initial state, and at any point during the stepwise construction of a run either take, without advancing time, any of the transitions that have been enabled long enough, or take the idle transition and advance time without delaying any transition for more than its maximal delay. During this procedure,

make sure that all liveness requirements are satisfied "eventually" and time is advanced infinitely often.

We give a sufficient condition on timed transition systems S that can be easily checked and ensures that the decomposition of S into safety and liveness components is real-time congruous.

Safety-liveness decomposition

Let $S = \langle \Sigma, \Theta, \mathcal{T}, l, u \rangle$ be a timed transition system. A finite timed state sequence $\rho = (\sigma, \mathcal{T})$ is said to be a *partial computation* of S iff it satisfies the *initiality*, *consecution*, and *lower-bound* requirements, and

Finite upper bound For every transition $\tau \in \mathcal{T}$ and position $0 < i < |\rho|$ such that τ is enabled on σ_i , either $T_{|\rho|-1} \leq T_{i-1} + u_\tau$ or there is some position $j > i$ with $T_j \leq T_{i-1} + u_\tau$ such that

- either τ is completed at position j ,
- or τ is not enabled on σ_j .

A deterministic partial computation is called a *partial run*. Let $\Pi_s(S)$ be the real-time property that contains an infinite timed state sequence ρ iff all finite prefixes of ρ are partial computations of S . We write $\mathcal{T}_0 \subseteq \mathcal{T}$ and $\mathcal{T}_\infty \subseteq \mathcal{T}$ for the sets of transitions of S with maximal delay 0 and ∞ , respectively. Let $\Pi_l(S)$ be the real-time property that contains an infinite timed state sequence ρ iff

Infinite upper bound For every transition $\tau \in \mathcal{T}_\infty$ there are infinitely many positions $i \geq 0$ such that

- either τ is completed at position i ,
- or τ is not enabled on σ_i .

Clearly,

$$\Pi(S) = \Pi_s(S) \cap \Pi_l(S).$$

It is also not hard to see that $\Pi_s(S)$ is a real-time safety property and $\Pi_l(S)$ is a real-time liveness property.

Thus it is easy to generate all partial computations of S incrementally, by maintaining local consistency: choose, at any point, a state transition τ and time increment δ such that Consecution τ is enabled.

Lower bound τ has been enabled long enough.

Finite upper bound All other enabled transitions can be delayed for another δ time units.

Our hope is that any sequence of such locally consistent choices is globally proper; that is, that any partial computation of S can be extended to a computation of S . In other words, we want to show that the pair $(\Pi_e(S), \Pi_l(S))$ defines $\Pi(S)$ congruously relative to our timing assumption TSS^ω . Unfortunately this is not the case. In fact, there are two impediments:

1. When trying to incrementally generate a *nondeterministic* computation, a locally consistent time increment may be too large. Consider, for example, the situation that a transition τ with minimal delay 5 and maximal delay 5 is disabled on state a and enabled on state b . Then the locally consistent time increment

$$\dots \longrightarrow (a, 5) \longrightarrow (b, 7)$$

may lead to a situation from which time cannot advance without violating either the *lower-bound* or the *upper-bound* requirement for τ . Note that a more restrictive definition of local consistency is not appropriate either, because if τ becomes disabled again, then the time increment given above may actually occur in a computation. Instead, we are content with generating all runs $Det(\Pi(S))$ of a reactive system S , which are deterministic.

2. Even when generating a deterministic computation of a reactive system S , maximal delays 0 may impede the advancement of time. We adopt a simple sufficient criterion for the operability of S that rules out systems with "too many" maximal delays 0: we say that S is an *operational* timed transition system (OTTS) iff there is no sequence of states and transitions

$$\sigma_0 \xrightarrow{\tau_0} \sigma_1 \xrightarrow{\tau_1} \dots \xrightarrow{\tau_{n-1}} \sigma_n$$

such that $n > |\mathcal{T}_0|$ and $\tau_i \in \mathcal{T}_0$ for all $0 \leq i < n$. Intuitively, the maximal delays 0 of an OTTS cannot prevent time from progressing. Formally, the safety-liveness decomposition $(\Pi_e(S), \Pi_l(S))$ of an OTTS S is an operational presentation of $\Pi(S)$:

Proposition 2.4 (Operational timed transition systems) *If S is an operational timed transition system S , then the set $Det(\Pi(S))$ of runs of S is defined congruently by the pair $(\Pi_s(S), \Pi_l(S))$.*

Proof of Proposition 2.4 Clearly,

$$Det(\Pi(S)) = \Pi_s(S) \cap \Pi_l(S) \cap Det(TSS^\omega).$$

It is also not hard to see that $\Pi_s(S)$ is safe relative to $Det(TSS^\omega)$ and $\Pi_l(S)$ is live relative to $Det(TSS^\omega)$.

Thus it remains to be shown that every partial run of an OTTS S can be extended to a run of S . We use the following strategy to extend partial runs. Let \mathcal{T} be the transitions of S and $N \geq (|\mathcal{T}| + 1)^2$ any sufficiently large integer constant. We alternate two phases:

Phase 1 For at least N positions, determine the maximal locally consistent time increment $\delta \in TIME$. If $\delta = 0$, take any transition whose (finite) *upper-bound* requirement prevents the progress of time. If $\delta > 0$, take the idle transition and advance time by δ . If no maximal locally consistent time increment exists (because all enabled transitions can be delayed an arbitrary amount of time), take the idle transition and advance time by any positive (nonzero) amount.

Phase 2 Once every N positions, take every transition with a maximal delay ∞ if it has been enabled long enough. This phase satisfies all *infinite-upper-bound* requirements.

The only way for this strategy not to yield a run of S is that, from some point on, time does not advance. In this case, there has to be a phase-1 sequence ρ of length $N + 1$ of the form

$$(\sigma_i, \tau_i) \xrightarrow{\tau_i} (\sigma_{i+1}, \tau_i) \xrightarrow{\tau_{i+1}} \dots \xrightarrow{\tau_{i+N-1}} (\sigma_{i+N}, \tau_i).$$

The sequence ρ must contain a subsequence ρ' of length $|\mathcal{T}| + 1$ such that every transition in ρ' has been taken at least once before in ρ . It is not hard to see that, contrary to our assumption, the state component of ρ' violates the operability condition on S . ■

It follows that every OTTS can be “executed” in the stepwise fashion that has been outlined above, by incrementally constructing partial runs and satisfying the liveness requirements, which are imposed by maximal delays ∞ and the progress of time, “eventually.”

Explicit-clock transition systems

For any timed transition system $S = \langle \Sigma, \Theta, \mathcal{T}, l, u \rangle$, it will be useful to model the safety part $\Pi_s(S)$ of S by an untimed transition system S^s that is given access to time as part of its state description. The safety machine S^s generates the set of infinite state sequences all of whose finite prefixes correspond to partial runs of S . The correspondence between partial runs of S , which are timed state sequences, and finite prefixes of computations of S^s , which are state sequences, is achieved by augmenting the states of the untimed transition system S^s . Every state of S^s consists of a state of S as well as values for the following new variables:

- A *clock variable* t that ranges over the time domain $TIME$; it records, in every state σ_i of a partial run $\rho = (\sigma, \mathbb{T})$ of S , the corresponding time T_i .
- A *delay counter* d_τ for every transition $\tau \in \mathcal{T}$ that ranges over all time values $\delta \in TIME$ with $0 \leq \delta \leq u_\tau$; it records, in every state of a partial run of S , for how many time units the transition τ has been continuously enabled without being taken. Note that the value of a delay counter is well-defined only for *deterministic* timed state sequences.

The *explicit-clock transition system* $S^s = \langle \Sigma^s, \Theta^s, \mathcal{T}^s \rangle$ associated with the timed transition S is defined to be the following untimed transition system:

1. Every state $\sigma \in \Sigma^s$ of S^s is a tuple that contains a state $\sigma^- \in \Sigma$ of S , a value $\sigma(t) \in TIME$ for the clock variable t , and a value $0 \leq \sigma(d_\tau) \leq u_\tau$ for each delay counter d_τ :

$$\Sigma^s = \Sigma \times TIME \times TIME^{\mathcal{T}}.$$

2. A state of S^s is initial iff it extends an initial state of S :

$$\sigma \in \Theta^s \text{ iff } \sigma^- \in \Theta.$$

3. Every transition of S is extended: \mathcal{T}^s contains, for every $\tau \in \mathcal{T}$, a transition τ^s such that $(\sigma_1^s, \sigma_2^s) \in \tau^s$ iff for all $\tau' \in \mathcal{T}$,

$$\begin{aligned} (\sigma_1, \sigma_2) &\in \tau, \\ \sigma_1^s(d_\tau) &\geq l_\tau, \end{aligned}$$

$$\sigma_2^s(t) = \sigma_1^s(t),$$

$$\sigma_2^s(d_{\tau'}) = \begin{cases} \sigma_1^s(d_{\tau'}) & \text{if } \tau' \neq \tau \text{ and } \tau' \text{ is enabled on } \sigma_2, \\ 0 & \text{otherwise.} \end{cases}$$

The second clause, $\sigma_1^s(d_{\tau}) \geq l_{\tau}$, enforces all *lower-bound* requirements.

In addition, T^s contains the *idle* transition τ_f^s and the *tick* transition τ_T^s that advances time: $(\sigma_1^s, \sigma_2^s) \in \tau_T^s$ iff there is a positive (nonzero) time increment $\delta \in \text{TIME}$ such that for all $\tau' \in T$,

$$\sigma_1 = \sigma_2,$$

$$\sigma_2^s(t) = \sigma_1^s(t) + \delta,$$

$$\sigma_2^s(d_{\tau'}) = \begin{cases} \sigma_1^s(d_{\tau'}) + \delta & \text{if } \tau' \neq \tau \text{ and } \tau' \text{ is enabled on } \sigma_2, \\ 0 & \text{otherwise,} \end{cases}$$

$$\sigma_2^s(d_{\tau'}) \leq u_{\tau'}.$$

The last clause enforces all *finite-upper-bound* requirements. Note that the idle transition τ_f^s would be subsumed by the tick transition τ_T^s if the time increment $\delta = 0$ were admitted for τ_T^s . Our distinction between the idle transition and the tick transition allows us to put different fairness requirements on the two transitions.

We point out that the definition of S^s is independent of the clock model; that is, the set $\Pi_N(S^s)$ of computations of S^s in the digital-clock model contains exactly the computations in the analog semantics $\Pi_R(S^s)$ of S^s that assign only integer values to the clock variable and all delay counters.

It is not hard to see that the timed transition system S and the explicit-clock transition system S^s are related in the following way:

- For every partial run (σ, T) of S , there is a finite state sequence σ^s with $(\sigma^s)^- = \sigma$ and $\sigma^s(t) = T$ such that σ^s is a prefix of a computation of S^s (let all delay counters record the times that the corresponding transitions have been enabled).
- For every finite prefix σ of a computation of S^s , the timed state sequence $(\sigma^-, \sigma(t))$ is a partial run of S .

In other words, S^s generates the state sequences that correspond to safe prefixes of runs of S . In particular:

$$\Pi(S)^- = \text{Det}(\Pi(S))^- \subseteq \Pi(S^s)^- \subseteq \Pi(S^-).$$

Liveness requirements can be easily added to the safety machine S' as follows:

1. A *weak-fairness* assumption stipulates that a transition cannot be continuously enabled without being taken [88]. Let the *weakly-fair extension* S^f of S' be the fair transition system that is obtained from S' by adding a weak-fairness assumption for every transition τ' if τ has a maximal delay ∞ .
2. A *strong-fairness* assumption stipulates that a transition cannot be enabled infinitely often without being taken [88]. Let the *strongly-fair extension* S^f of S' be the fair transition system that is obtained from S^f by adding a strong-fairness assumption for the tick transition τ_T . It is not hard to see that there is a one-to-one correspondence between the computations of S^f and the runs of S :

$$\sigma \in \Pi(S^f) \text{ iff } (\sigma^-, \sigma(t)) \in \text{Det}(\Pi(S)).$$

In particular, $\Pi(S)^- = \Pi(S^f)^-$.

2.2 Concrete Model: Multiprocessing Systems

The concrete real-time systems we consider first consist of a fixed number of sequential real-time programs that are executed in parallel, on separate processors, and communicate through a shared memory. We show how time-outs and real-time response can be programmed in this language. Then we add message passing primitives for process synchronization and communication.

2.2.1 Syntax: Timed transition diagrams

A *shared-variables multiprocessing system* P has the form

$$\{\theta\}[P_1 \parallel \dots \parallel P_m].$$

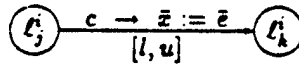
Each *process* P_i , $1 \leq i \leq m$, is a sequential nondeterministic real-time program over the finite set U_i of *private* (local) *data variables* and the finite set U_s of *shared data variables*. The formula θ , called the *data precondition* of P , restricts the initial values of the variables in

$$U = U_s \cup \bigcup_{1 \leq i \leq m} U_i.$$

The real-time programs P_i can be alternatively presented in a textual programming language or as transition diagrams. We shall use the latter, graphical, representation. For this purpose, we extend the untimed transition diagram language by labeling transitions with minimal and maximal time delays. A *timed transition diagram* for the process P_i is a finite directed graph whose vertices $L_i = \{L_0^i, \dots, L_n^i\}$ are called *locations*. The *entry location* — usually L_0^i — is indicated as follows:



The intended meaning of the entry location L_0^i is that the control of the process P_i starts at the location L_0^i . The component processes of a system are not required to start synchronously (i.e., at the same time). Each edge in the graph is labeled by a guarded instruction, a *minimal delay* $l \in \mathbb{N}$ and a *maximal delay* $u \in \mathbb{N} \cup \{\infty\}$ such that $u \geq l$:



where the guard c is a boolean expression, \bar{x} is a vector of variables, and e an equally typed vector of expressions (the guard *true* and the delay interval $[0, \infty]$ are usually suppressed; for the empty vector *nil*, the instruction $c \rightarrow \text{nil} := \text{nil}$ is abbreviated to c ?). We require that every cycle in the graph consists of no fewer than two edges, at least one of which is labeled by a positive (nonzero) maximal delay.

The intended operational meaning of the given edge is as follows. The minimal delay l guarantees that whenever the control of the process P_i has resided at the location L_j^i for at least l time units during which the guard c has been continuously true, then P_i may proceed to the location L_k^i . The maximal delay u ensures that whenever the control of the process P_i has resided at L_j^i for u time units during which the guard c has been continuously true, then P_i must proceed to L_k^i . In doing so, the control of P_i moves to the location L_k^i “instantaneously,” and the current values of e are assigned to the variables \bar{x} . In general, a process may have to proceed via several edges all of whose guards have been continuously true for their corresponding maximal delays. In this case, any such edge is chosen nondeterministically. It follows that the control of a process P_i may remain at a location L_j^i forever only in one of two situations: if L_j^i has no outgoing edges, we say that P_i has *terminated*;

if each of the guards that are associated with the outgoing edges of the location ℓ_j^i is false infinitely often, we say that P_i has *deadlocked*. The second condition is necessary (although not sufficient) for stagnation, because if one guard is true forever, then the corresponding maximal delay $u \leq \infty$ guarantees the progress of P_i .

2.2.2 Semantics: Timed transition systems

The operational view of timed transition diagrams can be captured by a simple translation into the abstract model of timed transition systems. With the given shared-variables multiprocessing system

$$P : \{\theta\}[P_1 \parallel \dots \parallel P_m],$$

we associate the following timed transition system $S_P = (\Sigma, \Theta, \mathcal{T}, l, u)$:

1. Σ contains all interpretations of the finite set

$$V = U \cup \{\pi_1, \dots, \pi_m\}$$

of data and control variables. Each *control variable* for π_i , where $1 \leq i \leq m$, ranges over the set $L_i \cup \{\perp\}$. The value of π_i indicates the location of the control of the process P_i ; it is \perp (undefined) before the process P_i starts.

2. Θ is the set of all states $\sigma \in \Sigma$ such that θ is true in σ and $\sigma(\pi_i) = \perp$ for all $1 \leq i \leq m$.
3. \mathcal{T} contains, in addition to the idle transition τ_I , an *entry transition* τ_0^i for every process P_i , $1 \leq i \leq m$, as well as a transition τ_E for every edge E in the timed transition diagrams for P_1, \dots, P_m . In particular, $\sigma' \in \tau_0^i(\sigma)$ iff

$$\begin{aligned} \sigma(\pi_i) = \perp \text{ and } \sigma'(\pi_i) = \ell_0^i, \\ \sigma'(y) = \sigma(y) \text{ for all } y \in V - \{\pi_i\}. \end{aligned}$$

If E connects the source location ℓ_j^i to the target location ℓ_k^i and is labeled by the instruction $c \rightarrow \bar{x} := \bar{e}$, then $\sigma' \in \tau_E(\sigma)$ iff

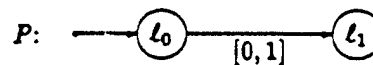
$$\begin{aligned} \sigma(\pi_i) = \ell_j^i \text{ and } \sigma'(\pi_i) = \ell_k^i, \\ c \text{ is true in } \sigma \text{ and } \sigma'(\bar{x}) = \sigma(\bar{e}), \\ \sigma'(y) = \sigma(y) \text{ for all } y \in V - \{\pi_i, \bar{x}\}. \end{aligned}$$

If τ_E is uniquely determined by its source and target locations, we often write $\tau_{j \rightarrow k}^i$.

4. If τ is an entry transition, then $l_\tau = 0$. For every edge E labeled by the minimal delay l , let $l_{\tau_E} = l$.
5. If τ is an entry transition, then $u_\tau = \infty$. For every edge E labeled by the maximal delay u , let $u_{\tau_E} = u$.

This translation defines the sets of possible computations and runs of the concrete real-time system P as the sets $\Pi(S_P)$ and $\text{Det}(\Pi(S_P))$ of (deterministic) timed state sequences. The condition on timed transition diagrams that every cycle contains at least one positive (nonzero) maximal delay ensures that the timed transition system S_P is operational.

For instance, the computations of the trivial system P that consists of a single process with the timed transition diagram



are exactly the infinite timed state sequences in the stuttering closure of

$$(\perp, 0) \longrightarrow (\ell_0, 0) \longrightarrow (\ell_1, 1)$$

(assuming that the time of the initial state change is 0). Note that had we chosen to restrict the semantics of timed transition systems to deterministic timed state sequences, the description of $\Pi(S_P)$ would be substantially more complex — namely, in the analog-clock model, the stuttering closure of an infinite set.

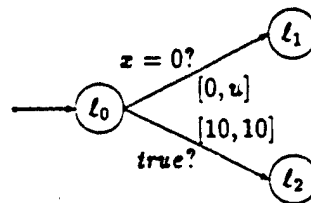
Finally, we remark that our semantics of shared-variables multiprocessing systems is conservative over the untimed case. Suppose that the system P contains no delay labels (recall that, in this case, all minimal delays are 0 and all maximal delays are ∞). Then the state components of the computations of S_P are precisely the legal execution sequences of P , as defined in the interleaving model of concurrency, that are weakly fair with respect to every transition [91]: no process can stop when one of its transitions is continuously enabled. Weak fairness for every individual transition and, consequently, progress for every process is guaranteed by the maximal delays ∞ .

2.2.3 Examples: Time-out and timely response

To demonstrate the scope of the timed transition diagram language, we model two extremely common real-time phenomena as shared-variables multiprocessing systems. In the first example (*time-out*), a process checks if an external event happens within a certain amount of time; in the second example (*traffic light*), a process reacts to an external event and is required to do so within a certain amount of time. A third example combines several processes.

Time-out

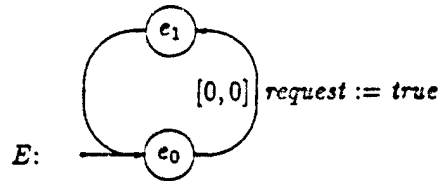
To see how a time-out situation can be programmed, consider the process P with the following timed transition diagram:



When at the location l_0 , the process P attempts, for 10 time units, to proceed to the location l_1 , by checking the value of z . If the value of z is not found to be 0, then P does not succeed and proceeds to the alternative location l_2 after 10 time units. The choice of the maximal delay u determines how often P checks the value of z . For example, if $u \geq 10$, then P may not check the value of z at all before timing out after 10 time units. If $0 < u < 10$, then P has to check the value of z at least once every u time units. Consequently, if the value of z is 0 for more than u time units, it will be detected. On the other hand, the value of z being 0 may go undetected if it fluctuates too frequently, even in the case of $u = 0$.

Traffic light

To give another typical real-time application of embedded systems, let us design a traffic light controller that turns a pedestrian light green within 5 time units after a button is pushed. The environment is given by the following process E . Whenever the request button is pushed, the shared boolean variable *request* is set to *true*:



Recall that the edge labels $true?$ and $[0, \infty]$ are suppressed; thus we have no knowledge about the frequency of requests.

We want to design a traffic light controller Q that controls the status of the traffic light through the variable $light$, whose value is either *green* or *red*. As unit of time we take the amount of time it takes to switch the light; for simplicity, we also assume that, in comparison, the time needed for local operations within Q is negligible. Now let us specify the desired process Q . The controller Q should behave in such a way that the combined system

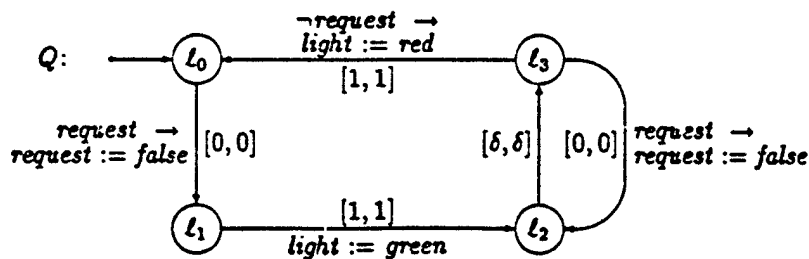
$$P : \{request = false, light = red\} [E||Q]$$

satisfies the following two correctness conditions:

- (A) Whenever $request$ is *true*, then $light$ is *green* within 5 time units for at least 5 time units.
- (B) Whenever $request$ has been *false* for 25 time units, then $light$ is *red*.

The first condition, (A), ensures that no pedestrian has to wait for more than 5 time units to cross the road and is given another 5 time units to do so; the second condition, (B), prevents the light from being always green. Both properties are real-time safety properties.

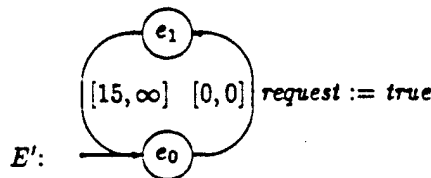
It is not hard to convince ourselves that, once it is started, the following process Q satisfies the specification:



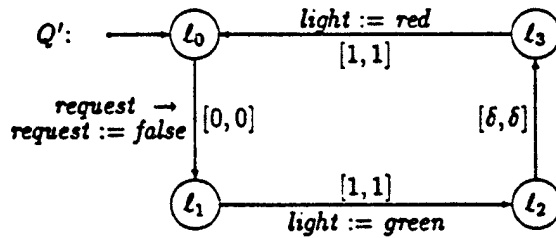
for any delay $4 \leq \delta \leq 23$. This implementation of the traffic light controller turns the light green as soon as possible after a request is received and then waits for δ time units before turning the light red again. Only if the request button has been pushed in the meantime, the light stays green for another δ time units. In the following chapters, we will state both requirements (A) and (B) in a formal language and present verification algorithms that prove the system correct with respect to its specification.

Multiple traffic lights

Let us generalize the *traffic light* example and design a system that reacts to *several* external events. We wish to do so by composing, in parallel, processes that are similar to Q . At this point it is convenient to accept some additional assumptions about the frequencies of the external events. In our example, suppose that the distance between any two requests at least 15 time units; that is,



Then we can simplify the traffic light controller to

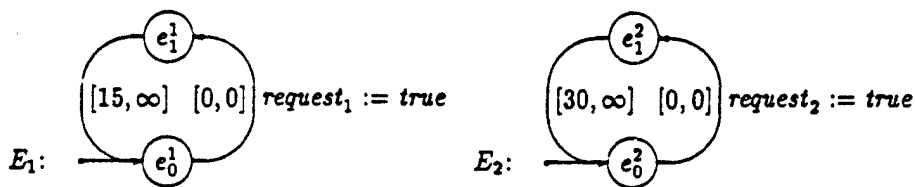


for any delay $4 \leq \delta \leq 17$. The combined system

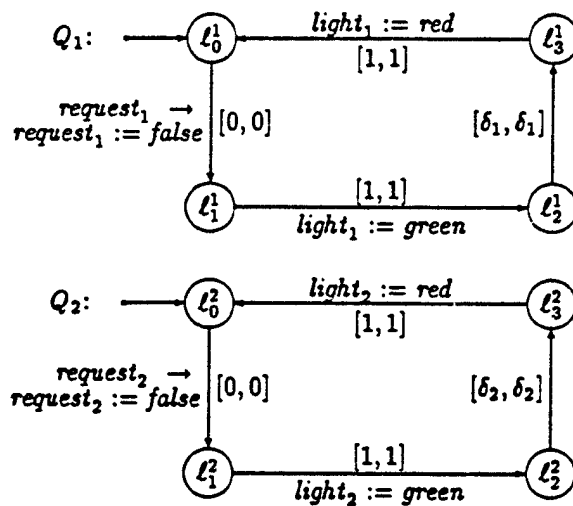
$$P' : \{request = false, light = red\} [E' || Q']$$

still satisfies both correctness requirements (A) and (B).

Now consider a more complex traffic light configuration, with two lights and two request buttons. In particular, we assume that the second light is designed for the special convenience of pedestrians that are in a hurry: it is required to turn green within 3 time units of a request but, on the other hand, has to stay green for only 3 time units. While pedestrians arrive at the first light with a frequency of at most one pedestrian every 15 time units, we assume that the more urgent requests are less frequent — only one every 30 time units:



The controller for both lights executes the following two processes:



If the combined traffic light controller makes use of two processors and the processes Q_1 and Q_2 are executed in a truly concurrent fashion, then the correctness of the entire system

$$P_{||} : \{request_1 = request_2 = false, light_1 = light_2 = red\} [E_1 || E_2 || Q_1 || Q_2]$$

follows from the correctness of its parts. Specifically, if $4 \leq \delta_1 \leq 17$ and $2 \leq \delta_2 \leq 30$, then all runs of $P_{||}$ satisfy the following conditions:

(A₁) Whenever *request*₁ is true, then *light*₁ is green within 5 time units for 5 time units.

(A₂) Whenever *request*₂ is true, then *light*₂ is green within 3 time units for 3 time units.

(B₁) Whenever *request*₁ has been false for 25 time units, then *light*₁ is red.

(B₂) Whenever *request*₂ has been false for 25 time units, then *light*₂ is red.

A more interesting case is obtained if only a single processor is available to control both lights and the two processes Q_1 and Q_2 have to share it. Using the *interleaving* (shuffle) operator of Hoare [59], we denote the resulting system $P_{|||}$ by the expression

$$\{request_1 = request_2 = false, light_1 = light_2 = red\} [E_1 || E_2 || (Q_1 ||| Q_2)].$$

Note that the behavior of the environment $E_1 || E_2$ is still truly concurrent to the behavior of the traffic light controller $Q_1 ||| Q_2$, which executes both processes Q_1 and Q_2 on a single processor in an interleaved fashion.

Let us assume that $\delta_1 = 10$ and $\delta_2 = 2$, in which case $P_{||}$ is correct. However, if we have no knowledge about the strategy by which Q_1 and Q_2 are scheduled on the same processor, other than that it is fair (i.e., the turn of each process will come eventually), then $P_{|||}$ does not satisfy the specification consisting of (A₁), (A₂), (B₁), and (B₂). For suppose that the process Q_1 is always given priority over the process Q_2 , and the traffic light controller receives a request for the second light only one time unit after it has received a request for the first light. Then it will serve the first request by turning *light*₁ green and (busy) waiting for 10 time units, thus violating (A₂).

On the other hand, if the process Q_2 that serves the more urgent yet less frequent requests is always given priority over the process Q_1 , then $P_{|||}$ is correct. This is because of the low frequency of requests for the second light only one such request can interrupt the service of a request for the first light. Clearly, this argument, which depends on a host of possible interleavings of four processes, calls for a formal proof. Even more challenging is the task of deriving sufficient and necessary conditions on the delays δ_1 and δ_2 for the correctness of the system $P_{|||}$.

2.2.4 Message passing

Note that *asynchronous* message passing can be modeled by shared variables that represent message channels. In this subsection, we extend our timed transition diagram language by

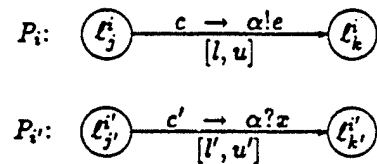
a primitive for *synchronous* (CSP-style) message passing, that can be used for the synchronization and communication of parallel processes.

Syntax

A (*message-passing*) *multiprocessing system* P has the form

$$\{\theta\}[P_1 \parallel \dots \parallel P_m],$$

where θ is a data precondition and each process P_i , for $1 \leq i \leq m$, is a sequential nondeterministic real-time program over the finite set $U_i \cup U_s$ of data variables (for true message-passing systems, we may assume that $U_s = \emptyset$). We use again timed transition diagrams to represent processes, but enrich the repertoire of instructions by guarded *send* and *receive* operations. The send operation $\alpha!e$ outputs the value of the expression e on the channel α ; the receive operation $\alpha?x$ reads an input value from the channel α and assigns it to the variable x . A send instruction and a receive instruction *match* iff they belong to different processes and address the same channel:



For any two matching communication instructions with the delay intervals $[l, u]$ and $[l', u']$, respectively, we require that $\max(l, l') \leq \min(u, u')$.

Since we use the paradigm of synchronous message passing, a send operation can be executed only jointly with a matching receive operation. Thus the intended operational meaning of the given two edges is as follows. Suppose that, for $\max(l, l')$ time units, the control of the process P_i has resided at the location ℓ_j^i and the control of the process $P_{i'}$ has resided at the location $\ell_{j'}^{i'}$, and the guards c and c' have been continuously true; then P_i and $P_{i'}$ may proceed, synchronously, to the locations ℓ_k^i and $\ell_{k'}^{i'}$, respectively. On the other hand, if P_i has resided at ℓ_j^i and $P_{i'}$ has resided at $\ell_{j'}^{i'}$, and the guards c and c' have been continuously true for $\min(u, u')$ time units, then both processes *must* proceed. In doing so, the current value of e is assigned to x .

Semantics

Synchronous message passing can be formally modeled by timed transition systems. We define the timed transition system $S_P = \langle \Sigma, \Theta, \mathcal{T}, l, u \rangle$ that is associated with the given message-passing multiprocessing system P as in the shared-variables case, only that \mathcal{T} contains an additional transition for every matching pair of communication instructions. Suppose that the two edges E (from ℓ_j^i to ℓ_k^i) and E' (from $\ell_{j'}^{i'}$ to $\ell_{k'}^{i'}$) in the timed transition diagrams for P_i and $P_{i'}$ are labeled by the matching instructions $c \rightarrow e!a$ and $c' \rightarrow \alpha?z$, respectively. Then

- \mathcal{T} contains, for the matching edges E and E' , a transition $\tau_{E,E'}$ such that $\sigma' \in \tau_{E,E'}(\sigma)$ iff

$$\begin{aligned} \sigma(\pi_i) &= \ell_j^i \text{ and } \sigma'(\pi_i) = \ell_k^i, \\ \sigma(\pi_{i'}) &= \ell_{j'}^{i'} \text{ and } \sigma'(\pi_{i'}) = \ell_{k'}^{i'}, \\ c \text{ and } c' &\text{ are true in } \sigma \text{ and } \sigma'(x) = \sigma(e), \\ \sigma'(y) &= \sigma(y) \text{ for all } y \in V - \{\pi_i, \pi_{i'}, x\}. \end{aligned}$$

If $\tau_{E,E'}$ is uniquely determined by its source and target locations, we often write $\tau_{j,j' \rightarrow k,k'}^{i,i'}$.

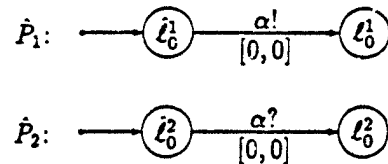
- If the matching edges E and E' are labeled by the minimal delays l and l' , respectively, let $l_{\tau_{E,E'}} = \max(l, l')$.
- If the matching edges E and E' are labeled by the maximal delays u and u' , respectively, let $u_{\tau_{E,E'}} = \min(u, u')$.

This translation defines the set of possible computations of any distributed real-time system P whose processes communicate either through shared variables or by message passing.

Process synchronization

Recall that the component processes of the multiprocessing system $P_1 || P_2$ may start at arbitrary, even vastly different, times. An important application of synchronous message passing is the synchronization of parallel processes. Let P_1 and P_2 be two real-time processes whose timed transition diagrams have the entry locations ℓ_0^1 and ℓ_0^2 , respectively, and let α

be a channel. Now consider the two processes \hat{P}_1 and \hat{P}_2 whose timed transition diagrams are obtained from the transition diagrams for P_1 and P_2 by adding new entry locations:



The added message-passing operations have the effect of synchronizing the start of the two processes P_1 and P_2 (whenever message passing is used for the purpose of process synchronization only, the data that is passed between processes is immaterial and the data components of the instructions are usually suppressed). It follows that the component processes of the multiprocessing system $\hat{P}_1 \parallel \hat{P}_2$ start synchronously, at the exact same (arbitrary) time.

From now on, we shall write $P_1 \parallel P_2$ for the system P whose component processes P_1 and P_2 start synchronously; that is, the notation $P_1 \parallel P_2$ is an abbreviation for the message-passing system $\hat{P}_1 \parallel \hat{P}_2$. Equivalently, we can directly define the formal semantics S_P of the *synchronous* multiprocessing system $P_1 \parallel P_2$ as containing a single entry transition $\tau_0^{1,2}$ for both processes P_1 and P_2 ; namely, $\sigma' \in \tau_0^{1,2}(\sigma)$ iff

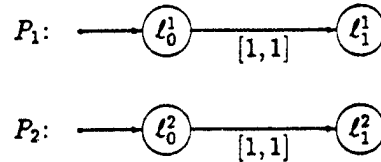
$$\begin{aligned} \sigma(\pi_1) &= \sigma(\pi_2) = \perp, \\ \sigma'(\pi_1) &= \ell_0^1 \text{ and } \sigma'(\pi_2) = \ell_0^2, \\ \sigma'(y) &= \sigma(y) \text{ for all } y \in V - \{\pi_1, \pi_2\}. \end{aligned}$$

It is not hard to generalize our notion of synchronous message passing to synchronous broadcasting, which allows arbitrarily many parallel processes to synchronize simultaneously on joint transitions.

2.3 Concrete Model: Multiprogramming Systems

While the interleaving model for concurrency identifies true parallelism (multiprocessing) with nondeterminism (multiprogramming), the *traffic light* example of Subsection 2.2.3 suggests that the ability of a system to meet its real-time constraints depends crucially on the number of processors that are available and the process allocation algorithm. This is

vividly demonstrated by the following trivial system consisting of the two processes P_1 and P_2 :



If both processes are executed in parallel on two processors, we denote the resulting system by $P_1 \parallel P_2$ (or $P_1 \parallel_s P_2$, if the processes are started at the same time); if they share a single processor and are executed one transition at a time according to some scheduling strategy, the composite system is denoted by $P_1 \parallel\parallel P_2$.

In the untimed case, it is the very essence of the interleaving semantics to identify both systems with the same set of possible (interleaved) execution sequences — the stuttering closure of the two untimed behaviors

$$\begin{array}{c}
 (\ell_0^1, \ell_0^2) \xrightarrow{P_1} (\ell_1^1, \ell_0^2) \xrightarrow{P_2} (\ell_1^1, \ell_1^2), \\
 (\ell_0^1, \ell_0^2) \xrightarrow{P_2} (\ell_0^1, \ell_1^2) \xrightarrow{P_1} (\ell_1^1, \ell_1^2)
 \end{array}$$

(a state is an interpretation of the two control variables π_1 and π_2). Real time, however, can distinguish between true concurrency and (sequential) nondeterminism: if both processes start synchronously, then the parallel execution of P_1 and P_2 terminates within 1 time unit; on the other hand, any interleaved sequential execution of P_1 and P_2 takes 2 time units. This distinction must be captured by our model:

Multiprocessing In the two-processor case $P_1 \parallel_s P_2$, we obtain as computations the stuttering closure of the two real-time behaviors

$$\begin{array}{c}
 (\perp, \perp, 0) \longrightarrow (\ell_0^1, \ell_0^2, 0) \longrightarrow (\ell_0^1, \ell_0^2, 1) \xrightarrow{P_1} (\ell_1^1, \ell_0^2, 1) \xrightarrow{P_2} (\ell_1^1, \ell_1^2, 1), \\
 (\perp, \perp, 0) \longrightarrow (\ell_0^1, \ell_0^2, 0) \longrightarrow (\ell_0^1, \ell_0^2, 1) \xrightarrow{P_2} (\ell_0^1, \ell_1^2, 1) \xrightarrow{P_1} (\ell_1^1, \ell_1^2, 1),
 \end{array}$$

where the third component of every triple denotes the time. Note that the system $P_1 \parallel P_2$ has more computations, because the time difference between the start of P_1 and the start of P_2 can be arbitrarily large.

Multiprogramming In the time-sharing case $P_1 ||| P_2$, the set of computations will be defined to be essentially the stuttering closure of the two real-time behaviors

$$\begin{aligned} (\perp, 0) &\longrightarrow (\ell_0^1, \ell_0^2, 0) \longrightarrow (\ell_0^1, \ell_0^2, 1) \xrightarrow{P_1} (\ell_1^1, \ell_0^2, 1) \longrightarrow (\ell_1^1, \ell_0^2, 2) \xrightarrow{P_2} (\ell_1^1, \ell_1^2, 2), \\ (\perp, 0) &\longrightarrow (\ell_0^1, \ell_0^2, 0) \longrightarrow (\ell_0^1, \ell_0^2, 1) \xrightarrow{P_2} (\ell_0^1, \ell_1^2, 1) \longrightarrow (\ell_0^1, \ell_1^2, 2) \xrightarrow{P_1} (\ell_1^1, \ell_1^2, 2). \end{aligned}$$

We write “essentially,” because we will augment the state by information about the status of the two processes (either active or suspended). Also, observe that we have silently assumed that the swapping of processes is instantaneous and that neither process has priority over the other process. All of these issues will be discussed in detail later.

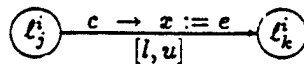
Thus, when time is of the essence, we can no longer ignore the difference between multiprocessing, where each parallel task is executed on a separate machine, and multiprogramming, where several tasks reside on the same machine. In this section, we first show how our model extends to concrete real-time systems that consist of a fixed number of sequential programs that are executed, by time-sharing, on a single processor. Then we use our framework to represent general multiprogramming systems, in which several processes share a pool of processors statically or dynamically.

2.3.1 Syntax and semantics

A *multiprogramming system* P has the form

$$\{\theta\}[P_1 ||| \dots ||| P_m].$$

Each process P_i , for $1 \leq i \leq m$, is again a sequential nondeterministic real-time program over the finite set U of data variables, whose initial values satisfy the data precondition ℓ . We represent the real-time programs P_i by timed transition diagrams as before. Note, however, that in the multiprogramming case the control of the (single) processor resides at one particular location of one particular process. Thus the intended operational meaning of the edge



is as follows. The minimal delay l guarantees that whenever the control (of the single processor) has resided at the location ℓ_j^i for at least l time units and the guard c is true, then the control *may* proceed to the location ℓ_k^i . The maximal delay u ensures that whenever the control has resided at ℓ_j^i for u time units and the guard c is true, then it *must* proceed to ℓ_k^i . This is because no other process can interfere with the active process and change the value of c .

The operational view of the concrete model is again captured formally by a translation into timed transition systems. With the given multiprogramming system P , we associate the following timed transition system $S_P = \langle \Sigma, \Theta, \mathcal{T}, l, u \rangle$:

1. Σ contains all interpretations of the finite set

$$V = U \cup \{\mu, \pi_1, \dots, \pi_m\}$$

of data and control variables. There are two kinds of control variables. the *processor control variable* μ ranges over the set $\{1, \dots, m, \perp\}$; each *process control variable* π_i , for $1 \leq i \leq m$, ranges over the set L_i of locations of the process P_i .

The value of the processor control variable μ is \perp (undefined) before the (single) processor starts executing processes; thereafter the control of the processor resides at the location π_μ of the process P_μ . We say that P_μ is *active*, while all other processes P_i , $i \neq \mu$, are *suspended* (if the value of μ is undefined, then all processes are suspended). The process control variable π_i of a suspended process indicates the location at which the execution of P_i will resume when P_i gains control of the processor.

2. Θ is the set of all states $\sigma \in \Sigma$ such that ϑ is true in σ , and $\sigma(\mu) = \perp$, and $\sigma(\pi_i) = \ell_0^i$ for all $1 \leq i \leq m$.
3. \mathcal{T} contains, in addition to the idle transition τ_I , an *action* transition τ_E for every edge E in the timed transition diagrams for P_1, \dots, P_m . If E connects the source location ℓ_j^i to the target location ℓ_k^i and is labeled by the instruction $c \rightarrow \bar{x} := \bar{z}$, then $\sigma' \in \tau_E(\sigma)$ iff

$$\begin{aligned} \sigma(\mu) &= i, \\ \sigma(\pi_i) &= \ell_j^i \text{ and } \sigma'(\pi_i) = \ell_k^i, \\ c &\text{ is true in } \sigma \text{ and } \sigma'(\bar{x}) = \sigma(\bar{z}), \\ \sigma'(y) &= \sigma(y) \text{ for all } y \in V - \{\pi_i, \bar{x}\}. \end{aligned}$$

Furthermore, there are *scheduling* transitions $\tau \in \mathcal{T}$ that change the status of the processes by resuming a suspended process: $\sigma' \in \tau(\sigma)$ implies that

$$\sigma'(y) = \sigma(y) \text{ for all } y \in U.$$

The scheduling policy determines the set of scheduling transitions.

A scheduling transition τ is called an *entry* transition iff it is enabled on some initial states. We restrict ourselves to scheduling policies with a single entry transition, τ_0 , that is enabled on all initial states. Moreover, we require that $\sigma' \in \tau_0(\sigma)$ implies that

$$\begin{aligned} \sigma(\mu) &= \perp, \\ \sigma'(y) &= \sigma(y) \text{ for all } y \in V - \{\mu\}; \end{aligned}$$

that is, the entry transition τ_0 is enabled precisely on the initial states and activates, perhaps nondeterministically, one of the competing processes.

4. For every edge E labeled by the minimal delay l , let $l_{\tau_E} = l$. Furthermore, $l_{\tau_0} = 0$.
5. For every edge E labeled by the maximal delay u , let $u_{\tau_E} = u$. Furthermore, $u_{\tau_0} = \infty$.

The computations of S_P clearly depend on the scheduling transitions and their delays.

In the untimed case, the scheduling issue can be reduced to fairness assumptions about the scheduling policy: correctness of an untimed multiprogramming system is generally shown for all fair scheduling strategies. It makes, however, little sense to desire that a multiprogramming system satisfies a real-time requirement under all (fair) scheduling strategies, because the scheduling algorithm usually determines if a system meets its timing constraints. In fact, fair scheduling strategies admit *thrashing*: by switching control too often between processes, only scheduling transitions may be performed, because no action transition is enabled long enough so that it has to be taken; thus the system may make no real progress at all and may certainly not meet any real-time deadlines. Consequently, we study the correctness of a real-time multiprogramming systems always with respect to a particular given scheduling policy.

2.3.2 Scheduling strategies

A scheduling strategy may be

Implicit or explicit The scheduling transitions of *implicit* scheduling policies are defined directly and uniformly for an entire class of timed transition systems, independent of the concrete system that is being modeled. A typical example of an implicit strategy is greedy scheduling, which allows the active process to remain active as long as possible. *Explicit* scheduling policies are defined by the concrete system itself. They are specified by scheduling instructions that are part of the individual process descriptions.

Centralized or distributed All scheduling instructions of a *centralized* policy are concentrated in one of the processes — the *scheduler*. In *distributed* policies, any of the processes may make scheduling decisions.

Static or dynamic Unlike *static* policies, *dynamic* scheduling policies may change over time, possibly conditional on the values of data variables.

Neither this attempt at a classification nor the following selection of scheduling strategies is intended to be categorical or comprehensive; we simply try to examine what we think is a representative variety of different scheduling mechanisms and, in the process, hope to convince ourselves of the utility of the timed transition system model. Throughout this subsection we assume a fixed multiprogramming system

$$P : \{\theta\}[P_1 || \dots || P_m]$$

and define the scheduling transitions of the associated timed transition system S_P for various scheduling algorithms.

Greedy scheduling

The simplest reasonable scheduling strategy, as well as our default strategy, is *greedy*. According to this policy, the process that is currently in control of the processor remains active until all its transitions are disabled; at this point an arbitrary other process with an enabled transition takes over. Formally, the set \mathcal{T} of transitions of S_P contains, in addition to the entry transition τ_0 , a single scheduling transition, τ_G , with $\sigma' \in \tau_G(\sigma)$ iff

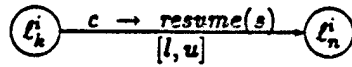
$$\begin{aligned} \sigma(\mu) &\neq \perp, \\ \sigma'(y) &= \sigma(y) \text{ for all } y \in V - \{\mu\}, \end{aligned}$$

$$\begin{aligned} \tau_E(\sigma) &= \emptyset \text{ for all action transitions } \tau_E, \\ \tau_E(\sigma') &\neq \emptyset \text{ for some action transition } \tau_E. \end{aligned}$$

If there is no cost associated with swapping processes, then $l_{\tau_G} = u_{\tau_G} = 0$; if switching processes is not instantaneous, then the minimal and maximal delays of τ_G can be adjusted accordingly.

Scheduling instructions

More flexible scheduling strategies can be implemented with explicit *scheduling* operations. For this purpose, we enrich our programming language by the instruction *resume*(s), where $s \subseteq \{1, \dots, m\}$ determines a subset of processes. The scheduling operation *resume*(s) suspends the currently active process, say, P_i and activates, nondeterministically, one of the processes P_j with $j \in s$:



We write *resume*(j) for *resume*($\{j\}$) and *suspend* for *resume*($\{1 \leq j \leq m \mid j \neq i\}$); that is, the instruction *suspend* delegates the control from the currently active process to any one of the competing processes.

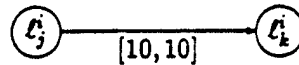
Formally, the set T of transitions of S_P contains, in addition to the entry transition τ_0 , a scheduling transition τ_E for every *resume* edge E in the timed transition diagrams for P_1, \dots, P_m . If E connects the source location ℓ_k^i to the target location ℓ_n^i and is labeled by the instruction $c \rightarrow \text{resume}(s)$, then $\sigma' \in \tau_E(\sigma)$ iff

$$\begin{aligned} \sigma(\mu) &= i \text{ and } \sigma'(\mu) \in s, \\ \sigma(\pi_i) &= \ell_k^i \text{ and } \sigma'(\pi_i) = \ell_n^i, \\ c &\text{ is true in } \sigma, \\ \sigma'(y) &= \sigma(y) \text{ for all } y \in V - \{\mu, \pi_i\}. \end{aligned}$$

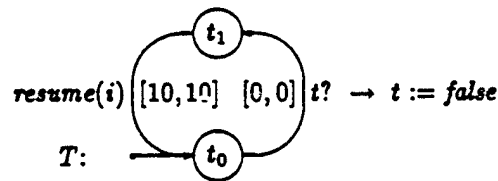
Furthermore, for every scheduling edge E labeled by the minimal delay l and the maximal delay u , let $l_{\tau_E} = l$ and $u_{\tau_E} = u$.

Delays and timers

Note that the instruction



models a busy wait; the process P_i occupies the processor for 10 time units while waiting. To implement a nonbusy wait, in which P_i releases the processor to a competing process for 10 time units before resuming execution, we use a timer T (alarm clock) as a parallel process:



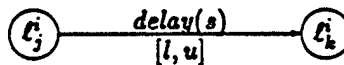
We make sure that the timer T is started (i.e., waiting for activation) when the process P_i becomes active. Then the timer is activated by the sequence



This timer construction

$$\{\emptyset\}[(P_1 || \dots || P_m) ||_s T]$$

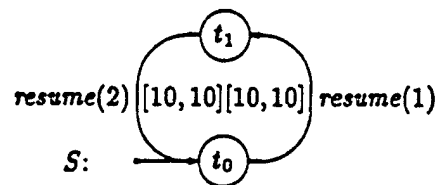
is abbreviated by the *delay* instruction



which allows us to program nonbusy delays without explicit timers. We assume that there exists, implicitly, a unique timer process for every *delay* instruction in a timed transition diagram.

Round-robin scheduling

A construction that is similar to the timer example allows us to implement a *round-robin* scheduling strategy for two processes P_1 and P_2 that share a single processor. In the system $(P_1 ||| P_2) ||_s S$, the scheduler



gives each of the two processes P_1 and P_2 in turn 10 time units of processor time. Needless to say, the explicit scheduling instructions give us the ability to design more sophisticated schedulers as well.

2.3.3 Processor allocation

Both the multiprogramming system with a timer and the multiprogramming system with a central scheduler are, in fact, combinations of multiprocessing and multiprogramming systems in which several tasks compete for some of the processors. In these systems, the question of *scheduling*, which determines the processor time that is granted to individual processes, is preceded by the question of *processor allocation*, which determines the assignment of processes to processors. This assignment can be

Static Every process is assigned to a fixed processor.

Dynamic A set of processes competes for a pool of processors. Over time, processes may reside at different processors.

We only hint how this general notion of real-time system fits into our framework and can be modeled by timed transition systems. A *static* (shared-variables or message-passing) system P with k processors is of the form

$$\{\theta\}[(P_{1,1} ||| \dots ||| P_{1,m_1}) ||| \dots ||| (P_{k,1} ||| \dots ||| P_{k,m_k})];$$

that is, m_i processes compete for the i -th processor. The definition of the associated timed transition system S_P is straightforward: every processor has its own process control variable μ_i , $1 \leq i \leq k$, which ranges over the set of competing processes $\{1, \dots, m_i, \perp\}$ and designates the active process. Furthermore, every processor operates according to a local scheduling policy with a single entry transition τ_0^i , $1 \leq i \leq k$.

To model systems in which a process competes for more than one processor, we simply write

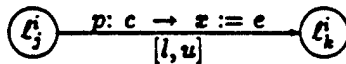
$$\{\theta\}[P_1, \dots, P_m]_k$$

for the *dynamic* system in which m processes compete for k processors according to some global processor allocation and scheduling policy. For these systems, it is useful to have a more general scheduling instruction, *resume*(s, x), which interrupts the process that is currently active on processor x and activates, on processor x , one of the processes from the set s .

2.3.4 Priorities and interrupts

While the explicit scheduling instructions of the previous subsection give us the flexibility to design a scheduler, we often wish to adapt a simple, static scheduling strategy without having to explicitly construct a scheduler. In this subsection, we offer this possibility by generalizing the *greedy* strategy. We assign a priority to every transition, and at any point in a computation, choose only among the transitions with the highest priority. If the transition with the highest priority belongs to a suspended process, then the currently active process is interrupted and the execution of the suspended process is resumed.

A *priority system* P is a (shared-variables or message-passing, static or dynamic) system in which a priority is associated with every instruction; that is, every edge in the timed transition diagrams for P . We use nonnegative integers as priorities (0 being the *highest* priority), and annotate an edge with a priority $p \in \mathbb{N}$ as follows:



We formalize the priority semantics only for simple multiprogramming systems; the generalization to systems with several processors is straightforward. With a given priority

system

$$P: \{\theta\} \{P_1 ||| \dots ||| P_m\},$$

we associate the following timed transition system $S_P = \langle \Sigma, \Theta, \mathcal{T}, l, u \rangle$:

- Σ and Θ are as before.
- \mathcal{T} contains, in addition to τ_I , an action transition τ_E for every assignment edge E in the transition diagrams for P_1, \dots, P_m . If E connects the source location ℓ_j^i to the target location ℓ_k^i and is labeled by the instruction $p: c \rightarrow \bar{x} := \bar{e}$, then $\sigma \rightarrow_E \sigma'$ iff

$$\begin{aligned} \sigma(\pi_i) &= \ell_j^i \text{ and } \sigma'(\pi_i) = \ell_k^i, \\ c &\text{ is true in } \sigma \text{ and } \sigma'(\bar{x}) = \sigma(\bar{e}), \\ \sigma'(y) &= \sigma(y) \text{ for all } y \in V - \{\mu, \pi_i, \bar{x}\}. \end{aligned}$$

Then $\sigma' \in \tau_E(\sigma)$ iff

$$\begin{aligned} \sigma \rightarrow_E \sigma' \text{ and } \sigma(\mu) &= \sigma'(\mu) = i \text{ and} \\ \text{there is no edge } E' &\text{ that is labeled by a higher priority } p' < p \text{ such that} \\ \sigma \rightarrow_{E'} \sigma'' &\text{ for some } \sigma''. \end{aligned}$$

For any matching pair of communication edges E and E' that are labeled by the priorities p and p' , respectively, we take the higher priority $\min(p, p')$ for the combined transition $\tau_{E, E'}$ (although this choice is arbitrary and may be reversed, if the need arises).

Furthermore, there is, in addition to the entry transition τ_0 , a scheduling transition τ_P such that $\sigma' \in \tau_P(\sigma)$ iff

$$\begin{aligned} \sigma(\mu) &\neq \perp, \\ \sigma'(y) &= \sigma(y) \text{ for all } y \in V - \{\mu\}, \\ \tau_E(\sigma) &= \emptyset \text{ for all action transitions } \tau_E, \\ \tau_E(\sigma') &\neq \emptyset \text{ for some action transition } \tau_E. \end{aligned}$$

- Let l_{τ_E} and u_{τ_E} be as before, and choose l_{τ_P} and u_{τ_P} to represent the cost of swapping processes.

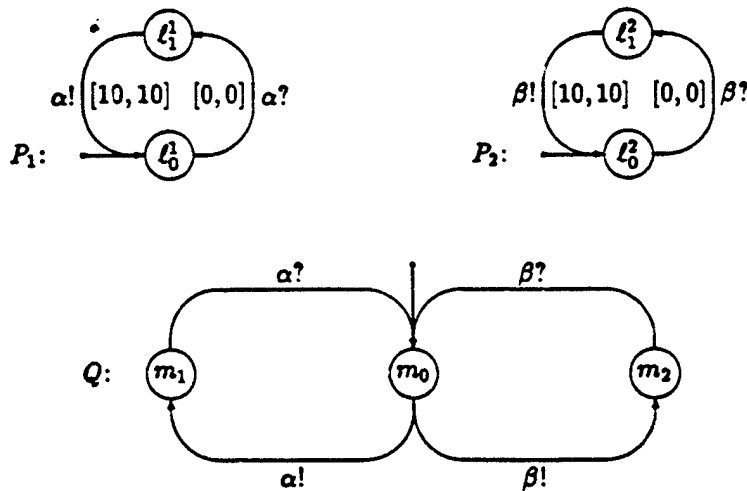
Note that if all transitions have equal priority, then the scheduling strategy is greedy. Thus priorities extend our previous discussion conservatively: all systems can be viewed as priority

systems whose instructions have the same default priority, unless they are annotated with explicit priorities (that is, $\tau_G = \tau_P$).

Dynamic priorities

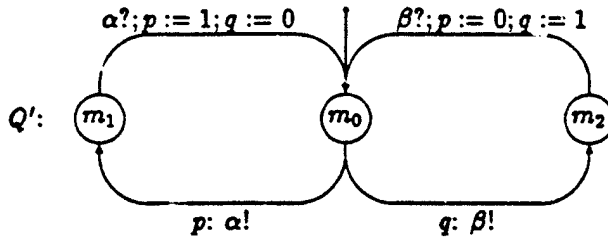
Priorities can be combined with explicit scheduling operations in the obvious way. It is, however, often more convenient to model dynamic scheduling strategies by *dynamic* priorities, which can be changed by any process during execution. Dynamic priorities offer exciting possibilities, such as the ability of a process to increase or decrease its own priority. Moreover, they are easily incorporated into our framework; we simply use data variables that range over the nonnegative integers N as priorities. Instead of giving the formal semantics of dynamic priorities, which is constructed straightforwardly from the semantics of constant (static) priorities, we present an interesting real-time application of dynamic priorities.

We have not yet pointed out that our interpretation of message passing is not entirely conservative over the untimed case: there the set of possible computations usually is restricted by strong-fairness assumptions for communication transitions [91]. This is convenient for the study of time-independent properties of a system, where simple fairness assumptions about “nondeterministic” branching points abstract complex implementation details. Consider, for example, the multiprocessing system $P_1 || P_2 || Q$ that consists of the following three processes P_1 , P_2 , and Q :



(Recall that we may omit the data components of message-passing operations, if they are immaterial.) The arbiter Q mediates between the two processes P_1 and P_2 and uses synchronous communication on the two channels α and β to ensure mutual exclusion: P_1 and P_2 can never be simultaneously in their critical sections ℓ_1^1 and ℓ_1^2 , respectively.

Strong-fairness assumptions on the communication transitions are used to guarantee that, in addition, neither of the two processes P_1 and P_2 is shut out from its critical section forever: the arbiter cannot always prefer one process over the other. Any such infinitary fairness assumption, however, is clearly without bearing on the satisfaction of a real-time requirement such as the demand that a process has to wait at most 10 time units before being able to enter its critical section. As has been the case with scheduling, we have again encountered a situation in which the infinitary notion of "fairness" is adequate for proving untimed properties, yet entirely inadequate for proving real-time constraints. To verify compliance with real-time requirements, we can no longer forgo an explicit description of how the arbiter Q decides between the two processes P_1 and P_2 when both are waiting to enter their critical sections. For instance, the following refinement Q' of Q never makes the same "nondeterministic" choice twice in a row:



(We use semicolons to concatenate instructions; the default value of priorities is assumed to be 0.) The arbiter modifies the priorities p and q of its nondeterministic alternatives to ensure that the system

$$\{p = q = 0\} \{P_1 \parallel P_2 \parallel Q'\}$$

satisfies the requirement that each process has to wait at most 10 time units before being able to enter its critical section. Note that none of the two nondeterministic alternatives is ever disabled, but, at any time, one of them is "preferred."

Finitary branching fairness

Since infinitary fairness assumptions, such as weak fairness for scheduling and strong fairness for synchronization, are insufficient to guarantee the satisfaction of real-time deadlines, one may like to add finitary branching conditions to timed transition systems. Such a finitary notion of fairness would restrict the nondeterminism of a system. We may want to require, for example, that no competitor of a transition τ can be taken more than n times without τ itself being taken (a similar concept has been called *bounded fairness* by Jayasimha [67]). We prefer, both for scheduling and synchronization, an explicit description of the selection process to such implicit assumptions. Since all selection processes that we have found useful can be described within our language, we see no need to introduce additional concepts that would only complicate any verification methodology.

Chapter 3

Real-time Logics

Real-time logics are logics that are interpreted over infinite timed state sequences. Every formula ϕ of a real-time logic defines (specifies) a real-time property $\Pi(\phi)$ — the set of all models of ϕ . We apply two criteria to measure the fitness of a logic as a specification and verification formalism:

Expressiveness *Which real-time properties can be specified?* The expressive power of a logic \mathcal{L} is measured as the set of real-time properties that are definable by the propositional fragment of \mathcal{L} : a real-time property Π can be specified iff there is a propositional formula ϕ of \mathcal{L} such that Π contains exactly the models of ϕ .

Complexity *How difficult is it to verify the expressible properties?* The complexity of a logic \mathcal{L} is measured by the computational complexity of the decision (validity) problem for the propositional fragment of \mathcal{L} : given a propositional formula ϕ of \mathcal{L} , is every timed state sequence a model of ϕ ? The difficulty of this problem is closely related to the hardness of the verification problem for finite-state systems.

The restriction to the *propositional* fragment of a logic

1. gives us information about the intrinsic structural expressiveness and complexity of the logic, independent of the adjunction of first-order data domains, and
2. is necessary to obtain decision procedures. These can then be used for the algorithmic verification of *finite-state* systems, because any finite number of states can be modeled by a finite number of boolean-valued propositions.

We study classical as well as modal real-time logics. First we identify a fundamental boundary between decidable and undecidable classical theories of timed state sequences. Then we introduce two linear-time propositional temporal logics for the specification of real-time properties — *timed* temporal logic TPTL and *metric* temporal logic MTL. Both languages are shown to be as expressive as the maximal decidable classical theory of timed state sequences. In the second part of the thesis, we will identify the complexity of both logics and present algorithms as well as proof systems for the verification of timed transition systems with respect to specifications that are given as formulas of TPTL or MTL.

3.1 From Temporal to Real-time Logics

Since proposed by Pnueli [106], linear temporal logic has firmly established itself as a suitable specification language for many untimed properties of reactive systems (consult, for example, [77, 91, 108]). The tableau-based decision procedure for its propositional version, PTL, provides a proven tool for the algorithmic verification and synthesis of finite-state systems [83, 94]. Finite axiomatizations of PTL [40] have led to relatively complete proof systems for the deductive verification of reactive systems [92].

PTL is interpreted over infinite sequences of states. Its practical appeal stems largely from the strong theoretical connections that PTL enjoys with the classical first-order theory of the natural numbers with linear order and monadic predicates: PTL captures an elementary, yet expressively complete, fragment of this nonelementary theory [40]; that is, any property of state sequences that is definable in the monadic first-order theory of (\mathbb{N}, \leq) can also be specified in PTL, which has a much simpler decision problem.

The formulas of PTL cannot refer explicitly to time. Furthermore, the interpretations of PTL — state sequences — abstract away from the actual times of state changes and retain only the temporal ordering information of states. To admit the specification of timing requirements of reactive systems, we have to

1. extend the *syntax* of PTL by introducing explicit references to time, and
2. extend the *semantics* of PTL by interpreting formulas over timed state sequences, which associate a time with every state.

We wish to carry out both extensions in a way that does not only yield a language that allows the “natural” specification of the real-time properties that encountered in practice, but also preserves the desirable theoretical qualities of PTL:

Expressive completeness We want to be able to characterize the expressive power of a real-time specification language in comparison to a classical theory of timed state sequences. A logic is called *expressively complete* with respect to a classical theory iff the same class of (real-time) properties is definable in both languages.

Elementary decidability We want to be able to generalize the PTL-based tools for the algorithmic verification and synthesis of finite-state systems.

Finite axiomatizability We want to be able to generalize the PTL-based proof techniques for the deductive verification of reactive systems.

To meet these objectives, we will, in particular, have to identify the class of timing constraints that may be added to PTL without sacrificing its (elementary) decidability.

Later in this chapter, we will present *two* extensions of PTL that satisfy our criteria. First, however, let us review both PTL and the “obvious” — and most commonly proposed — way to introduce time explicitly into PTL, and let us illustrate why it falls short of our aspirations. Throughout this chapter and the remaining chapters of this thesis, we shall interpret logics over *infinite* sequences (of states or observations) only; so whenever we refer to a (timed) state sequence, we refer, by default, to a sequence in *TSS*“.

3.1.1 Linear temporal logic

Let $P = \{p, q, \dots\}$ a set of propositions. A state determines the truth value of all propositions. We write $\sigma \models p$, and say that σ is a “ p -state,” iff the proposition p is true in state σ . Thus we may identify states with subsets of propositions (let $p \in \sigma$ iff $\sigma \models p$).

Propositional linear temporal logic (PTL) is a modal logic that is interpreted over infinite sequences of states. The formulas of PTL are built from propositions by boolean connectives and temporal operators. For example, the formula $\Box p$ is true over a state sequence σ iff the proposition p is true in every state of σ . Thus the temporal operator \Box can be intuitively thought of as capturing the temporal notion “always.” The operator \Diamond formalizes the dual

notion “eventually”: the formula $\Diamond p$ is true over a state sequence σ iff the proposition p is true in some state of σ .

Syntax and semantics

More precisely, the logic PTL, as defined by Gabbay, Pnueli, Shelah, and Stavi [40], contains two basic temporal operators — \bigcirc (*next*) and \mathcal{U} (*until*) — from which other operators, including \square (*always*) and \Diamond (*eventually*), can be defined. Thus the formulas ϕ of PTL are inductively defined as follows:

$$\phi := p \mid \text{false} \mid \phi_1 \rightarrow \phi_2 \mid \bigcirc \phi \mid \phi_1 \mathcal{U} \phi_2$$

for $p \in P$. Additional boolean connectives are defined in terms of the two connectives *false* and \rightarrow as usual. Moreover,

Eventually $\Diamond \phi$ stands for $\text{true} \mathcal{U} \phi$.

Always $\square \phi$ stands for $\neg \Diamond \neg \phi$.

Unless $\phi_1 \mathcal{U} \phi_2$ stands for $\phi_1 \mathcal{U} \phi_2 \vee \square \phi_1$. The temporal *unless* operator \mathcal{U} is sometimes called *weak* until, as opposed to the “*strong*” until operator \mathcal{U} .

A PTL-formula ϕ is *satisfiable* (*valid*) iff it is true over some (every) infinite sequence of states. The formula ϕ is true over the infinite state sequence σ iff $\sigma \models \phi$ for the following inductive definition of the satisfaction relation \models :

$$\sigma \models p \text{ iff } p \in \sigma_0.$$

$$\sigma \not\models \text{false}.$$

$$\sigma \models \phi_1 \rightarrow \phi_2 \text{ iff } \sigma \models \phi_1 \text{ implies } \sigma \models \phi_2.$$

$$\sigma \models \bigcirc \phi \text{ iff } \sigma^1 \models \phi.$$

$$\sigma \models \phi_1 \mathcal{U} \phi_2 \text{ iff } \sigma^i \models \phi_2 \text{ for some } i \geq 0, \text{ and } \sigma^j \models \phi_1 \text{ for all } 0 \leq j < i.$$

Consequently, the until formula $p \mathcal{U} q$ requires that p holds until the first subsequent q -state and that there is such a q -state; the unless formula $p \mathcal{U} q$ requires that p holds either until the next q -state or forever, if there no such q -state exists. In the following segment, we give some examples of PTL-formulas that are commonly used for the specification of reactive systems.

Invariance and response properties

Two of the most important classes of qualitative (untimed) properties of reactive systems are invariance and response properties.

- An *invariance* property asserts that, once triggered, a certain condition will hold forever; it is often used to specify that something will never happen. A typical application of invariance is the specification of mutual exclusion: no two processes will ever be simultaneously in their critical sections. Formally, a state sequence σ is contained in the invariance property $\Pi^{\square} \subseteq TSS_1^{\square}$ that specifies that “no p -state is followed by a q -state, ever” iff for all $i \geq 0$,

if $\sigma_i \models p$, then $\sigma_j \not\models q$ for all $j \geq i$.

This property is definable in PTL by the formula

$$\square(p \rightarrow \square\neg q).$$

- A *response* property asserts that, once triggered, a certain condition will become true eventually. A typical application of response is the specification of channel reliability: once a message is sent, it will eventually be received. Formally, a state sequence σ is contained in the response property $\Pi^{\diamond} \subseteq TSS_1^{\diamond}$ that specifies that “every p -state is followed by a q -state, eventually” iff for all $i \geq 0$,

if $\sigma_i \models p$, then $\sigma_j \models q$ for some $j \geq i$.

This property is definable in PTL by the formula

$$\square(p \rightarrow \diamond q).$$

Note that every invariance property is an (untimed) safety property and every response property is a liveness property.

3.1.2 Bounded invariance and bounded response

Not surprisingly, two of the most important classes of quantitative timing requirements of reactive systems are time-bounded versions of invariance and response properties.

- A *bounded-invariance* property asserts that, once triggered, a condition will hold continuously for a certain amount of time; it is often used to specify that something will not happen for a certain amount of time. A typical application of bounded invariance is the specification of best-case performance; that is, the specification of a lower bound l on the termination of a system S : if started at time t , then S will not reach a final state before time $t + l$.

Given two boolean conditions p and q on states and $l \in \mathbb{N}$, we let $\Pi_l^{\square} \subseteq TSS^{\omega}$ stand for the bounded-invariance property that specifies that “no p -state is followed by a q -state within less than l time units” (a boolean condition on states may, for example, be any boolean combination of propositions). Formally, the timed state sequence $\rho = (\sigma, T)$ is contained in the bounded-invariance property Π_l^{\square} iff for all $i \geq 0$,

if $\sigma_i \models p$, then $\sigma_j \not\models q$ for all $j \geq i$ with $T_j < T_i + l$.

- A *bounded-response* property asserts that something will happen within a certain amount of time. A typical application of bounded response is the specification of worst-case performance; that is, the specification of an upper bound u on the termination of a system S : if started at time t , then S is guaranteed to reach a final state no later than at time $t + u$.

Given two boolean conditions p and q on states and $u \in \mathbb{N}$, we let $\Pi_u^{\circ} \subseteq TSS^{\omega}$ stand for the bounded-response property that specifies that “every p -state is followed by a q -state within u time units.” Formally, the timed state sequence $\rho = (\sigma, T)$ is contained in the bounded-response property Π_u° iff for all $i \geq 0$,

if $\sigma_i \models p$, then $\sigma_j \models q$ for some $j \geq i$ with $T_j \leq T_i + u$.

Both bounded invariance and bounded response are (real-time) safety properties. The conjunction of the bounded-invariance property Π_{δ}^{\square} and the bounded-response property Π_{δ}° with the same time bound δ asserts that “the distance between a p -state and the first subsequent q -state is always exactly δ time units.”

Let us add two important remarks about bounded-invariance and bounded-response properties. First, recall that Proposition 1.10 affirms that digital verification methods can be used for establishing analog properties that are inversely closed under digitization. We show that both bounded-invariance and bounded-response properties satisfy this criterion.

This result is crucial to our entire approach, because we will introduce real-time logics for which we give, on one hand, verification techniques in the digital-clock model and, on the other hand, show that verification in the analog-clock model is undecidable. The following proposition justifies the use of the digital-clock model to prove that all real-time behaviors of a reactive system satisfy bounded-invariance and bounded-response properties; it guarantees that our digital methods do establish the desired analog properties.

Proposition 3.1 (Digitizability) *All bounded-invariance and bounded-response properties are digitizable.*

Proof of Proposition 3.1 The argument resembles exactly the proof of Proposition 2.3, which shows that all computations of a timed transition system are digitizable. This is because all finite *lower-bound* requirements of a timed transition system are bounded-invariance properties, and all finite *upper-bound* requirements of a timed transition system are bounded-response properties. ■

Secondly, Proposition 1.11 characterizes the real-time properties for which our approach to verification is complete: it requires that if an analog property Π contains a real-time behavior ρ , then it contains also the stuttering closure of ρ . It is not hard to see that while both bounded-invariance and bounded-response properties generally are not (weakly) closed under stuttering, they do satisfy the criterion

$$\Gamma(\text{Beh}(\Pi_R)) \subseteq \Pi_R$$

of Proposition 1.11. Thus, to check if all real-time behaviors of a reactive system satisfy a bounded-invariance or a bounded-response requirement, it always suffices to check containment of the corresponding analog or digital properties.

3.1.3 Real-time temporal logics

We wish to extend PTL so that bounded-invariance and bounded-response properties can be defined. To begin with, a notational extension of PTL must be able to relate the times of different states. The obvious solution is to employ a first-order temporal logic with a state variable t that represents, in every state, the time of the state (i.e., the "current" time). The bounded-invariance property Π_5^\square can then be written as

$$\forall x. \square((p \wedge t = x) \rightarrow \square(t < x + 5 \rightarrow \neg q)). \quad (\phi_5^\square)$$

The bounded-response property Π_1^\diamond can be defined by the formula

$$\forall x. \Box((p \wedge t = x) \rightarrow \Diamond(q \wedge t \leq x + 1)). \quad (\phi_1^\diamond)$$

Note that in both cases the rigid (global) variable x is used to record the time of any p -state. Let $V = \{x, y, \dots\}$ be a set of rigid variables that range over the time domain *TIME*. By “real-time temporal logic” we refer to the first-order temporal logic over the propositions P , the state variable t , and the rigid variables V that admits an unspecified set of operations on time variables. Our terminology follows Ostroff, who has called this logic RTTL [104] (essentially the same logic has been called GCTL, for “global-clock temporal logic,” by Pnueli and Harel [110]; neither reference is specific about the kind of timing constraints that are permitted).

We will argue that this upgrade from a propositional logic to its full first-order version is unnatural, unnecessary, and prohibitively expensive:

Unnatural We claim that the unconstrained classical quantification of rigid variables allowed in “real-time temporal logic” does not restrict the user to reasonable and readable specifications. We will propose a novel, restricted, form of quantification — we call it *freeze* quantification — by which time variables cannot be bound to arbitrary times, but, in any temporal context, only to the “current” time. This restriction makes the presence of the state variable t superfluous. Freeze quantification identifies, so we will argue, precisely the subclass of “natural,” intended specifications and it leads to a concise and readable notation.

Unnecessary We will show that the restriction of classical quantification to freeze quantification is harmless: it does not limit the class real-time properties that are definable.

Expensive We will show that the restriction of classical quantification to freeze quantification is essential: while the decision problem for “real-time temporal logic” is nonelementary, the decision problem for the corresponding logic with freeze quantification is “only” in EXPSpace.

These results obviously depend on the operations on time that are permitted in “real-time temporal logic.” Thus we shall first identify the maximal theory of time that can be added to PTL without sacrificing its decidability. For this purpose, we will study classical theories of timed state sequences. The most expressive of these theories that is decidable will be

called "the" classical theory of timed state sequences. We will use its expressive power as point of reference for determining which timing constraints can be safely admitted in "real-time temporal logic" and its restriction to freeze quantification, which will be called TPTL.

Unfortunately, the addition of *past* temporal operators, which allow more natural definitions of certain properties, renders TPTL nonelementary. This handicap of TPTL will induce us to introduce a second real-time extension of PTL, called MTL, which contains no time variables at all but, instead, refers to time through time-bounded versions of the temporal operators (including past operators). In a pleasing analogy to the untimed case, we will be able to show that both TPTL and MTL capture expressively complete, elementary, and finitely axiomatizable fragments of the classical theory of timed state sequences, which is nonelementary. Even though both languages are equally expressive, they select *orthogonal* fragments of the classical theory, which makes it easier to specify some real-time properties in TPTL and others are more succinctly defined in MTL. Since both logics inherit the strong theoretical appeal of PTL, the untimed verification techniques can be generalized cleanly to both real-time specification languages.

3.2 The Classical Theory of Timed State Sequences

We introduce "the" classical theory of timed state sequences, show its decidability, and characterize its expressiveness by ω -regular sets. We claim that this theory of timed state sequences is indeed *the* theory for reasoning about finite-state real-time systems. This is because all conceivable extensions and variations, syntactic or semantical, will be shown to be highly undecidable (Π_1^1 -hard) theories.

3.2.1 The classical theory of state sequences

First, we recapitulate briefly why the theory of the natural numbers with linear order and monadic predicates underlies propositional linear-time temporal logics. We also take this opportunity to survey some important results about the complexity and expressive power of PTL.

Syntax and semantics

Let \mathcal{L}^2 be the second-order language with unary predicate symbols and the binary predicate symbol \leq , and let \mathcal{L} be its first-order fragment. We interpret \mathcal{L}^2 over the natural numbers \mathbb{N} , with \leq being interpreted as the usual linear order. Throughout we consider only formulas that contain no free individual variables. Thus, given a formula ϕ of \mathcal{L}^2 with the free predicate symbols p_1, \dots, p_n , an interpretation I for ϕ specifies the sets $p_1^I, \dots, p_n^I \subseteq \mathbb{N}$. Such an interpretation can be viewed as an infinite sequence σ of states $\sigma_i \subseteq \{p_1, \dots, p_n\}$, for $i \geq 0$ (let $p_k \in \sigma_i$ iff $i \in p_k^I$). We denote the set of state sequences that satisfy ϕ by $\Pi_1(\phi)$ or simply $\Pi(\phi)$; hence every \mathcal{L}^2 -formula ϕ defines the untimed property $\Pi(\phi) \subseteq TSS_1^\omega$.

Observe that \mathcal{L}^2 is essentially the language underlying the theory S1S, the second-order theory of the natural numbers with successor and monadic predicates. This is because, in S1S, the order predicate \leq can be defined from the successor function using second-order quantification (and vice versa). Büchi established a close connection of the theory S1S with finite automata over infinite sequences [21] and used this relationship to show that S1S is decidable [22].

Complexity and expressiveness

Formulas of the propositional linear temporal logic PTL can be faithfully translated into \mathcal{L} , by replacing propositions with monadic predicates. For example, the response property Π^\diamond that is expressed in PTL by the formula

$$\Box(p \rightarrow \Diamond q)$$

can be written in \mathcal{L} as

$$\forall i. (p(i) \rightarrow \exists j \geq i. q(j)), \quad (\phi^\diamond)$$

without changing the set of models; that is, $\Pi_1(\phi^\diamond) = \Pi^\diamond$.

Although PTL corresponds to a proper subset of \mathcal{L} , it has the full expressive power of \mathcal{L} [40, 68]; that is, for every \mathcal{L} -formula there is a PTL-formula that defines the same property of state sequences. Furthermore, the decision problem for \mathcal{L} is nonelementary [121], whereas PTL is only PSPACE-complete [119] and has a singly exponential decision procedure [18]. To attain the greater expressive power of \mathcal{L}^2 , PTL may be strengthened by the addition of operators that correspond to right-linear grammars [130]. The resulting logic,

extended temporal logic (ETL), has the expressive power of \mathcal{L}^2 and, like PTL, still a singly exponential decision procedure (these and more results about PTL are surveyed in [35]).

The expressiveness of \mathcal{L}^2 can also be characterized by ω -regular expressions [95]: for any formula ϕ of \mathcal{L}^2 , the set $\Pi_1(\phi)$ can be defined by an ω -regular expression over the alphabet $2^{\{p_1, \dots, p_n\}}$. For example, $\Pi_1(\phi^\circ)$ is described by the expression

$$(\{p, q\} + \{q\} + \{\} + (\{p\}; true^*; (\{p, q\} + \{q\})))^\omega.$$

The restricted expressive power of \mathcal{L} corresponds to the star-free fragment of ω -regular expressions, in which the Kleene star may be applied only to the expression *true* [96, 122] (for an excellent survey of these and related results about regular sets of infinite sequences, see [123]).

3.2.2 Adding time to state sequences

To obtain a theory of *timed* state sequences, we need to identify a suitable time domain *TIME*, with appropriate primitives, and couple the theory of state sequences with this theory of time through a unary ("time") function f , which associates a time with every state. We choose, as the theory of time, the theory of the natural numbers (i.e., $TIME = \mathbb{N}$) with linear-order and congruence primitives. Since time cannot decrease and cannot stagnate, we require that f be monotonic and unbounded. We will have an opportunity to justify these decisions later.

Let \mathcal{L}_T^2 be a second-order language with two sorts, namely a *state* sort and a *time* sort. The vocabulary of \mathcal{L}_T^2 consists of unary predicate symbols and the binary predicate symbol \leq over the state sort, the unary function symbol f from the state sort into the time sort, and the binary predicate symbols $\leq, \equiv_1, \equiv_2, \dots$ over the time sort. By \mathcal{L}_T we denote the first-order fragment of \mathcal{L}_T^2 . We restrict our attention to structures that choose the set of natural numbers \mathbb{N} as domain for both sorts, and interpret the primitives in the intended way (the predicate symbols \equiv_c are interpreted as congruence relations modulo $c \geq 1$). Thus, given a formula ϕ of \mathcal{L}_T^2 with the free predicate symbols p_1, \dots, p_n , an interpretation I for ϕ specifies the sets $p_1^I, \dots, p_n^I \subseteq \mathbb{N}$ and a monotonic and unbounded function $f^I: \mathbb{N} \rightarrow TIME$. The satisfaction relation is defined as usual. Every interpretation I for ϕ can be viewed as an infinite timed state sequence (σ, T) over \mathbb{N} (choose σ as in the untimed case, and let $T_i = f^I(i)$ for all $i \in \mathbb{N}$). We denote the set of timed state sequences over \mathbb{N} that satisfy ϕ

(i.e., the *models* of ϕ) by $\Pi_N(\phi)$ or simply $\Pi(\phi)$. As is standard, we often write $I \models \phi$ for $I \in \Pi(\phi)$.

It follows that \mathcal{L}_T^2 -formulas specify sets of digitally timed state sequences; the formula ϕ defines the digital property $\Pi(\phi) \subseteq TSS_N^\omega$. For example, the bounded-invariance requirement Π_5^\square — that “no p -state is followed by a q -state within less than 5 time units” — can be specified by a formula of \mathcal{L}_T :

$$\forall i. (p(i) \rightarrow \forall j \geq i. (f(j) < f(i) + 5 \rightarrow \neg q(j))) \quad (\phi_5^\square)$$

(note that the successor functions, over either sort, are definable in \mathcal{L}_T); the bounded-response requirement Π_1^\diamond — that “every p -state is followed by a q -state within 1 time unit” — is definable by the \mathcal{L}_T -formula

$$\forall i. (p(i) \rightarrow \exists j \geq i. (q(j) \wedge f(j) \leq f(i) + 1)). \quad (\phi_1^\diamond)$$

An \mathcal{L}_T^2 -formula ϕ is satisfiable (valid) iff it is satisfied by some (every) timed state sequence over N . The (second-order) *theory of timed state sequences* is the set of all valid sentences of \mathcal{L}_T^2 . We prove it to be decidable.

3.2.3 Decidability and expressibility

First we show that, given an interpretation I for an \mathcal{L}_T^2 -formula ϕ , the information in f^I essential for determining the truth of ϕ has finite-state character.

Finite-state character of time

Let us consider the sample formula ϕ_1^\diamond again. A timed state sequence for ϕ_1^\diamond specifies, for every state, the truth values of the predicates p and q , and the value of the time function. Since f is interpreted as a monotonic function, it can be viewed as a state variable dt that records, in every state, the increase in time from the previous state. Although dt ranges over the infinite domain N , observe that if the time increases by more than 1 from a state to its successor, then the actual value of the increase is of no relevance to the truth of the formula ϕ_1^\diamond .

Consequently, to determine the truth of ϕ_1^\diamond , the state variable dt can be modeled using a *finite* number of unary *time-difference* predicates. We employ the three new predicates

$Prev_0$, $Prev_1$, and $Prev_{\geq 2}$ in the following way: $Prev_0$ is true of a state iff the time increase from the previous state is 0, $Prev_1$ is true iff it is 1, and $Prev_{\geq 2}$ is true iff it is greater than 1. Accordingly, we define the notion of an *extended state sequence* for ϕ_1^\diamond , as a state sequence over the propositions $p, q, Prev_0, Prev_1$, and $Prev_{\geq 2}$ such that precisely one of the propositions $Prev_0, Prev_1$, and $Prev_{\geq 2}$ is true in any state.

Given an extended state sequence, we can recover a corresponding *timed state sequence*: the value of the time function in any $Prev_\delta$ -state and $Prev_{\geq \delta}$ -state is obtained by adding δ to its value in the previous state (if $Prev_\delta$ or $Prev_{\geq \delta}$ holds in the first state, let δ be its time). This establishes a many-to-one correspondence between the timed and the extended state sequences for ϕ_1^\diamond ; it induces an equivalence relation on the set of all interpretations for ϕ_1^\diamond such that the truth of ϕ_1^\diamond is invariant within any equivalence class. Every equivalence class is, furthermore, definable by a finite number of predicates.

For formulas with congruence primitives, we need to introduce, apart from time-difference predicates, also unary *time-congruence* predicates, to keep track of the congruence class of the time value of every state. For example, consider the following formula ψ , which states that "p is true in every state with an even time value":

$$\forall i. (f(i) \equiv_2 0 \rightarrow p(i)).$$

Given an interpretation I for ψ , the information in f^I can be captured by the two predicates $Cong_{2,0}$ and $Cong_{2,1}$: the predicate $Cong_{2,0}$ is true for states with even time, and $Cong_{2,1}$ is true for states with odd time.

Now we formalize this idea. Let $c(\phi)$ be the least common multiple of the set

$$\{c \mid \equiv_c \text{ occurs in } \phi\},$$

and $d(\phi)$ the product of $c(\phi)$ and 4^Q , where Q is the number of time quantifiers (i.e., quantifiers over variables of the time sort) occurring in ϕ . Given a formula ϕ of \mathcal{L}_T^2 with the free predicate symbols p_1, \dots, p_n , an *extended state sequence* J for ϕ specifies the sets $p_1^J, \dots, p_n^J \subseteq \mathbb{N}$, a partition of \mathbb{N} into the sets $Prev_0^J, \dots, Prev_{\geq d(\phi)}^J$, and another partition of \mathbb{N} into the sets $Cong_{c(\phi),0}^J, \dots, Cong_{c(\phi),c(\phi)-1}^J$. For any interpretation I for ϕ , the extended state sequence J underlying I is defined as follows:

- J agrees with I on p_1, \dots, p_n .

- For all $i \geq 0$ and $0 \leq \delta < d(\phi)$, $i \in \text{Prev}_\delta^J$ iff $f^I(i) = f^I(i-1) + \delta$.
- For all $i \geq 0$, $i \in \text{Prev}_{\geq d(\phi)}^J$ iff $f^I(i) \geq f^I(i-1) + d(\phi)$.
- For all $i \geq 0$ and $0 \leq \delta < c(\phi)$, $i \in \text{Cong}_{c(\phi), \delta}^J$ iff $f^I(i) \equiv_{c(\phi)} \delta$.

Throughout we use the convention that $f^I(-1) = 0$ for every interpretation I .

Lemma 3.1 (Finite-state character of time) *Given a formula ϕ of \mathcal{L}_T^2 and two interpretations I and J for ϕ with the same underlying extended state sequence, $I \models \phi$ iff $J \models \phi$.*

Proof of Lemma 3.1 Consider two interpretations I and J for the \mathcal{L}_T^2 -formula ϕ that have the same underlying extended state sequence; that is, I and J agree on the free predicate symbols of ϕ , and for each $i \geq 0$, $f^I(i)$ and $f^J(i)$ belong to the same congruence class modulo $c(\phi)$, and either $f^I(i) - f^I(i-1)$ is the same as $f^J(i) - f^J(i-1)$, or both are at least $d(\phi)$.

We use induction on the structure of ϕ to prove our claim. To handle subformulas with free variables properly, we need to strengthen our assumptions about the equivalence of interpretations with respect to a formula. Let ψ be a subformula of ϕ , possibly with free variables. Let $d(\psi)$ be the product of $c(\phi)$ and 4^Q , where Q is the number of time variables bound in ψ . For ease of presentation, we represent the function f by the countable set of variables $\{f_i \mid i \geq 0\}$: for any interpretation I , let $f_i^I = f^I(i)$. By $Tvar(\psi)$ we denote the union of the set of free time variables of ψ with $\{f_i \mid i \geq 0\}$. We say that two interpretations I' and J' for ψ are equivalent with respect to ψ iff they satisfy the following conditions:

- For every predicate symbol q free in ψ , $q^{I'} = q^{J'}$.
- For every state variable i free in ψ , $i^{I'} = i^{J'}$.
- For all $x, y \in Tvar(\psi)$, $x^{I'} \leq y^{I'}$ iff $x^{J'} \leq y^{J'}$.
- For every $x, y \in Tvar(\psi)$, if $0 \leq x^{I'} - y^{J'} < d(\psi)$, then $x^{J'} - y^{J'} = x^{I'} - y^{I'}$, and vice versa.
- For every $x \in Tvar(\psi)$, $x^{I'} \equiv_{c(\phi)} x^{J'}$.

Clearly, the given two interpretations I and J are equivalent with respect to the given formula ϕ . Thus, it suffices to show that, for any subformula ψ of ϕ and equivalent interpretations I' and J' for ψ , $I' \models \psi$ implies $J' \models \psi$. We do so by induction on the structure of ϕ .

The interpretations I' and J' agree on the assignment to predicate symbols and state variables of ψ . They may assign different values to the elements in $Tvar(\psi)$, but they agree on their ordering and modulo- $c(\phi)$ congruence classes. Clearly, if ψ is an atomic formula, then $I' \models \psi$ iff $J' \models \psi$.

The case of boolean connectives is straightforward.

Suppose that ψ is of the form $\exists p. \psi'$, for a predicate symbol p , and that $I' \models \psi$. Let I'' be an extension of I' such that $I'' \models \psi'$. From the inductive hypothesis, the extension of J' that assigns the set $p^{I''}$ to p is a model of ψ' . Hence, $J' \models \psi$. The case that ψ is of the form $\forall p. \psi'$ is similar.

If the outermost operator of ψ is a quantifier for a state variable, then we can proceed as in the previous case.

Now consider the case that ψ is of the form $\exists x. \psi'$, for a time variable x . Suppose that $I' \models \psi$. Let I'' be an extension of I' such that $I'' \models \psi'$. First note that $d(\psi') = c(\phi) \cdot 4^{Q-1}$. We extend J' to an interpretation J'' for ψ' in the following way: if for some $y \in Tvar(\psi)$, $|y^{I''} - x^{I''}| < d(\psi')$, then choose $x^{J''}$ to be $y^{J''} + x^{I''} - y^{I''}$. Otherwise, let $y_1, y_2 \in Tvar(\psi)$ be such that $y_1^{I''} < x^{I''} < y_2^{I''}$. Note that $y_2^{I''} - y_1^{I''}$ is at least $d(\psi)$, and hence, so is $y_2^{J''} - y_1^{J''}$. We choose $x^{J''}$ between $y_1^{J''}$ and $y_2^{J''}$ at a distance at least $d(\psi')$ from either of them. Furthermore, since the difference between $d(\psi)$ and $2d(\psi')$ is at least $c(\phi)$, we can require the modulo- $c(\phi)$ congruence class of $x^{J''}$ to be the same as that of $x^{I''}$. Now I'' and J'' satisfy the requirements listed above. Using the inductive hypothesis, $J'' \models \psi'$, and hence, $J' \models \psi$. The case of universal quantification is similar. ■

It follows that the extended state sequence that underlies a given interpretation for an \mathcal{L}_T^2 -formula ϕ has enough information for deciding the truth of ϕ . Consequently, every formula ϕ can be viewed as characterizing a set $\Pi_1(\phi)$ of extended state sequences, instead of the set $\Pi_N(\phi)$ of timed state sequences. We say that the set

$$\Pi_1(\phi) = \{J \mid J \text{ underlies some } I \in \Pi_N(\phi)\}$$

contains the *untimed models* of ϕ .

Regular nature of the time primitives

Our next task is to show that untimed property $\Pi_1(\phi)$ is ω -regular for every \mathcal{L}_T^2 -formula ϕ . This is achieved by constructing a formula in the language \mathcal{L}^2 that defines the same set of extended state sequences. For instance, the untimed models of the \mathcal{L}_T -formula ϕ_1^\diamond are exactly the models of the \mathcal{L} -formula

$$\forall i. \left(p(i) \rightarrow \exists j \geq i. \left(q(j) \wedge \left(\begin{array}{l} \forall k. (i < k \leq j \rightarrow \text{Prev}_0(k)) \vee \\ \exists k. \left(\begin{array}{l} i < k \leq j \wedge \text{Prev}_1(k) \wedge \\ \forall k' \neq k. \left(\begin{array}{l} i < k' \leq j \rightarrow \\ \text{Prev}_0(k') \end{array} \right) \end{array} \right) \end{array} \right) \right) \right)$$

A generalization of this construction leads to the following theorem.

Theorem 3.1 (Regular nature of the time primitives) *For every formula ϕ of \mathcal{L}_T^2 , there exists a formula ψ of \mathcal{L}^2 that contains the additional time-difference predicates $\text{Prev}_0, \text{Prev}_1, \dots, \text{Prev}_{\geq d(\phi)}$ and the time-congruence predicates $\text{Cong}_{c(\phi),0}, \dots, \text{Cong}_{c(\phi),c(\phi)-1}$, such that $\Pi_1(\phi) = \Pi(\psi)$. Furthermore, if $\phi \in \mathcal{L}_T$ then $\psi \in \mathcal{L}$.*

Proof of Theorem 3.1 Given an \mathcal{L}_T^2 -formula ϕ , we construct an equivalent (with respect to extended state sequences) \mathcal{L}^2 -formula ψ in four steps.

First, we eliminate all time quantifiers. Let I be an interpretation for ϕ , and let $\Delta = d(\phi) + c(\phi)$. We can easily find an interpretation J with the same underlying extended state sequence, such that $f^J(i) \leq f^J(i-1) + \Delta$ for all $i \geq 0$. By Lemma 3.1, we know that $J \models \phi$ iff $I \models \phi$. Based on this observation we perform the following transformation: a subformula $\exists y. \psi(y)$, where y is a time variable, is replaced by the disjunction

$$\bigvee_{\delta=0}^{\Delta} \psi(\delta) \vee \exists i_y. \bigvee_{\delta=0}^{\Delta} \psi(f(i_y) + \delta),$$

for a new state variable i_y . Let ϕ' be the formula obtained from ϕ by applying the above transformation repeatedly until there are no time quantifiers left; clearly $\Pi_1(\phi) = \Pi_1(\phi')$.

The second step, resulting in ϕ'' , models the primitive time arithmetic of comparisons and addition by constants by the time-difference predicates. For instance, consider the subformula $f(i) + 1 \leq f(j)$, for state variables i and j . Intuitively, for $f(i)$ to be less than $f(j)$ in any interpretation, state i has to precede state j , and the time increase from

the previous state has to be positive for some intermediate state. Hence, we replace the subformula by

$$i < j \wedge \exists k. (i < k \leq j \wedge \neg \text{Prev}_0(k)).$$

Similarly, $f(i) \leq f(j)$ and $f(i) \leq f(j) + 1$ can be replaced by the two formulas

$$\forall k. (j < k \leq i \rightarrow \text{Prev}_0(k)), \quad (\phi_0)$$

$$\phi_0 \vee \exists k. (j < k \leq i \wedge \text{Prev}_1(k) \wedge \forall k' \neq k. (j < k' \leq i \rightarrow \text{Prev}_0(k'))),$$

respectively. The generalization of this technique to subformulas of the forms $f(i) + c \leq f(j)$ and $f(i) \leq f(j) + c$, for arbitrary $c > 1$, is straightforward.

In a third step, we model the congruence primitives of ϕ'' with the help of the time-congruence predicates. Consider a subformula of the form $f(i) + c \equiv_d f(j)$. Since there is only a finite number of modulo- $c(\phi)$ congruence classes to which $f(i)$ and $f(j)$ can belong, we can use a case analysis to express this relationship. We replace the subformula by

$$\bigwedge_{k=1}^d \left(\bigvee_{k'=1}^{c(\phi)/d} \text{Cong}_{c(\phi), (k+dk')} \text{ mod } c(\phi)(i) \leftrightarrow \bigvee_{k'=1}^{c(\phi)/d} \text{Cong}_{c(\phi), (k+c+dk')} \text{ mod } c(\phi)(j) \right).$$

Subformulas of the form $f(i) \equiv_d c$ can be handled similarly.

Let ϕ''' be the formula resulting from eliminating all time primitives in the described way. The desired \mathcal{L}^2 -formula ψ is obtained by adding, to ϕ''' , the following conjuncts:

- Exactly one of the time-difference predicates $\text{Prev}_0, \dots, \text{Prev}_{\geq d(\phi)}$ is true for every state $i \geq 0$.
- Exactly one of the time-congruence predicates $\text{Cong}_{c(\phi), 0}, \dots, \text{Cong}_{c(\phi), c(\phi)-1}$ is true for every state $i \geq 0$.
- For all $i \geq 0$, the congruence classes of i and $i + 1$, and the time jump $f(i + 1) - f(i)$ are related in a consistent fashion:

$$\forall i. \bigwedge_{k=c}^{d(\phi)-1} \bigwedge_{k'=0}^{c(\phi)-1} \left(\text{Prev}_k(i+1) \wedge \text{Cong}_{c(\phi), k'}(i) \rightarrow \text{Cong}_{c(\phi), (k'+k)} \text{ mod } c(\phi)(i+1) \right).$$

■

This theorem, combined with the earlier stated facts about \mathcal{L}^2 , gives the following important results regarding the decidability and expressiveness of the theory of timed state sequences.

Corollary 3.1 (Decidability) *The validity problem of the language \mathcal{L}_T^2 is decidable.*

Clearly, the validity problem is nonelementary even for the first-order language \mathcal{L}_T , as \mathcal{L} is a fragment of \mathcal{L}_T (recall that Stockmeyer showed \mathcal{L} to be nonelementary [121]). We point out that the requirement that the time function f be unbounded, which corresponds to the *progress* condition on timed state sequences, has not been used to obtain the decidability result for the language \mathcal{L}_T^2 ; it follows that the validity problem of \mathcal{L}_T^2 is solvable even if \mathcal{L}_T^2 is interpreted over arbitrary monotonic sequences of observations. The *monotonicity* requirement on time, on the other hand, will be shown to be essential for \mathcal{L}_T to be decidable.

Corollary 3.2 (Expressiveness) *Given a formula ϕ of \mathcal{L}_T^2 with the free predicate symbols p_1, \dots, p_n , the set $\Pi_1(\phi)$ can be characterized by an ω -regular expression over the alphabet*

$$2^{\{p_1, \dots, p_n\}} \times \{\text{Prev}_0, \dots, \text{Prev}_{d(\phi)}\} \times \{\text{Cong}_{c(\phi), 0}, \dots, \text{Cong}_{c(\phi), c(\phi)-1}\}.$$

Furthermore, if $\phi \in \mathcal{L}_T$, then $\Pi_1(\phi)$ can be defined by a star-free ω -regular expression.

3.3 Timed Temporal Logic

In this section, we introduce a novel extension of PTL that is interpreted over timed state sequences: *timed propositional temporal logic* (TPTL). We compare the digital properties that are expressible in TPTL with those that are expressible in the classical language \mathcal{L}_T^2 . TPTL is shown to correspond to an expressively complete fragment of \mathcal{L}_T ; that is, the set of models of any \mathcal{L}_T -formula can be defined by a TPTL-formula. Later, in Chapter 4, we will show that TPTL is, in fact, much cheaper to decide (doubly exponential) than the full first-order theory of timed state sequences (nonelementary). Both results are important as they establish TPTL as

1. a sufficiently expressive real-time specification language and
2. a suitable formalism for the algorithmic verification of finite-state real-time systems.

It follows that the gains in complexity in moving from a classical theory to temporal logic are, as in the untimed case, not achieved at the cost of expressive power.

We also try to extend TPTL in many possible ways. Some of these extensions, including "real-time temporal logic," correspond to larger fragments of \mathcal{L}_T and, therefore, are still decidable. However, they turn out to be nonelementary, thus affirming our choice of TPTL as verification formalism. Several more daring generalizations of TPTL will be shown to be even highly undecidable. TPTL can, on the other hand, be extended to attain the full expressiveness of the second-order language \mathcal{L}_T^2 at no cost in complexity.

3.3.1 Syntax and semantics

TPTL is obtained from PTL by adding variables that refer explicitly to time. A time variable x can be bound by a freeze quantifier " $x.$ " that "freezes" x to the "current" time. Let $\phi(x)$ be a formula in which the variable x may occur free. Then $x.\phi(x)$ is satisfied by the timed state sequence $\rho = (\sigma, T)$ iff $\phi(T_0)$ is satisfied by ρ (the formula $\phi(T_0)$ is obtained from $\phi(x)$ by replacing all free occurrences of the variable x by the constant T_0). For example, in the formula $\Diamond x.\phi(x)$, the time reference x is bound to the time of the state at which ϕ is "eventually" true: it specifies that $\phi(T_i)$ is true of *some* suffix ρ^i of a timed state sequence $\rho = (\sigma, T)$. Similarly, the formula $\Box x.\phi(x)$ asserts that $\phi(T_i)$ is true of *every* suffix ρ^i of ρ .

This extension of PTL with explicit references to the times of states admits the expression of timing constraints by atomic formulas that relate the times of different states. The formulas of TPTL are built from propositions and timing constraints by boolean connectives, temporal operators, and freeze quantifiers. For instance, the bounded-invariance requirement Π_5^\square — that "no p -state is followed by a q -state within less than 5 time units" — can be stated in TPTL as

$$\Box x.(p \rightarrow \Box y.(y < x + 5 \rightarrow \neg q)); \quad (\phi_5^\square)$$

the bounded-response requirement Π_1^\diamond — that "every p -state is followed by a q -state within 1 time unit" — is definable by the formula

$$\Box x.(p \rightarrow \Diamond y.(q \wedge y \leq x + 1)). \quad (\phi_1^\diamond)$$

(Read this formula as "Whenever there is a request p , and the variable x is frozen to the current time, the request is followed by a response q , at time y , such that y is at most $x + 5$.")

Syntax of TPTL

Given a set P of proposition symbols and a set V of (time) variables, the terms π and formulas ϕ of TPTL are inductively defined as follows:

$$\pi := x + c \mid c$$

$$\phi := p \mid \pi_1 \leq \pi_2 \mid \pi_1 \equiv_d \pi_2 \mid \text{false} \mid \phi_1 \rightarrow \phi_2 \mid \bigcirc \phi \mid \phi_1 \mathcal{U} \phi_2 \mid x. \phi$$

for $x \in V$, $p \in P$, $c, d \in \mathbb{N}$, and $d \neq 0$. If all terms in a formula ϕ of TPTL contain variables, we say that ϕ contains no *absolute time references*. The abbreviations x (for $x + 0$), $=$, $<$, $>$, \geq , *true*, \neg , \wedge , \vee , and \leftrightarrow are defined as usual. Additional temporal operators are defined in terms of the given *next* and *until* operators as in PTL:

Eventually $\diamond \phi$ stands for *true* $\mathcal{U} \phi$.

Always $\square \phi$ stands for $\phi \mathcal{U} \text{false}$ or, equivalently, $\neg \diamond \neg \phi$.

Unless $\phi_1 \cup \phi_2$ stands for $\phi_1 \mathcal{U} \phi_2 \vee \square \phi_1$.

Note that the timing constraints of TPTL allow only the addition of integer *constants* to time variables, not the addition (or any other binary operation) of variables. From a logical point of view, this restriction limits us to the *successor* operation on time; we will have a chance to justify it later. We have, on the other hand, not defined TPTL-terms by a unary successor operator, because for determining the length of a TPTL-formula, we assume that all constants are given in a reasonably succinct (e.g., binary) encoding. The size of a formula will be important for locating the computational complexity of problems whose input includes formulas of TPTL.

Semantics of TPTL

The formulas of TPTL are interpreted over timed state sequences. Let $\rho = (\sigma, \tau)$ be a timed state sequence and $\mathcal{E} : V \rightarrow \text{TIME}$ be an interpretation (environment) for the variables. The pair (ρ, \mathcal{E}) satisfies the TPTL-formula ϕ iff $\rho \models_{\mathcal{E}} \phi$, where the satisfaction relation \models is inductively defined as follows for all $i \geq 0$:

$$\rho^i \models_{\mathcal{E}} p \text{ iff } p \in \sigma_i.$$

$$\rho^i \models_{\mathcal{E}} \pi_1 \leq \pi_2 \text{ iff } \mathcal{E}(\pi_1) \leq \mathcal{E}(\pi_2).$$

$$\rho^i \models_{\mathcal{E}} \pi_1 \equiv_d \pi_2 \text{ iff } \mathcal{E}(\pi_1) \equiv_d \mathcal{E}(\pi_2).$$

$$\rho^i \not\models_{\mathcal{E}} \text{false}.$$

$$\rho^i \models_{\mathcal{E}} \phi_1 \rightarrow \phi_2 \text{ iff } \rho^i \models_{\mathcal{E}} \phi_1 \text{ implies } \rho^i \models_{\mathcal{E}} \phi_2.$$

$$\rho^i \models_{\mathcal{E}} \bigcirc \phi \text{ iff } \rho^{i+1} \models_{\mathcal{E}} \phi.$$

$$\rho^i \models_{\mathcal{E}} \phi_1 \mathcal{U} \phi_2 \text{ iff } \rho^j \models_{\mathcal{E}} \phi_2 \text{ for some } j \geq i, \text{ and } \rho^k \models_{\mathcal{E}} \phi_1 \text{ for all } i \leq k < j.$$

$$\rho^i \models_{\mathcal{E}} x. \phi \text{ iff } \rho^i \models_{\mathcal{E}[x:=T_i]} \phi.$$

Here $\mathcal{E}(x + c) = \mathcal{E}(x) + c$ and $\mathcal{E}(c) = c$. Moreover, $\mathcal{E}[x := t]$ denotes the environment that agrees with $\mathcal{E} : V \rightarrow \text{TIME}$ on all variables except x , which is mapped to $t \in \text{TIME}$. Note that although our definition of syntax and semantics of TPTL is independent of the clock model, in the analog-clock model congruence relations are not meaningful, and therefore not permitted.

A TPTL-formula ϕ is *satisfiable (valid)* iff $\rho \models_{\mathcal{E}} \phi$ for some (every) timed state sequence ρ and some (every) environment \mathcal{E} . The truth value of a *closed* formula, which contains no free variables, is completely determined by a timed state sequence alone. We say that the timed state sequence ρ is a *model* of the closed formula ϕ , and write $\rho \models \phi$, iff the pair (ρ, \mathcal{E}) satisfies ϕ for any environment \mathcal{E} . As usual, two closed formulas are called *equivalent* iff they have the same models.

Henceforth, we shall consider only closed formulas of TPTL: whenever we refer to a TPTL-formula ϕ , we assume that ϕ contains no free variables. Thus every TPTL-formula ϕ defines, in the analog-clock model, an analog property $\Pi_R(\phi) \subseteq \text{TSS}_R^{\omega}$; in the digital-clock model, a digital property $\Pi_N(\phi) \subseteq \text{TSS}_N^{\omega}$; in the untimed model, an untimed property $\Pi_1(\phi) \subseteq \text{TSS}_1^{\omega}$:

$$\rho \in \Pi(\phi) \text{ iff } \rho \models \phi$$

for all $\rho \in \text{TSS}^{\omega}$. If we omit the parameter *TIME*, we refer, by default, to the *digital-clock* model: unless stated otherwise, the models of a TPTL-formula are taken to be timed state sequences over \mathbb{N} . This decision will also be justified later.

Observe that every TPTL-formula ϕ without freeze quantifiers specifies a time-invariant property and can be read as a formula ψ of PTL. Indeed, TPTL builds conservatively on

the semantics of PTL:

$$\Pi(\phi)^- = \Pi_1(\phi) = \Pi(\psi).$$

We also remark that TPTL as we originally defined it [10] differs syntactically in that the freeze quantifiers were coupled with the temporal operators. The coupling does not restrict us in any essential way: by separating the quantifier "x." from the temporal operators, we admit more formulas (such as $\Box(x.\phi \rightarrow x.\psi)$), for each of which there is an equivalent formula in which every quantifier follows a temporal operator ($\Box x.(\phi \rightarrow \psi)$). However, when we study, in Chapter 5, TPTL as a modal logic, the separation of quantifiers from the modal operators will turn out to be useful.

TPTL as a specification language

Let us demonstrate how TPTL improves the readability of real-time specifications by replacing classical quantification with freeze quantification. A typical real-time requirement for a reactive system is that a switch p has to be turned off (represented by the proposition q) within, say, 10 time units of its activation. In TPTL this condition can be expressed by the formula

$$\Box x.(p \rightarrow p\mathcal{U}y.(q \wedge y \leq x + 10)). \quad (1)$$

Using "real-time temporal logic" and its state variable t that assumes the value of the current time in every state, this specification usually is written as follows [104]:

$$\Box((p \wedge z = t) \rightarrow p\mathcal{U}(q \wedge y = t \wedge y \leq z + 10)). \quad (2)$$

The meaning of this formula depends, not surprisingly, on the quantification of the rigid time variables x and y , which is left implicit. The very fact that the quantification is often omitted [104, 110] suggests that the authors have some particular quantifiers for rigid variables in mind, whose force, location, and order are considered to be so obvious that they are not worth mentioning. If any quantifiers are given, they form a prefix to the entire formula [53].

We claim that the following quantification is the (only) "intended" one:

$$\Box \forall x.((p \wedge z = t) \rightarrow p\mathcal{U} \exists y.(q \wedge y = t \wedge y \leq z + 10)). \quad (3)$$

Note that (3) is equivalent to (1), but not to (2) with any quantifier prefix. In particular, (3) does not imply the stronger condition

$$\forall x. \exists y. \Box((p \wedge x = t) \rightarrow pU(q \wedge y = t \wedge y \leq x + 10)). \quad (4)$$

The difference is subtle: while the formula (3) asserts that every p -state of time x is followed by p -states and, eventually, a q -state of time $y \leq x + 10$, the formula (4) demands more; that if there is a p -state of time x , then there is a time $y \leq x + 10$ such that every p -state of time x is followed by p -states and, eventually, a q -state of time y . For instance, the timed state sequence

$$(\{p\}, 0) \longrightarrow (\{q\}, 0) \longrightarrow (\{p\}, 0) \longrightarrow (\{q\}, 1) \longrightarrow (\{\}, 2) \longrightarrow (\{\}, 3) \longrightarrow \dots$$

satisfies (3) but not (4).

Thus, TPTL selects the fragment of "real-time temporal logic" in which all variables are bound to the times of states: the rigid variables are, immediately upon introduction, frozen to the time (i.e., the value of the state variable t) in the local temporal context. In particular, the TPTL-formula $x. \phi$ defines the same property as the formula

$$\forall x. (x = t \rightarrow \phi)$$

or, equivalently,

$$\exists x. (x = t \wedge \phi)$$

of "real-time temporal logic." It is precisely this restriction of our ability to arbitrarily quantify over time variables that permits (and limits) us to express timing constraints between states by concise and readable specifications (compare (1) with (3)). Later we will see that it is also precisely this restriction of time variables to refer to "temporal" sets of states that allows the development of verification algorithms as well as complete proof systems.

Decidability of TPTL

In the digital-clock model, every TPTL-formula ϕ can be translated into the classical language \mathcal{L}_T , while preserving the set of models $\Pi_N(\phi)$. For every proposition p of TPTL, we have a corresponding unary state predicate $p(i)$ of \mathcal{L}_T . We translate a TPTL-formula ϕ to the \mathcal{L}_T -formula $Classic_0(\phi)$, where the mappings $Classic_i$, for $i \geq 0$, are inductively defined as follows:

$$\begin{aligned}
\text{Classic}_i(p) &= p(i), \\
\text{Classic}_i(\pi_1 \leq \pi_2) &= \pi_1 \leq \pi_2, \\
\text{Classic}_i(\pi_1 \equiv_d \pi_2) &= \pi_1 \equiv_d \pi_2, \\
\text{Classic}_i(\text{false}) &= \text{false}, \\
\text{Classic}_i(\phi_1 \rightarrow \phi_2) &= \text{Classic}_i(\phi_1) \rightarrow \text{Classic}_i(\phi_2), \\
\text{Classic}_i(\bigcirc \phi) &= \text{Classic}_{i+1}(\phi), \\
\text{Classic}_i(\phi_1 \mathcal{U} \phi_2) &= \exists j \geq i. (\text{Classic}_j(\phi_2) \wedge \forall i \leq k < j. \text{Classic}_k(\phi_1)), \\
\text{Classic}_i(x.\phi) &= \text{Classic}_i(\phi)[x := f(i)].
\end{aligned}$$

We write $\phi[x := f(i)]$ for the formula that is obtained from ϕ by replacing all free occurrences of x by $f(i)$.

It is not hard to see that a TPTL-formula ϕ is true over a timed state sequence $\rho \in TSS_{\mathbb{N}}^w$ iff ρ satisfies the \mathcal{L}_T -formula $\text{Classic}_0(\phi)$:

$$\Pi(\phi) = \Pi(\text{Classic}_0(\phi))$$

for every TPTL-formula ϕ . For example, the bounded-response formula Π_1^\diamond is equivalent to its translation $\text{Classic}_0(\Pi_1^\diamond)$:

$$\forall i \geq 0. (p(i) \rightarrow \exists j \geq i. (q(j) \wedge f(j) \leq f(i) + 1)).$$

From Corollary 3.1, it follows that TPTL is decidable. Note that the mapping Classic_0 embeds TPTL into \mathcal{L}_T ; its range constitutes a proper subset of all well-formed \mathcal{L}_T -formulas. Thus, just as PTL corresponds to a subset of \mathcal{L} , we may view TPTL as a fragment of \mathcal{L}_T : quantification over the state sort is restricted to the "temporal" way of PTL, while quantification over the time sort is prohibited entirely.

We remark that the mapping Classic_0 translates a TPTL-formula ϕ with free variables into a \mathcal{L}_T -formula with free variables of the time sort. It follows that ϕ is valid iff the universal closure of $\text{Classic}_0(\phi)$ is valid, which shows that TPTL with free variables is decidable as well.

3.3.2 Expressive completeness

We show that the restrictions imposed by TPTL on the quantification in \mathcal{L}_T -formulas do not diminish its expressive power. In other words, any digital property that can be defined

in \mathcal{L}_T can already be defined in TPTL. The natural embedding $Classic_0$ gives, for any TPTL-formula ϕ , an equivalent \mathcal{L}_T -formula $Classic_0(\phi)$, thus demonstrating that \mathcal{L}_T is as expressive as TPTL. By the following theorem, the converse is also true.

Theorem 3.2 (Expressive completeness of TPTL) *For every formula ϕ of \mathcal{L}_T , there exists a formula ψ of TPTL such that $\Pi(\phi) = \Pi(\psi)$.*

Proof of Theorem 3.2 Given an \mathcal{L}_T -formula ϕ , we construct an equivalent TPTL-formula ψ in four steps. By Theorem 3.1, we obtain an \mathcal{L} -formula ϕ' , with additional time-difference predicates $Prev_\delta$ and $Prev_{\geq\delta}$ and time-congruence predicates $Cong_{c,\delta}$, such that $\Pi_1(\phi) = \Pi(\phi')$. By the expressive completeness of PTL, there is a PTL-formula ϕ'' such that $\Pi(\phi') = \Pi(\phi'')$ [40].

We transform ϕ'' into an equivalent PTL-formula ϕ''' such that every time-difference proposition $Prev_\delta$ and $Prev_{\geq\delta}$ is either not within the scope of any temporal operator, or immediately preceded by a *next* operator. This can be done by repeatedly rewriting subformulas of the form $\bigcirc(\phi_1 \rightarrow \phi_2)$ and $\phi_1 \mathcal{U} \phi_2$, to $\bigcirc\phi_1 \rightarrow \bigcirc\phi_2$ and $\phi_2 \vee (\phi_1 \wedge (\bigcirc\phi_1) \mathcal{U} (\bigcirc\phi_2))$, respectively. From ϕ''' we arrive at ψ by replacing every time-difference proposition $Prev_\delta$ and $Prev_{\geq\delta}$ that is not within the scope of a temporal operator with $x.x = \delta$ and $x.x \geq \delta$, respectively; by replacing every subformula $\bigcirc Prev_\delta$ and $\bigcirc Prev_{\geq\delta}$ with $x.\bigcirc y.y = x + \delta$ and $x.\bigcirc y.y \geq x + \delta$; and by replacing every time-congruence proposition $Cong_{c,\delta}$ with $x.x \equiv_c \delta$. ■

We conclude the discussion of properties that are expressible in TPTL by interpreting the logic over untimed state sequences, and investigating the expressive power of the congruence relations.

Timeless expressiveness

Note that in the untimed model every atomic timing constraint of TPTL is trivially true. Thus TPTL can express, in the untimed model, exactly the properties that are definable in PTL. It follows that the *untimed* expressive power of TPTL is that of the first-order language \mathcal{L} or, equivalently, star-free ω -regular expressions.

There is, however, another way to interpret TPTL-formulas over infinite state sequences. With every TPTL-formula ϕ we can associate the untimed property $\Pi_N(\phi)^- \subseteq TSS_1^*$ that

results from collecting the state components of all models of ϕ . Interpreted in this fashion, TPTL can define strictly more properties of state sequences than PTL. For example, the untimed property *even*(p), that “ p holds in every even state,” is not expressible in PTL [130]. In TPTL, we may (ab)use time to identify the even states as exactly those in which the time does not increase:

$$\bigcirc y. x = y \wedge \square z. \bigcirc y. (x = y \rightarrow p \wedge \bigcirc z. (z > y \wedge \bigcirc u. u = z)).$$

We refer to the set of projections of digital properties that are definable in TPTL as the *timeless* (as opposed to *untimed*) expressiveness of TPTL. The following proposition shows that the timeless expressive power of TPTL is that of the second-order language \mathcal{L}^2 or, equivalently, ω -regular expressions.

Proposition 3.2 (Timeless expressiveness of TPTL) *For every formula ϕ of TPTL, there is a formula ψ of \mathcal{L}^2 such that $\Pi_N(\phi)^- = \Pi(\psi)$, and vice versa.*

Proof of Proposition 3.2 (1) Given a TPTL-formula ϕ , we know how to construct an equivalent \mathcal{L}_T -formula ϕ' . By Theorem 3.1, we obtain an \mathcal{L} -formula ϕ'' , with additional time-difference predicates $Prev_\delta$ and $Prev_{>\delta}$ and time-congruence predicates $Cong_{\leq\delta}$, such that $\Pi_1(\phi') = \Pi(\phi'')$. The \mathcal{L}^2 -formula ψ that binds all of the new time predicates in ϕ'' by an existential prefix is easily seen to have the desired models.

(2) In order to show the second implication, we use a normal-form theorem for \mathcal{L}^2 : given an \mathcal{L}^2 -formula ψ , there is an equivalent \mathcal{L}^2 -formula ψ' of the form $\exists p_1 \dots \exists p_n. \psi'_M$, whose matrix ψ'_M contains no second-order quantifiers [22]. We construct a TPTL-formula ϕ that characterizes the models of ψ' by using the (existentially quantified) time map to encode the interpretation of the unary predicates p_j , $1 \leq j \leq n$, that are bound in ψ' .

Assign to every subset $J_\delta \subseteq \{1, \dots, n\}$ a unique code $\delta \in N$. By the expressive completeness of PTL, there is a PTL-formula ψ''_M such that $\Pi(\psi'_M) = \Pi(\psi''_M)$ [40]. From ψ''_M , we obtain ϕ by replacing every proposition p_j , $1 \leq j \leq n$, with

$$x. \bigcirc y. \bigvee_{j \in J_\delta} y = x + \delta.$$

Now it is straightforward to establish a one-to-many correspondence between the models $I = (\sigma, p_1^I, \dots, p_n^I)$ of ψ'_M and the timed state sequences (σ, T) that satisfy ϕ : given I , let

$T_{i+1} = T_i + \delta$ such that $J_\delta = \{j \mid p_j^i(i)\}$; and given T , let $p_j^i(i)$ iff $j \in J_{T_{i+1}-T_i}$ (assume that $j \notin J_\delta$ if δ is no proper code). ■

It follows that \mathcal{L}_T , with the time function existentially quantified, has the full expressive power of the second-order language \mathcal{L}^2 . In fact, the proof given above shows that equality and successor over the time sort are sufficient to achieve this timeless expressiveness.

Expressive power of congruences

If we disallow the use of congruence relations in TPTL, the resulting logic is strictly less expressive. Consider the following formula ϕ :

$$\Box x. (x \equiv_2 0 \rightarrow p).$$

It characterizes the timed state sequences in which “ p is true at all even times.” We show that this property is not expressible without congruence relations. Suppose that the TPTL-formula ψ , which does not contain any congruence relations, were equivalent to ϕ . Let $c - 1 \in \mathbb{N}$ be the largest constant that occurs in ψ . It is not hard to see that ψ cannot distinguish between the timed state sequences $\rho_1 = (\sigma, \lambda_i. 2ic)$ and $\rho_2 = (\sigma, \lambda_i. (2ic + 1))$, for any state sequence σ . Yet if p is not continuously true in σ , only one of ρ_1 and ρ_2 satisfies ϕ .

Note that TPTL without congruence relations has the same expressive power as the first-order language \mathcal{L}_T without congruences. However, as has been pointed out above, the congruence primitives do not affect the “timeless” expressiveness of these formalisms; for example, we have demonstrated that the property that “ p holds in every even state” (as opposed to every state with an even time) can be defined without congruences.

3.3.3 Timed extended temporal logic

PTL does not have the full expressive power of the second-order language \mathcal{L}^2 : recall that the property *even*(p) — that “ p is true in every even state” —

$$\exists q. (q(0) \wedge \forall i. (q(i) \rightarrow (p(i) \wedge \neg q(i+1) \wedge q(i+2))))),$$

is not expressible in PTL [130]. That is why Wolper has defined *extended temporal logic* (ETL), which includes a temporal operator for every right-linear grammar. ETL has

the same expressiveness as \mathcal{L}^2 or, equivalently, ω -regular expressions, and yet a singly exponential decision procedure.

The situation for TPTL is similar: there is no TPTL-formula whose models are precisely the *timed* state sequences in which, independent of the time map, p holds at every even state. For suppose there were such a formula ϕ ; we show that this would imply the expressibility of $\text{even}(p)$ in \mathcal{L} . First construct an \mathcal{L} -formula ϕ' that is equivalent to ϕ and contains the additional time-difference and time-congruence predicates Prev_δ , $\text{Prev}_{\geq\delta}$, and $\text{Cong}_{c,\delta}$, as usual. Then replace, in ϕ' , all occurrences of Prev_δ , $\text{Prev}_{\geq\delta}$, and $\text{Cong}_{c,\delta}$ by *true* or *false* depending on whether $\delta = c$, $\delta \geq c$, and $\delta = 0$, respectively. This simplification does not affect the truth of the formula over interpretations in which the time increases, starting from 0, always by the constant c from one state to the next. Thus, the resulting formula ψ is satisfied by a state sequence σ iff $(\sigma, \lambda i. ic) \in \Pi_{\mathbb{N}}(\phi)$; that is, iff p is true in every even state of σ .

Fortunately, analogously to PTL, we are able to generalize TPTL to *timed extended temporal logic*, TETL, by introducing temporal grammar operators. Here TETL is shown to have the full expressive power of \mathcal{L}_T^2 ; in Chapter 4 we will prove that it is no more expensive to decide than TPTL.

Syntax and semantics of TETL

Given a set P of propositions symbols and a set V of variables, the terms of TETL are the same as in TPTL. The formulas of TETL are inductively defined as follows:

$$\phi := p \mid \pi_1 \leq \pi_2 \mid \pi_1 \equiv_d \pi_2 \mid \text{false} \mid \phi_1 \rightarrow \phi_2 \mid \mathcal{G}(\phi_1, \dots, \phi_m) \mid x. \phi$$

where $x \in V$, $p \in P$, $d \neq 0$, and $\mathcal{G}(a_1, \dots, a_m)$ is a right-linear grammar with the m terminal symbols a_1, \dots, a_m . (Like ETL, TETL can alternatively be defined using automata connectives for all Büchi-automata, instead of grammar operators [131].)

As with TPTL, TETL-formulas are interpreted over timed state sequences. Given a timed state sequence ρ , a position $i \geq 0$, and an environment \mathcal{E} , the semantics of the grammar operators is defined by the following clause:

$$\rho^i \models_{\mathcal{E}} \mathcal{G}(\phi_1, \dots, \phi_m) \text{ iff there is a (possibly infinite) word } w = a_{w_0} a_{w_1} a_{w_2} \dots \text{ generated by } \mathcal{G}(a_1, \dots, a_m) \text{ such that } \rho^{i+j} \models_{\mathcal{E}} \phi_{w_j} \text{ for all } j \geq 0.$$

As with TPTL, we restrict ourselves to closed formulas of TETL and take every TETL-formula ϕ to define, unless another time domain is given explicitly, the digital property $\Pi(\phi) \subseteq TSS_{\mathbb{N}}^w$.

Note that all temporal operators of TPTL are expressible by the grammar operators of TETL. For example, the TPTL-operator \square corresponds to the grammar $\mathcal{G}_{\square}(a)$ with the only production

$$\mathcal{G}_{\square}(a) \longrightarrow a\mathcal{G}_{\square}(a)$$

(we identify grammars with their starting nonterminal symbols). The formula $even(p)$, which is not expressible in TPTL, can be stated as $\mathcal{G}_{even}(true, p)$, for the production

$$\mathcal{G}_{even}(a_1, a_2) \longrightarrow a_1 a_2 \mathcal{G}_{even}(a_1, a_2).$$

Expressiveness of TETL

By defining the property $even(p)$ in TETL, we have demonstrated that its expressiveness is strictly greater than that of TPTL. The following theorem characterizes the expressiveness of TETL as equivalent to the second-order language \mathcal{L}_T^2 .

Theorem 3.3 (Expressiveness of TETL) *For every formula ϕ of TETL, there exists a formula ψ of \mathcal{L}_T^2 such that $\Pi(\phi) = \Pi(\psi)$, and vice versa.*

Proof of Theorem 3.3 (1) We extend the translation $Classic_0$ that embeds TPTL into \mathcal{L}_T to accommodate the grammar operators of TETL. The target formulas will contain second-order quantifiers over unary predicates, and thus belong to \mathcal{L}_T^2 . For the sake of keeping the presentation simple, we assume that all grammar operators correspond to productions of the form

$$\mathcal{G}(a_1, \dots, a_m) \longrightarrow a_{i_1} \mid a_{i_2} \mathcal{G}'(a_{j_1}, \dots, a_{j_n}).$$

We add the following clause to the definition of $Classic_k$, for $k \geq 0$:

$$Classic_k(\mathcal{G}_0(\phi_1, \dots, \phi_m)) = \exists p_{\mathcal{G}_0} \dots \exists p_{\mathcal{G}_M}. (p_{\mathcal{G}_0}(k) \wedge \forall k' \geq k. \bigwedge_{0 \leq i \leq M} \phi_{\mathcal{G}_i}(k'))$$

for some new unary predicate symbols $p_{\mathcal{G}_0}, \dots, p_{\mathcal{G}_M}$, where $\mathcal{G}_0, \dots, \mathcal{G}_M$ are all the nonterminal symbols that occur in the grammar $\mathcal{G}_0(a_1, \dots, a_m)$, and $\phi_{\mathcal{G}}(k)$ stands for the \mathcal{L}_T^2 -formula

$$p_{\mathcal{G}}(k) \rightarrow (Classic_k(\phi_{i_1}) \vee (Classic_k(\phi_{i_2}) \wedge p_{\mathcal{G}'}(k+1))).$$

Consider an arbitrary timed state sequence ρ . We show, by induction on the structure of ϕ , that $\rho^k \models_{\mathcal{E}} \phi$ iff $\rho^k \models_{\mathcal{E}} \text{Classic}_k(\phi)$ for all $k \geq 0$ and environments \mathcal{E} .

The crucial case that ϕ has the form $\mathcal{G}_0(\phi_1, \dots, \phi_m)$ is derived as follows. To establish the existence of appropriate predicates $p_{\mathcal{G}_l}$, for $0 \leq l \leq M$, let $p_{\mathcal{G}_l}$ be true in state $k' \geq k$ iff $\rho^{k'} \models_{\mathcal{E}} \mathcal{G}_l(\phi_1, \dots, \phi_m)$. On the other hand, given the predicates $p_{\mathcal{G}_l}$ satisfying $\phi_{\mathcal{G}_l}(k')$ for all $k' \geq k$, we can construct a word $w = a_{w_0} a_{w_1} a_{w_2} \dots$ generated by $\mathcal{G}_0(a_1, \dots, a_m)$ such that $\rho^{k'} \models_{\mathcal{E}} \phi_{w_{k'-k}}$. It follows that, for any TETL-formula ϕ , the \mathcal{L}_T^2 -formula $\text{Classic}_0(\phi)$ is equivalent to ϕ .

(2) The argument for the expressive completeness of TETL with respect to \mathcal{L}_T^2 is analogous to the corresponding proof for TPTL and \mathcal{L}_T (use the expressive completeness of ETL with respect to \mathcal{L}^2). ■

Let us complete the expressibility picture by a few remarks. It is not hard to see that both the “untimed” and the “timeless” expressiveness of TETL (as measured by the sets $\Pi_1(\phi)$ and $\Pi_N(\phi)^-$, respectively, for TETL-formulas ϕ) are that of ETL or, equivalently, the second-order language \mathcal{L}^2 . It is also immediate that the congruence relations contribute even to the expressive power of TETL (and \mathcal{L}_T^2) in a nontrivial way, because the property that “ p is true at all even times” is still not expressible without congruence relations.

TPTL with quantification over propositions

There are several alternatives to the grammar operators of ETL. PTL can be extended by fixed-point operators (thus obtaining a variant of the propositional μ -calculus [75]; see also [15, 126]) or second-order quantification over propositions (QPTL of Wolper [129] and Sistla [118]) in order to achieve the full expressive power of \mathcal{L}^2 . While fixed-points can be viewed as generalized grammar operators and yield to verification algorithms, QPTL is nonelementary. It is straightforward to show that both extensions have, indeed, the expected, analogous effect in the TPTL-framework; they give decidable real-time specification languages with the expressiveness of \mathcal{L}_T^2 . However, *timed* QPTL is, as a superset of QPTL, nonelementary, and thus unsuitable as a finite-state verification formalism.

3.3.4 Nonelementary extensions

In the Chapter 4, we will see that the decision problem for TPTL (as well as TETL) is in EXPSPACE. We have seen that TPTL restricts \mathcal{L}_T to "temporal" quantification over the state sort and no quantification over the time sort. Can we relax the restrictions without sacrificing elementary decidability? Arbitrary quantification over the state sort encompasses full \mathcal{L} and is, therefore, clearly nonelementary. In this subsection, we first study the generalization of TPTL that admits quantification over the time sort, and show it to be nonelementary as well. Then we try to add past temporal operators to TPTL, an extension that does not affect the complexity of PTL. Therefore it is quite surprising that the past operators render TPTL nonelementary.

TPTL with quantification over time

Recall that several authors have proposed to use first-order temporal logic with the a single state variable t , which represents the time in every state, for the specification of real-time properties. For instance, the bounded-response property Π_1^\diamond has been defined in "real-time temporal logic" by the formula

$$\forall x. \Box(p \wedge t = x \rightarrow \Diamond(q \wedge t \leq x + 1)),$$

which uses auxiliary rigid variables like x to refer to the time (i.e., the value of t) of different temporal contexts. Eliminating the state variable t , we see that this notation corresponds to TPTL extended by classical universal and existential first-order quantification over time variables:

$$\forall x. \Box y. (p \wedge y = x \rightarrow \Diamond z. (q \wedge z \leq x + 1)).$$

We call this generalization of TPTL, whose syntax definition is supplemented by the new clause

if ϕ is a formula and $x \in V$, then $\exists x. \phi$ is also a formula,

quantified TPTL or TPTL $_{\exists}$. Given a timed state sequence ρ , a position $i \geq 0$, and an environment \mathcal{E} , the classical quantifiers are interpreted as usual:

$$\rho^i \models_{\mathcal{E}} \exists x. \phi \text{ iff } \rho^i \models_{\mathcal{E}[x:=t]} \phi \text{ for some } t \in \text{TIME}.$$

TPTL_∃ seems, on the surface, more expressive than TPTL, because it can state properties of times that are not associated with any state. But it is easy to see that TPTL_∃ can still be embedded into \mathcal{L}_T ; let

$$\text{Classic}_i(\exists x. \phi) = \exists x. \text{Classic}_i(\phi).$$

The satisfiability of TPTL_∃ is, therefore, decidable, and its expressive power, measured as the sets of timed state sequences specifiable in the logic, is the same as that of TPTL. We show that TPTL_∃, however, is not elementarily decidable. This provides additional justification for our preference for TPTL over the existing notation with classical quantifiers over time: prohibiting quantification over time not only leads, as was argued above, to a more natural specification language, but is *necessary* for the existence of finite-state verification algorithms.

Theorem 3.4 (Complexity of TPTL_∃) *The validity problem for TPTL_∃ is nonelementary.*

Proof of Theorem 3.4 We translate the nonelementary monadic first-order theory of (\mathbb{N}, \leq) [121] into TPTL_∃. With the help of the formula

$$\Box x. \bigcirc y. y = x + 1, \quad (\phi_{+1})$$

we can force time to act as a state counter; then we can simulate state quantifiers by the time quantifiers of TPTL_∃. Given a formula ϕ of \mathcal{L} , we construct a formula ψ of TPTL_∃ such that ϕ is valid iff the TPTL_∃-formula $\phi_{+1} \rightarrow \psi$ is valid. The formula ψ is obtained from ϕ by replacing every atomic subformula of the form $p(i)$ with $\bigcirc x. (p \wedge x = i)$ (read the quantifiers of ϕ as quantifiers over the time sort). ■

TPTL with past

Lichtenstein, Pnueli, and Zuck [84] extended PTL with the past temporal operators \ominus (*previous*) and \mathcal{S} (*since*), the duals of \bigcirc and \mathcal{U} . These operators can be added at no extra cost, and although they do not increase the expressive power of PTL, they allow a more direct and convenient expression of certain properties. Let TPTL_P be the logic that results from TPTL by adding the following clause to the inductive definition of formulas:

if ϕ_1 and ϕ_2 are formulas, then so are $\neg\phi_1$ and $\phi_1 \mathcal{S} \phi_2$.

The meaning of the past operators is given by

$$\rho^i \models_{\mathcal{E}} \neg\phi \text{ iff } i = 0 \text{ or } \rho^{i-1} \models_{\mathcal{E}} \phi.$$

$$\rho^i \models_{\mathcal{E}} \phi_1 \mathcal{S} \phi_2 \text{ iff } \rho^j \models_{\mathcal{E}} \phi_2 \text{ for some } j \leq i \text{ and } \rho^k \models_{\mathcal{E}} \phi_1 \text{ for all } j < k \leq i.$$

Clearly, TPTL_P can still be embedded into \mathcal{L}_T :

$$\text{Classic}_0(\neg\phi) = \text{true},$$

$$\text{Classic}_{i+1}(\neg\phi) = \text{Classic}_i(\phi),$$

$$\text{Classic}_i(\phi_1 \mathcal{S} \phi_2) = \exists j \leq i. (\text{Classic}_j(\phi_2) \wedge \forall j < k \leq i. \text{Classic}_k(\phi_1)).$$

Hence the satisfiability of this logic is, again, decidable, and its expressive power is no greater than that of TPTL. However, unlike in the case of PTL, there is a surprisingly heavy price to be paid for adding the past operators.

Theorem 3.5 (Complexity of TPTL_P) *The validity problem for TPTL_P is nonelementary.*

Proof of Theorem 3.5 Again, we are able to use the nonelementary nature of the monadic first-order theory of (\mathbb{N}, \leq) . By adopting time as a state counter, we can simulate true existential quantification over time by \diamond , because \diamond allows us to restore the correct temporal context. Given a formula ϕ of \mathcal{L} , we construct a formula ψ of TPTL_P such that ϕ is valid iff the TPTL_P -formula $\phi_{+1} \rightarrow \psi$ is valid.

The first step in translating ϕ is the same as in the proof of the nonelementary complexity of TPTL_{\exists} . In a second step we replace every subformula of the form $\exists x. \varphi$ with the formula

$$y. (\diamond x. \diamond z. (z = y \wedge \varphi) \vee \diamond x. \diamond z. (z = y \wedge \varphi)).$$

■

3.4 Metric Temporal Logic

Several authors have tried to adapt temporal logic to reason about real-time properties by interpreting modalities as real-time operators. For instance, Koymans suggested the notation $\Diamond_{\leq c}$ to express the notion “eventually within c time units” [71]. Similar temporal operators that are parameterized with constant bounds have been used by Pnueli and Harel [110] as well as by Emerson, Mok, Sistla, and Srinivasan [37]. In this section, we extend PTL by time-bounded temporal operators and interpret the resulting logic over timed state sequences. For example, the bounded-invariance property Π_5^{\square} — that “no p -state is followed by a q -state within less than 5 time units” — will be written as

$$\Box(p \rightarrow \Box_{<5} \neg q); \quad (\phi_5^{\square})$$

the bounded-response property Π_1^{\diamond} — that “every p -state is followed by a q -state within 1 time unit” — will be expressed by the formula

$$\Box(p \rightarrow \Diamond_{\leq 1} q). \quad (\phi_1^{\diamond})$$

It is easy to see that we have, in fact, only obtained a notational variant of a subset of TPTL (rewrite, for example, every subformula $\Diamond_{\leq c} \phi$ as $x. \Diamond y. (y \leq x + c \wedge \phi)$).

We will show that PTL with bounded temporal operators is interesting, and worth studying in its own right, for two reasons. First, and surprisingly, it is already as expressive as full TPTL. And secondly, it may, unlike full TPTL, be augmented by past temporal operators without sacrificing its elementary decidability. Following Koymans, we call the resulting language, which includes past operators, *metric temporal logic* (MTL). We will conclude that MTL, too, represents a suitable formalism for the specification and verification of real-time properties: just like TPTL, MTL captures an expressively complete and yet elementary fragment of \mathcal{L}_T . But the two subsets of \mathcal{L}_T that correspond to TPTL and MTL, respectively, are not identical. We will illustrate that either logic can state certain properties more directly and succinctly than the other one and may therefore be preferred for some specifications.

3.4.1 Syntax and semantics

Given a set of propositions P , the formulas ϕ of MTL are defined inductively as follows:

$$\phi := p \mid \text{false} \mid \phi_1 \rightarrow \phi_2 \mid \odot_I \phi \mid \otimes_I \phi \mid \phi_1 \mathcal{U}_I \phi_2 \mid \phi_1 \mathcal{S}_I \phi_2$$

for $p \in P$. The subscript I is one of the following:

1. A possibly unbounded *interval* of *TIME* whose end-points are natural number constants. Intervals may be open, half-open, or closed; empty, bounded, or unbounded. We say that all intervals of the forms $[0, d)$, $(c, d]$, for $c \in \mathbb{N}$, $d \in \mathbb{N} \cup \{\omega\}$, and $c \leq d$, are *open*; all intervals of the forms $[c, d]$ and $[c, \infty)$, for $c, d \in \mathbb{N}$ and $c \leq d$, are *closed*. We freely denote intervals by pseudo-arithmetic expressions. For example, the expressions $\leq d$ and $> c$ stand for the intervals $[0, d]$ and (c, ∞) , respectively. The expression $\delta + I$, where I is an interval and $\delta \in \text{TIME}$, denotes the interval $\{\delta + t \mid t \in I\}$.
2. A *congruence expression* of the form $\equiv_d c$, for $c, d \in \mathbb{N}$ and $d \neq 0$. Congruence expressions are, as usual, prohibited in the analog-clock model.

We remark that we do not choose to bound *both* arguments of the *until* and *since* operators by subscripts, simply because we feel that doing so would impair the readability of formulas and, as we shall see, would not increase the expressive power of the logic. As in the case of TPTL, we assume that all constants in an MTL-formula are given in a binary encoding.

Timed state sequence semantics

The formulas of MTL are interpreted over timed state sequences. Instead of giving MTL its own semantics, we translate every MTL-formula ϕ into a TPTL_P-formula $\text{Freeze}(\phi)$:

$$\begin{aligned} \text{Freeze}(p) &= p, \\ \text{Freeze}(\text{false}) &= \text{false}, \\ \text{Freeze}(\phi_1 \rightarrow \phi_2) &= \text{Freeze}(\phi_1) \rightarrow \text{Freeze}(\phi_2), \\ \text{Freeze}(\odot_I \phi) &= x. \bigcirc y. (y \in x + I \wedge \phi), \\ \text{Freeze}(\ominus_I \phi) &= x. \ominus y. (y \in x - I \wedge \phi), \\ \text{Freeze}(\phi_1 \mathcal{U}_I \phi_2) &= x. (\phi_1 \mathcal{U} y. (y \in x + I \wedge \phi_2)), \\ \text{Freeze}(\phi_1 \mathcal{S}_I \phi_2) &= x. (\phi_1 \mathcal{S} y. (y \in x - I \wedge \phi_2)) \end{aligned}$$

such that

1. If I is an interval of the form $[c, d)$, then the expressions $x \in y + I$ and $x \in y - I$ stand for the timing constraints $y + c \leq x < y + d$ and $x + c \leq y < x + d$, respectively. It is straightforward to fill in which timing constraints of TPTL are denoted by other interval expressions.

2. If I is a congruence expression the form $\equiv_d c$, then the expressions $x \in y + I$ and $x \in y - I$ both stand for the timing constraint $x \equiv_d c$.

We take an MTL-formula ϕ to define the same property as the TPTL_P-formula $\text{Freeze}(\phi)$:

$$\Pi(\phi) = \Pi(\text{Freeze}(\phi)).$$

It follows that every MTL-formula defines, in the analog-clock model, the analog property $\Pi_R(\phi) \subseteq TSS_R^a$; in the digital-clock model, the digital property $\Pi_N(\phi) \subseteq TSS_N^d$; in the untimed model, the untimed property $\Pi_1(\phi) \subseteq TSS_1^u$. As for TPTL, the digital-clock model is taken to be the default for MTL. Note that in the digital-clock model, for every MTL-formula there is an equivalent formula that contains only *closed* time intervals (for example, replace any subformula $\phi_1 \mathcal{U}_{(c,d)} \phi_2$ with $\phi_1 \mathcal{U}_{[c+1,d-1]} \phi_2$ if $c < d$, and with *false* if $c = d$).

Additional bounded temporal operators can be defined in terms of the given MTL-operators \odot_I (*bounded strong next*) and \mathcal{U}_I (*bounded until*) as follows:

Bounded weak next $\ominus_I \phi$ stands for $\neg \odot_I \neg \phi$.

Bounded eventually $\diamond_I \phi$ stands for $\text{true} \mathcal{U}_I \phi$.

Bounded always $\square_I \phi$ stands for $\neg \diamond_I \neg \phi$.

Bounded unless $\phi_1 \cup_I \phi_2$ stands for $\phi_1 \mathcal{U}_I \phi_2 \vee \square_{[0,\infty)} \phi_1$.

Note that while the *strong-next* formula $\odot_{=2} p$ is satisfied by a timed state sequence iff the "next" (i.e., second) state is a p -state *and* its time is 2 greater than the time of the "current" (i.e., initial) state, the *weak-next* formula $\ominus_{=2} p$ requires that the second state is a p -state only *if* the time increase between the first and the second state is 2. Similarly, for any time interval I , the MTL-formula $\diamond_I p$ asserts that there is a p -state with a time that is within the interval I of the "current" (initial) time; the MTL-formula $\square_I p$ stipulates that all states in that interval are p -states (although there may be none):

$$\rho \models \diamond_I \phi \text{ iff } \rho^i \models \phi \text{ for some } i \geq 0 \text{ with } T_i \in T_0 + I.$$

$$\rho \models \square_I \phi \text{ iff } \rho^i \models \phi \text{ for all } i \geq 0 \text{ with } T_i \in T_0 + I.$$

On the other hand, the MTL-formula $\diamond_{\equiv,1} p$ is true of a timed state sequence iff p is true in some state whose time is odd (independent of the initial time). While time intervals constrain the time *differences* between states, congruence expressions refer to the *absolute* times of states.

We usually suppress the universal interval $[0, \infty)$ as a subscript. Thus the MTL-operators \odot (which is equivalent to \ominus), \diamond , \square , \mathcal{U} , and \mathcal{U} coincide with the conventional unbounded *next*, *eventually*, *always*, *until*, and *unless* operators of PTL. More precisely, every MTL-formula ϕ without operator subscripts specifies a time-invariant property and can be read as a formula ψ of PTL:

$$\Pi(\phi)^- = \Pi_1(\phi) = \Pi(\psi).$$

It follows that MTL, like TPTL, is a conservative extension of PTL.

From our embedding of MTL into $\text{TPTL}_{\mathcal{P}}$, it follows that MTL is decidable, and that both TPTL and MTL are orthogonal fragments of $\text{TPTL}_{\mathcal{P}}$ and, hence, $\mathcal{L}_{\mathcal{T}}$: while TPTL prohibits past operators, MTL corresponds to a subset of $\text{TPTL}_{\mathcal{P}}$ wherein all timing constraints relate only variables that refer to "adjacent" temporal contexts. In the following subsection, we will show that, like TPTL, MTL selects also an expressively complete subset of $\mathcal{L}_{\mathcal{T}}$. First, however, let us indicate another possible semantics for MTL.

Interval semantics

It is obvious that the properties that are definable in MTL (or TPTL) are not necessarily (weakly) closed under stuttering. There are two reasons why MTL-formulas can detect stuttering:

1. MTL contains *next* operators. Thus we may wish to ban the *next* operators from specifications, as has been argued by Lamport for PTL [78]. Observe, for example, that both bounded-invariance and bounded-response properties can be defined without *next* operators. On the other hand, when we represent timed transition systems by temporal formulas in Chapter 4, we will find it convenient to use *next* operators to define the next-state relation of a transition system.
2. MTL is interpreted over observation sequences (i.e., timed state sequences) rather than state interval sequences. For example, the MTL-formula $\square_{[3,8]} \text{false}$ specifies a

property that is not weakly closed under stuttering. However, while the requirement $\Box_{[3,8]}$ *false* is, perhaps undesirably, satisfied by some observation sequences (if there is no observation between time 3 and time 8), it cannot be satisfied by any state interval sequence, which has one or more states at every point in time. This phenomenon can be exploited by interpreting real-time languages over state interval sequences.

Recall that state interval sequences correspond to deterministic timed state sequences. Indeed, over deterministic timed state sequences we can give MTL (and TP TL, for that matter) an *interval* semantics such that every property that can be defined by a formula without *next* operators is closed under stuttering. We have pursued this approach at a different opportunity [9]; here, we present only the interval interpretation of a small fragment of MTL. Let $\rho = (\sigma, T)$ be a deterministic timed state sequence and I a time interval:

$$\rho \models p \text{ iff } p \in \sigma_0.$$

$$\rho \not\models \text{false}.$$

$$\rho \models \phi_1 \rightarrow \phi_2 \text{ iff } \rho \models \phi_1 \text{ implies } \rho \models \phi_2.$$

$$\rho \models \Diamond_I \phi \text{ iff } \rho' \models \phi \text{ for some } \delta\text{-suffix } \rho' \text{ of } \rho \text{ with } \delta \in I.$$

This interval interpretation of MTL-formulas implies that

$$\rho \models \Box_I \phi \text{ iff } \rho' \models \phi \text{ for every } \delta\text{-suffix } \rho' \text{ of } \rho \text{ with } \delta \in I$$

and, in particular, that $\rho \not\models \Box_I \text{false}$ if I is not empty.

3.4.2 Expressive completeness

Because of the past operators, MTL can express certain properties more succinctly than TP TL. Recall, for example, the traffic-light controller from Subsection 2.2.3. Requirement (A) — that within 5 time units from any request, the light turns green and stays green for at least 5 time units — can be naturally specified in both logics:

$$\Box x. (\text{request} \rightarrow \Diamond y. (y \leq x + 5 \wedge \Box z. (z \leq y + 5 \rightarrow \text{light} = \text{green}))),$$

$$\Box (\text{request} \rightarrow \Diamond_{\leq 5} \Box_{\leq 5} (\text{light} = \text{green})).$$

But requirement (B) — that the light should be red if there has not been a request for 25 time units — belongs to the class of properties that assert that every effect was preceded by a cause, which are most naturally expressed by past operators:

$$\Box(\text{light} = \text{green} \rightarrow \Diamond_{\leq 25} \text{request}).$$

On the other hand, consider the following TPTL-formula, which asserts that “every p -state is followed by a q -state and, later, an r -state within 5 time units”:

$$\Box x.(p \rightarrow \Diamond(q \wedge \Diamond y.(r \wedge y \leq x + 5))).$$

This property has no natural expression in MTL. However, in the digital-clock model, the discrete nature of time can be exploited to translate the property into MTL:

$$\Box(p \rightarrow \bigvee_{c=0}^5 \Diamond_{=c}(q \wedge \Diamond_{\leq 5-c} r)).$$

In fact, we show that in the digital-clock model the expressiveness of MTL is no less than that of TPTL in any crucial way. Only properties that need to be defined, in TPTL, using absolute time references cannot be defined in our version of MTL. These properties put constraints on the absolute time of some states in a timed state sequence, such as “the time of the initial state is 2” ($x.x = 2$ in TPTL). Thus the inability of MTL to express absolute time references is of no importance to the analysis of timed transition systems, whose computations are closed under shifting the origin of time.

Formally, we say that a timed state sequence (σ, T) is *initial* iff the time of its initial state is 0; that is, $T_0 = 0$. The following theorem states that if the expressiveness of a logic is measured by the sets of initial timed state sequences that are definable, then MTL has the same expressive power as \mathcal{L}_T or, equivalently, TPTL.

Theorem 3.6 (Expressive completeness of MTL) *For every formula ϕ of \mathcal{L}_T , there exists a formula ψ of MTL (without past operators) such that $\rho \models \phi$ iff $\rho \models \psi$ for every initial timed state sequence $\rho \in TSS_N^w$.*

Proof of Theorem 3.6 As in the proof of the expressive completeness of TPTL, given a formula ϕ of \mathcal{L}_T , we construct a PTL-formula ϕ' , with additional time-difference propositions $Prev_{\leq \delta}$ and $Prev_{> \delta}$ and time-congruence propositions $Cong_{c,\delta}$, such that $\Pi_1(\phi) = \Pi(\phi')$.

Furthermore, in ϕ' all of the propositions $Prev_\delta$, $Prev_{\geq\delta}$, and $Cong_{c,\delta}$ are either not within the scope of any temporal operator, or immediately preceded by a *next* operator.

From ϕ' we obtain the desired MTL-formula ψ by eliminating the time-difference and time-congruence propositions as follows. Since we consider only initial models, we replace each proposition $Prev_\delta$, $Prev_{\geq\delta}$, and $Cong_{c,\delta}$ that is not within the scope of any temporal operator by *true* or *false*, depending on whether $\delta = 0$. Then we replace each subformula $\bigcirc Prev_\delta$ by $\bigcirc_{=0} true$, each subformula $\bigcirc Prev_{\geq\delta}$ by $\bigcirc_{\geq\delta} true$, and each subformula $\bigcirc Cong_{c,\delta}$ by $\bigcirc_{=c\delta} true$. (Observe that only the *strong-next* operator needs to be subscripted.) ■

3.5 Real-time Properties That Cannot Be Verified

At last, let us justify our decisions to

1. restrict the *semantics* of TPTL, MTL, and the classical theory of timed state sequences to the digital-clock model (i.e., the discrete time domain \mathbb{N}), and
2. restrict the *syntax* of timing constraints in TPTL, MTL, and the classical theory of timed state sequences to comparison, successor, and congruence operations on time.

Indeed, both decisions seem overly limiting for real-time specification languages. For example, without addition of time values the property that “the time difference between subsequent p -states increases forever” cannot be defined. We show, however, that both restrictions are absolutely necessary to obtain formal verification techniques; without them there exist neither decision procedures nor complete proof systems.

3.5.1 Timed temporal logic revisited: Undecidable extensions

We consider two natural extensions of TPTL, a syntactic one (allowing addition over time) and a semantic one (interpreting TPTL-formulas over a dense time domain). Both extensions are shown to be Π_1^1 -complete, by reducing a Σ_1^1 -hard problem of 2-counter machines to the respective satisfiability problems. It follows that they cannot even be (recursively) axiomatized (for an exposition of the analytical hierarchy consult, for instance, the book by Rogers [113]).

A Σ_1^1 -complete problem

A *nondeterministic 2-counter machine* M consists of two counters C and D , and a sequence of n instructions, each of which may increment or decrement one of the counters, or jump, conditionally upon one of the counters being zero. After the execution of a non-jump instruction, M proceeds nondeterministically to one of two specified instructions. We represent the configurations of M by triples $\langle i, c, d \rangle$, where $0 \leq i < n$, $c \geq 0$, and $d \geq 0$ are the current values of the location counter and the two counters C and D , respectively. The consecution relation on configurations is defined in the obvious way. A *computation* of M is an infinite sequence of related configurations, starting with the initial configuration $\langle 0, 0, 0 \rangle$. It is called *recurring* iff it contains infinitely many configurations with the value of the location counter being 0.

The problem of deciding whether a nondeterministic Turing machine has, over the empty tape, a computation in which the starting state is visited infinitely often, has been shown Σ_1^1 -complete by Harel, Pnueli, and Stavi [52]. Along the same lines we obtain the following result.

Lemma 3.2 (Complexity of 2-counter machines) *The problem of deciding whether a given nondeterministic 2-counter machine has a recurring computation, is Σ_1^1 -hard.*

Proof of Lemma 3.2 Every Σ_1^1 -formula is equivalent to a Σ_1^1 -formula χ of the form

$$\exists f. (f(0) = 1 \wedge \forall x. g(f(x), f(x+1))),$$

for a recursive predicate g [52]. For any such χ we can construct a nondeterministic 2-counter machine M that has a recurring computation iff χ is true.

Let M start by computing $f(0) = 1$, and proceed, indefinitely, by nondeterministically guessing the next value of f . At each stage, M checks whether $f(x)$ and $f(x+1)$ satisfy g , and if (and only if) so, it jumps to instruction 0. Such an M exists, because 2-counter machines can, being universal, compute the recursive predicate g . It executes the instruction 0 infinitely often iff a function f with the desired properties exists. ■

Encoding computations of 2-counter machines

We show that the satisfiability problem for several extensions of TPTL is Σ_1^1 -complete. First, we observe that the satisfiability of a formula ϕ can, in all cases, be phrased as a

Σ_1^1 -sentence, asserting the existence of a model for ϕ . Any timed state sequence ρ for ϕ can be encoded, in first-order arithmetic, by finitely many infinite sets of natural numbers; say, one for each proposition p in ϕ , characterizing the states in which p holds, and one to encode state-time pairs. It is routine to express, as a first-order predicate, that ϕ holds in ρ . We conclude that satisfiability is in Σ_1^1 .

To show that the satisfiability problem of a logic is Σ_1^1 -hard, it suffices, given a non-deterministic 2-counter machine M , to construct a formula ϕ_M such that ϕ_M is satisfiable iff M has a recurring computation. We demonstrate the technique of encoding recurring computations of M by showing that the *monotonicity* constraint on time is necessary for the decidability of TPTL. (From the decision procedure for TPTL that will be presented in Chapter 4 it follows that the *progress* requirement on time may, unlike monotonicity, be relaxed without affecting the complexity of TPTL.)

Theorem 3.7 (Nonmonotonic time) *Relaxing the monotonicity condition for timed sequences renders the satisfiability problem for TPTL Σ_1^1 -complete.*

Proof of Theorem 3.7 We encode the computation Γ of M by the divergent "time" sequence \top such that, for all $k \geq 0$, $\top_{4k} = i$, $\top_{4k+1} = n + c$, $\top_{4k+2} = n + d$, and $\top_{4k+3} = n + k$ for the k -th configuration (i, c, d) of Γ . Now it is easy to express, by a TPTL-formula ϕ_M , that a time sequence encodes a recurring computation of M . First specify the initial configuration, by

$$x.x = 0 \wedge \bigcirc x.x = n \wedge \bigcirc^2 x.x = n \wedge \bigcirc^3 x.x = n \quad (\phi_{INIT})$$

(we abbreviate a sequence of m next operators by \bigcirc^m). Then ensure proper consecution by adding a \square -conjunct ϕ_i for every instruction i of M . For instance, the instruction 1 that increments the counter C and proceeds, nondeterministically, to either instruction 2 or 3, contributes the conjunct

$$\square x. \left(x = 1 \rightarrow \left(\begin{array}{l} \bigcirc^4 y. (y = 2 \vee y = 3) \wedge \\ \bigcirc y. \bigcirc^4 z. z = y + 1 \wedge \\ \bigcirc^2 y. \bigcirc^4 z. z = y \wedge \\ \bigcirc^3 y. \bigcirc^4 z. z = y + 1 \end{array} \right) \right) \quad (\phi_1)$$

The recurrence condition can be expressed by a $\square\Diamond$ -formula:

$$\square\Diamond x.x = 0. \quad (\phi_{RECUR})$$

Clearly, the conjunction ϕ_M of these $n + 2$ formulas is satisfiable iff M has a recurring computation. \square

Note that we do not require any propositions in the proof. It follows that first-order temporal logic with a single state variable ranging over the natural numbers is Π_1^1 -complete, provided the underlying assertion language has at least successor (in addition to equality) as a primitive.

Presburger TPTL

We show that a certain extremely modest relaxation of the syntax of timing constraints leads to a highly undecidable logic. Consequently, TPTL with addition over time is undecidable.

Theorem 3.8 (Presburger TPTL) *If the syntax of TPTL is extended to allow multiplication by 2, the satisfiability problem becomes Σ_1^1 -complete.*

Proof of Theorem 3.8 To encode computations of M , we use the propositions p_1, \dots, p_n, r_1 , and r_2 , precisely one of which is true in any state; hence we may identify states with propositions. The configuration (i, c, d) of M is represented by the finite sequence $p_i r_1^c r_2^d$ of states.

The initial configuration (p_0) as well as the recurrence condition $(\Box \Diamond p_0)$ can be easily expressed in PTL. The crucial property that allows a temporal logic to specify the consecution relation of configurations, and thus the set of computations of M , is the ability to copy an arbitrary number of r -states. In real-time temporal logics, the times that are associated with a state sequence can be used for copying. With the availability of multiplication by 2, we are able to have the k -th configuration of a computation correspond, for all $k \geq 0$, to the finite sequence of states that is mapped to the time interval $[2^k, 2^{k+1})$. First, we force the time to increase by a strictly positive amount between successive states $(\Box x. \bigcirc y. y > x)$, to ensure that every state is uniquely identifiable by its time. Then we can copy groups of r -states by establishing a one-to-one correspondence of r_j -states ($j = 1, 2$) at time t and time $2t$; clearly there are enough time gaps to accommodate an additional r_j -state when required by an increment instruction.

For instance, the instruction 1 that increments the counter C and proceeds, nondeterministically, to either instruction 2 or 3, can be expressed as follows:

$$\square x. \left(\begin{array}{l} p_1 \rightarrow \diamond z. (z = 2x \wedge (p_2 \vee p_3)) \wedge \\ \square y_1. \bigcirc y_2. (y_2 < 2x \rightarrow \diamond z_1. (z_1 = 2y_1 \wedge \bigcirc z_2. z_2 = 2y_2)) \wedge \\ \bigwedge_{i=1,2} \square y. (y < 2x \wedge r_i \rightarrow \diamond z. (z = 2y \wedge r_i)) \wedge \\ \square y_1. \bigcirc y_2. (y_2 = 2x \rightarrow \diamond z_1. (z_1 = 2y_1 \wedge \bigcirc z_2. r_1 \wedge \bigcirc \bigcirc z_3. z_3 = 2y_2)) \end{array} \right)$$

The first conjunct ensures the proper progression to one of the two specified instructions, 2 or 3; the second one establishes a one-to-one correspondence between states in successive intervals representing configurations, while the third and fourth conjuncts copy r_j -states ($j = 1, 2$). The last conjunct adds, finally, an r_1 -state at the end of the successor configuration, as required by the increment operation. ■

We can modify this proof by reducing time to a state counter ($\square x. \bigcirc y. y = x + 1$), and letting all propositions be false in the resulting additional (padding) states. Thus, the satisfiability problem for TPTL with multiplication by 2 is Σ_1^1 -hard even if time is replaced by a state counter. As a corollary we infer that the first-order theory of the natural numbers with \leq , multiplication by 2, and monadic predicates is Π_1^1 -complete. A similar result was obtained independently by Halpern [49], who showed that Presburger arithmetic becomes Π_1^1 -complete with the addition of a single unary predicate.

Dense TPTL

Another possible direction to extend the expressive power of TPTL is to relax its semantics by adopting a dense time domain; that is, between any two given points in time there is another time point. We show that the resulting logic is, again, highly undecidable.

Theorem 3.9 (Dense TPTL) *If TPTL is interpreted over the rational numbers (i.e., $\text{TIME} = \mathbb{Q}$), the satisfiability problem becomes Σ_1^1 -complete.*

Proof of Theorem 3.9 The proof depends, once more, on the ability to copy groups of r -states. This time, we are able to have the k -th configuration of a computation of M correspond, for all $k \geq 0$, to the finite sequence of states that is mapped to the time interval $[k, k + 1)$, because dense time allows us to squeeze arbitrarily many states into every interval of length 1. Again, we identify every state with a unique time, and can then establish a

one-to-one correspondence of r_j -states ($j = 1, 2$) at time t and time $t + 1$. In fact, we may simply replace all occurrences of multiplication by 2 in the Presburger-TPTL formula encoding the recurring computations of M , by a successor operation, in order to obtain the desired dense-TPTL formula ϕ_M . \square

This proof goes through for any time domain $(T, <, S)$ such that $(T, <)$ is a dense linear order, and S is a unary function over T that satisfies the following two first-order axioms:

$$\forall x. x < S(x),$$

$$\forall x, y. (x < y \rightarrow S(x) < S(y)).$$

To show that, for arbitrary dense time domains, the satisfiability problem is in Σ_1^1 , a standard Löwenheim-Skolem argument is necessary to infer the existence of countable models. It follows, in particular, that the validity problem for TPTL is Π_1^1 -complete in the analog-clock model.

3.5.2 The classical theory revisited: Undecidable extensions

Now we justify our claim that "the" classical theory of timed state sequences builds on the theory $(\mathbb{N}, \leq, \equiv)$ of time. The reason is, once again, that more expressive theories of time cause Π_1^1 -hardness. In fact, the proof technique of encoding 2-counter machines is extremely robust and can be used to show that a wide variety of real-time specification languages with an expressive power greater than that of \mathcal{L}_T^2 have highly undecidable decision problems. This suggests that we have been able to characterize an intrinsic boundary between the decidability and undecidability of formalisms that combine propositional reasoning about state sequences with first-order reasoning about time.

Theorem 3.10 (Undecidable theories of real time) *The validity problems for the following two-sorted first-order theories are Π_1^1 -complete:*

	<i>state theory</i>	<i>time theory</i>	<i>time function</i> (from states to time)
1	(\mathbb{N}, \leq)	$(\mathbb{N}, +1)$	f
2	(\mathbb{N}, \leq) with <i>monadic predicates</i>	$(\mathbb{N}, \cdot 2)$	<i>identity</i> f

	<i>state theory</i>	<i>time theory</i>	<i>time function</i> (<i>from states to time</i>)
3	(\mathbb{N}, \leq) with monadic predicates	dense linear order (D, \prec) with "successor" S : $x \prec S(x)$ $x \prec y \rightarrow S(x) \prec S(y)$	strictly monotonic f
4	(\mathbb{N}, \leq) with monadic predicates	$(\mathbb{N}, +1)$	identity f and strictly monotonic f'

Proof of Theorem 3.10 First, we observe that the satisfiability of a formula ϕ can, in all cases, be phrased as a Σ_1^1 -sentence that asserts the existence of a model for ϕ . In Case 3, the Löwenheim-Skolem theorem ensures the existence of countable models. Thus the satisfiability problem is in Σ_1^1 in each case. To prove Σ_1^1 -hardness, we use again Lemma 3.2; that is, we show that the satisfiability problem of a language is Σ_1^1 -hard by constructing a formula ϕ_M such that ϕ_M is satisfiable iff a given nondeterministic 2-counter machine M has a recurring computation.

(1)–(3) Σ_1^1 -hardness of the cases 1 through 3 follows immediately from the proofs of the Theorems 3.7, 3.8, and 3.9, respectively. The TPTL-formulas that are given there to the encode computations of M can be directly translated into classical first-order formulas ϕ_M that satisfy, in each case, the required restrictions. For example, in case 1 the result ϕ_M of the translation $Classic_0$ uses only the successor primitive over time and no unary predicates.

(4) This case corresponds to having *two* time bases (or "clocks") f and f' that are updated, from one state to the next, independently of each other. The result holds already for the special case in which f is the identity function and f' is strictly increasing.

The encoding of M -computations is very similar to the one used in the proof of Theorem 3.8 to establish the Σ_1^1 -hardness of case 2. We use the unary predicates p_1, \dots, p_n , r_1 , and r_2 and require that at most one of these predicates is true of any state; hence we may identify states with predicate symbols. The configuration $\langle l, c, d \rangle$ of M is represented by a sequence of 2^k states in the time interval $[2^k, 2^{k+1})$ that starts with a p_1 -state, and contains precisely c r_1 -states and d r_2 -states. Even though the assertion language does not include the primitive of multiplication by 2, which is used to copy states, multiplication

by 2 can be simulated with the help of the second time function f' . This can be done by restricting ourselves to interpretations in which $f'(i) = 2i$ for all $i \geq 0$, which is enforced by the formula

$$f'(0) = 0 \wedge \forall i. (f'(i+1) = f'(i) + 2).$$

■

This theorem demonstrates that extending the syntax or semantics of \mathcal{L}_T in one of any number of directions causes tremendous undecidability. On the other hand, we have shown that the congruence primitives over time can be added to the language without sacrificing decidability. Furthermore, we have proved decidability for the second-order case of \mathcal{L}_T^2 as well. Thus we claim that the first-order theory of (N, \leq) with monadic predicates (for state sequences) combined with the theory of (N, \leq, \equiv) (for time) is *the* theory of timed state sequences.

3.5.3 Decidability versus undecidability in real-time logics

Let us consider the wide-ranging implications of Theorem 3.10 for designing real-time specification languages. We look at the ramifications of all four cases of the theorem. The fact that the monotonicity constraint on the time function is required for decidability (case 1) has little consequences in the context of real-time logics, because we are interested only in monotonic time functions anyway.

Choosing the domain of time

When devising a real-time logic we need to select an appropriate mathematical domain to represent time. Ideally, to model systems whose state changes can occur arbitrarily close in time, we like to choose a dense linear order, such as the analog-clock model. Since the ordering predicate and addition by constant time values are the basic primitives needed to express the simplest of timing constraints, the undecidability of the resulting theory (case 3) is a major stumbling block in the design of useful logics over dense time. In fact, a close inspection of our proof reveals that even the satisfiability of a very simple class of real-time properties is undecidable in the analog-clock model: the only timing constraints required to copy sequences of states are, using the notation of MTL, of the form

$$\square \odot_{>0} \text{true},$$

so that any time identifies a unique state, and of the forms

$$\Box(p \rightarrow \Diamond_{=1} q), \quad (\dagger)$$

$$\Box(q \rightarrow \Diamond_{=1} p),$$

to assert that “every p -state is followed by a q -state *precisely* 1 time unit later” and “every q -state is preceded by a p -state 1 time unit earlier.” Observe that with an interval semantics, the latter (past) formula is not even needed. It follows our techniques can be used to show that MTL is undecidable in the analog-clock model:

Theorem 3.11 (Dense MTL) *If MTL is interpreted over a dense linear order, the satisfiability problem becomes Σ_1^1 -complete.*

Similar undecidability results can be obtained for the MTL-like branching-time logics considered by Alur, Courcoubetis, and Dill [7] and by Lewis [82], both of which use an interval semantics over the set of real numbers to model time.

There are two options to escape the predicament that is caused by this trade-off between the realistic modeling of time and the ability to verify timing properties:

1. We have adopted the *semantic* abstraction that, for every observation, we may record only a discrete approximation — the number of ticks of a digital clock — to the “real,” physical time. We have justified this decision by identifying the circumstances under which verification in the digital-clock model is both sound and complete; in Part 2 of this thesis we will develop concrete verification techniques under the assumption of a discrete time domain.
2. A recent result has opened the interesting alternative of a viable *syntactic* concession. We have shown that MTL without *singular* intervals (i.e., intervals of the form $[c, d]$ for $c = d$) as subscripts of the temporal operators can be decided in EXPSPACE even in the analog-clock model [9]. This syntactic restriction of MTL ensures that the time difference between two state changes can be enforced only with finite (yet arbitrary) precision; in particular, it rules out the specification of the *punctuality* requirement (\dagger) . Note that (\dagger) is not equivalent to the stronger demand

$$\Box(p \rightarrow \Box_{<1} \neg q) \wedge \Box(p \rightarrow \Diamond_{\leq 1} q)$$

that "every p -state is followed by a q -state no sooner and no later than after 1 time unit," which can be expressed without singular time intervals. We do not have the space to pursue this approach here and instead refer the interested reader to the original paper.

Choosing the operations on time

Having constrained ourselves to the discrete time domain of the digital-clock model, we need to choose the operations on time that are admitted by a real-time specification language. We have proved (case 2) that the addition of time variables causes undecidability. In fact, using our results and techniques, we can show the Π_1^1 -hardness of various real-time logics that have been proposed in the literature, such as the logics of Jahanian and Mok [63], Pnueli and Harel [110], Koymans [71], and [104], all of which include addition as a primitive operation on time. This list demonstrates vividly that it has not been understood until recently how expressive a theory of time may be added to reasoning about state sequences without sacrificing decidability. Harel, Lichtenstein, and Pnueli proved later the decidability of a fragment of what we called "real-time temporal logic" that permits the addition of time variables [54]. This decidable fragment XCTL (for "explicit-clock temporal logic") puts, however, such substantial restrictions on the use of time quantifiers that it is not closed under complementation.

The real-time logic RTL of Jahanian and Mok can be viewed as a two-sorted logic with multiple monotonic functions from the state sort to the time sort. Our results (case 4) imply that RTL is undecidable, even if we restrict its syntax to allow only the successor primitive over time (RTL allows addition over time).

Part II
Verification

Chapter 4

Finite-state Verification

We present algorithms for checking if a formula ϕ of TPTL or MTL holds over all runs of a finite-state timed transition system S . The algorithms use a technique called *model checking*: they construct a finite graph — a “tableau” — that represents all models of the formula ϕ and compare it to the finite graph that represents all possible runs of the system S . Thus model checking is applicable only to

1. logics that have the *finite-model* property; that is, if a formula is satisfiable, then it is satisfiable in a finite model. Although timed state sequences are infinite structures over an infinite time domain, in Chapter 3 we have shown that for any given formula ϕ of TPTL or MTL, the infinitary timing information in a model of ϕ can be encoded by finitely many time-difference and time-congruence propositions. Moreover, we have seen that the properties that are definable in TPTL and MTL are regular, which implies that every model of ϕ is “eventually periodic” and can be obtained by unrolling a finite structure. We will use both observations to represent all models of the formula ϕ by a tableau whose size is doubly exponential in the length of ϕ .
2. *finite-state* systems; that is, transition systems whose state graphs, which encode the consecution (i.e., next-state) relation, are finite. We will extend the notion of finite-state system to *timed* transition systems that can be represented by finite graphs.

Since both TPTL and MTL are undecidable in the analog-clock model (see Section 3.5), we restrict ourselves to the digital-clock model for model checking (i.e., $TIME = \mathbb{N}$ throughout

this chapter). A model-checking algorithm for a decidable fragment of analog MTL was presented elsewhere [9].

The first step of the model-checking process yields, as a by-product, decision procedures for TPTL and MTL: a formula ϕ is valid iff the tableau that represents all models of the negated formula $\neg\phi$ does not contain any timed state sequences. While being elementary, the tableau-based decision procedures for TPTL and MTL are still quite expensive; they run in doubly exponential time. We show that this cost is, however, intrinsic to real-time reasoning: any reasonably succinct and reasonably expressive extension of PTL is necessarily EXPSPACE-hard.

4.1 Deciding Timed Temporal Logic

We present a doubly-exponential-time, tableau-based decision procedure for TPTL and show that the decision problem for TPTL is EXPSPACE-complete. This result establishes, as we have promised in Chapter 3, that TPTL corresponds to an elementary fragment of the nonelementary first-order language \mathcal{L}_T . We then integrate the grammar operators of TETL into the tableau method. In Section 4.3, we will demonstrate how the tableau techniques can be applied to verify TETL-properties of finite-state timed transition systems.

4.1.1 Timed tableaux

First observe that to solve the validity problem for a formula, it suffices to check if its negation is satisfiable. Throughout this subsection, we are given a formula ϕ of TPTL and wish to determine iff ϕ is satisfiable. The tableau method searches systematically for a model of ϕ . It originated with the propositional calculus (consult [120]), was extended to modal logics (consult [39]), and was first applied to obtain a decision procedure for a logic of computation by Pratt [111]. We follow the standard presentation of the tableau-based decision procedure for PTL [18, 130] and begin by constructing the initial tableau for ϕ . Checking the satisfiability of ϕ can then be reduced to checking if the finite initial tableau for ϕ contains certain infinite paths. The tableau method for PTL is, in fact, subsumed by our procedure as the special case in which ϕ contains no timing constraints.

Preliminary assumptions

For the moment, we assume that

1. ϕ contains no absolute time references; that is, every term in ϕ contains a variable. Thus we may perform simple arithmetic manipulations so that all timing constraints in ϕ are of the form $x \leq y + c$ or $x + c \leq y$ or $x \equiv_d y + c$, for natural numbers $d > c \geq 0$.
2. ϕ contains only the temporal operators \bigcirc and \square ; that is, the first argument of every occurrence of the *until* operator \mathcal{U} in ϕ is *true*.

While neither of the two restrictions is essential, they simplify the exposition of the decision procedure greatly. Later we will show how to accommodate absolute time references, the *until* operator, and even general grammar operators. We also assume that ϕ is of the form $z. \phi'$; if necessary, this can be easily achieved by prefixing ϕ with any variable z that does not occur freely in ϕ .

Let us say that an infinite timed state sequence $\rho = (\sigma, T)$ is Δ -bounded, for a constant $\Delta \in \mathbb{N}$, iff

$$T_{i-1} \leq T_i \leq T_{i-1} + \Delta$$

for all $i \geq 0$; that is, the time increases from a state to its successor state by no more than Δ . Recall that we use the convention that $T_{-1} = 0$; it follows that the absolute time of the initial state of a Δ -bounded timed state sequence is at most Δ . To begin with, we restrict ourselves to Δ -bounded models for checking satisfiability. This case has finite-state character: the times that are associated with states can be modeled by finitely many (new) time-difference propositions $Prev_\delta$, for $0 \leq \delta \leq \Delta$, that represent, in the initial state, the initial time δ and, in all other states, the time increase δ from the predecessor state. Formally, we capture the (state and) time information in a timed state sequence $\rho = (\sigma, T)$ by the state sequence $\hat{\sigma}$ with

$$\hat{\sigma}_i = \sigma_i \cup \{Prev_{T_i - T_{i-1}}\}$$

for all $i \geq 0$. This reduction of timed state sequences to state sequences allows us to adopt the tableau techniques for PTL. Later, we show how we can find an appropriate constant Δ for the given formula ϕ .

Updating timing constraints

The key observation underlying the tableau method for PTL is that any formula can be split into two conditions: a non-temporal (“present”) requirement on the initial state and a temporal (“future”) requirement on the rest of the model (i.e., the successor state). For example, the eventuality $\Diamond\phi$ can be satisfied by either ϕ or $\bigcirc\Diamond\phi$ being true in the initial state. Since the number of conditions generated in this way is finite, checking for satisfiability is reducible to checking for satisfiability in a finite structure, the initial tableau.

The splitting of TPTL-formulas into a present and a future (next-state) condition demands more care; to obtain the requirement on the successor state, all timing constraints need to be updated appropriately to account for the time increase δ from the initial state to its successor. Consider, for example, the formula $x.\Diamond y.\psi(x, y)$, and recall that the free occurrences of x in ψ are references to the initial time. This eventuality can be satisfied either by having the initial state satisfy $y.\psi(y, y)$, with all free occurrences of x in ψ replaced by y , or by having the next state satisfy the updated eventuality “ $x.\Diamond y.\psi(x - \delta, y)$.” For $\delta > 0$, a naive replacement of x by $x - \delta$ would, however, successively generate infinitely many new conditions. Fortunately, the *monotonicity* of time can be exploited to keep the tableau finite; the observation that y is always instantiated, in the “future,” to a value greater than or equal to the initial time x , allows us to simplify timing assertions of the form $x \leq y + (c + 1)$ and $y + (c + 1) \leq x$ to *true* and *false*, respectively.

We define, therefore, the TPTL-formula $x.\psi(x)^\delta$ that results from updating all references in ψ to the initial time x by the time difference δ . For instance, if $x.\psi$ is the formula

$$x.\Box y.(p \rightarrow y.y \leq x + 5),$$

then $x.\psi^1$, $x.\psi^5$, and $x.\psi^6$ are the following formulas:

$$x.\Box y.(p \rightarrow y.y \leq x + 4),$$

$$x.\Box y.(p \rightarrow y.y \leq x),$$

$$x.\Box y.(p \rightarrow \text{false}).$$

In general, given a TPTL-formula $x.\psi$ and $\delta \in \mathbb{N}$, the TPTL-formula $x.\psi^\delta$ is defined inductively as follows:

- $x.\psi^0$ equals $x.\psi$.

- $x.\psi^{\delta+1}$ results from $x.\psi^{\delta}$ by replacing every term of the form $x + (c + 1)$ with $x + c$, and every subformula of the form $x \leq y + c$, $y + c \leq x$, and $x \equiv_d y + c$ with *true*, *false*, and $x \equiv_d y + ((c + 1) \bmod d)$, respectively, provided that the occurrence of x in the specified terms and formulas is free in ψ^{δ} .

The following lemma confirms that this transformation has the intended effect and updates all time references correctly; that is, the formula $x.\psi^{\delta}$ expresses the condition " $x.\psi(x - \delta)$."

Lemma 4.1 (Time step) *Let $\rho = (\sigma, T)$ be a timed state sequence and $\delta \in \mathbb{N}$ such that $\delta \leq T_0$. Then $\rho \models x.\psi^{\delta}$ iff $\rho \models_{[x:=T_0-\delta]} \psi$ for every formula $x.\psi$ of TPTL.*

Proof of Lemma 4.1 The proof proceeds by a straightforward induction on the structure of ψ . ■

Closure of a TPTL-formula

We collect all conditions that may arise by recursively splitting a formula ϕ into its present and future parts in the closure of ϕ . It suffices to define the closure only for formulas whose outermost symbol is a freeze quantifier. The *closure set* $Closure(z.\phi')$ of the TPTL-formula $z.\phi'$ is the smallest set of formulas containing $z.\phi'$ that is closed under the following operation *Sub*:

$$\begin{aligned} Sub(z.(\psi_1 \rightarrow \psi_2)) &= \{z.\psi_1, z.\psi_2\}, \\ Sub(z.\bigcirc \psi) &= \{z.\psi^{\delta} \mid \delta \geq 0\}, \\ Sub(z.\square \psi) &= \{z.\psi, z.\bigcirc \square \psi\}, \\ Sub(z.x.\psi) &= \{z.\psi[x := z]\}, \end{aligned}$$

where the TPTL-formula $\psi[x := z]$ results from ψ by replacing all free occurrences of x with z . Note that all formulas in a closure set are of the form $z.\psi$.

We say that a constant $c > 0$ *occurs* in the TPTL-formula ϕ iff ϕ contains a subformula of the form $x \leq y + (c - 1)$ or $x + (c - 1) \leq y$, or ϕ contains the predicate symbol \equiv_c . Let C be the largest constant that occurs in the formula ϕ . The closure set of ϕ is finite, because $x.\psi^{\delta}$ is $z.\psi^C$ for all formulas $z.\psi$ in $Closure(\phi)$ and all $\delta \geq C$. The size of the closure set of ϕ depends on both the structure of ϕ and the constants that occur in ϕ .

Lemma 4.2 (Size of closure) *Let $n - 1$ be the number of boolean, temporal, and freeze operators in the formula ϕ of TPTL, and let k be the product of all constants that occur in ϕ . Then $|\text{Closure}(\phi)| \leq 2nk$.*

Proof of Lemma 4.2 Given a TPTL-formula $z.\phi'$, we define, by induction on the structure of ϕ' , the set $D_{\phi'}$ of formulas that contains ϕ' and is closed under updating of timing constraints; the set $C_{\phi'}$ is, in addition, closed under subformulas:

$$\begin{aligned} C_p &= D_p = \{p\}, \\ C_{x \leq y+c} &= D_{x \leq y+c} = \{x \leq y+c, x \leq y+(c-1), \dots, x \leq y\}, \\ C_{x \equiv_d y+c} &= D_{x \equiv_d y+c} = \{x \equiv_d y+(d-1), x \equiv_d y+(d-2), \dots, x \equiv_d y\}, \\ C_{\phi_1 \rightarrow \phi_2} &= C_{\phi_1} \cup C_{\phi_2} \cup D_{\phi_1 \rightarrow \phi_2}, \quad D_{\phi_1 \rightarrow \phi_2} = D_{\phi_1} \rightarrow D_{\phi_2}, \\ C_{\bigcirc \phi} &= C_{\phi} \cup D_{\bigcirc \phi}, \quad D_{\bigcirc \phi} = \bigcirc D_{\phi}, \\ C_{\square \phi} &= C_{\phi} \cup D_{\square \phi} \cup D_{\bigcirc \square \phi}, \quad D_{\square \phi} = \square D_{\phi}, \\ C_{x.\phi} &= C_{\phi[x:=z]} \cup D_{x.\phi}, \quad D_{x.\phi} = x.D_{\phi}. \end{aligned}$$

Here $x.E = \{x.\phi \mid \phi \in E\}$ for any set E of formulas; the other operators are applied to sets in an analogous fashion. The case of formulas of the form $x+c \leq y$ is treated similarly to the timing constraints $x \leq y+c$. Furthermore, let $E^* = E \cup \{\text{true}, \text{false}\}$.

Observe that $D_{\phi} \subseteq C_{\phi}$. It is straightforward to show, by induction on the structure of ϕ , that

- (1) $\phi' \in D_{\phi'}$ and, hence, $\phi' \in C_{\phi'}$.
- (2) For all $\delta \geq 0$, $z.\phi'^{\delta} \in z.D_{\phi'}^*$, and, therefore, $z.\phi'^{\delta} \in z.C_{\phi'}^*$.
- (3) $z.C_{\phi'}^*$ is closed under *Sub* (use (2)).

From (1) and (3) it follows that $\text{Closure}(z.\phi') \subseteq z.C_{\phi'}^*$. Thus, it suffices to show that $|D_{\phi'}| \leq k$ and $|C_{\phi'}| \leq 2nk$, which may again be done by induction on the structure of ϕ' . ■

Initial tableau of a TPTL-formula

Tableaux for TPTL are finite, directed state graphs (Kripke structures) with local and global consistency constraints on all vertices. Unlike the states of a timed state sequence, the vertices of a tableau contain not only propositions, but are annotated with arbitrary

formulas of TPTL. Since the set of propositions that labels a vertex determines a state, we shall informally refer to the vertices of a tableau themselves as "states," provided this usage does not give rise to ambiguities. The set of TPTL-formulas that labels a vertex of a tableau is closed under "subformulas" and its members express conditions on the annotated state and its successor states. Every vertex contains, in addition, a time-difference proposition $Prev_\delta$, where $0 \leq \delta \leq \Delta$, that denotes the time increase from the predecessor states.

Formally, we define the vertices of a tableau for ϕ as the maximally consistent subsets of the finite universe

$$Closure^*(\phi) = Closure(\phi) \cup \{Prev_\delta \mid 0 \leq \delta \leq \Delta\}$$

of TPTL-formulas. First, we put together all requirements for local consistency. A subset Φ of $Closure^*(\phi)$ is (maximally) *consistent* iff it satisfies the following conditions, where all formulas range only over the finite set $Closure^*(\phi)$:

- $Prev_\delta$ is in Φ for precisely one $0 \leq \delta \leq \Delta$; this $\delta \in \mathbb{N}$ is referred to as δ_Φ .
- $z.(z \sim z + c)$ is in Φ iff $0 \sim c$ holds in the natural numbers (for \sim being one of $\leq, \geq,$ and \equiv_d).
- $z.false$ is not in Φ .
- $z.(\psi_1 \rightarrow \psi_2)$ is in Φ iff either $z.\psi_1$ is not in Φ or $z.\psi_2$ is in Φ .
- $z.\Box\psi$ is in Φ iff both $z.\psi$ and $z.\bigcirc\Box\psi$ are in Φ .
- $z.x.\psi$ is in Φ iff $z.\psi[x := z]$ is in Φ .

Now we are ready to define the initial tableau of a formula in a way that ensures the global consistency of both temporal and real-time constraints. The *initial tableau* $T(\phi)$ for the TPTL-formula ϕ is a directed graph whose vertices are the consistent subsets of $Closure^*(\phi)$, and which contains an edge from Φ to Ψ iff, for all formulas $z.\bigcirc\psi$ in $Closure^*(\phi)$,

$$z.\bigcirc\psi \in \Phi \text{ iff } z.\phi^{\delta_\Phi} \in \Psi.$$

The significance of the (finite) initial tableau $T(\phi)$ for the formula ϕ is that every model of ϕ corresponds to an infinite path through $T(\phi)$ along which all eventualities are satisfied

in time, and vice versa. This implies a finite-model property for TPTL, in the sense that every satisfiable TPTL-formula ϕ is satisfiable by a model whose state part, extended by the time-difference propositions $Prev_\delta$, does not only consist of no more than finitely many distinct states, but is eventually periodic. To be precise, we say that an infinite path

$$\Phi : \Phi_0 \rightarrow \Phi_1 \rightarrow \Phi_2 \rightarrow \dots$$

through a tableau is a ϕ -path iff it satisfies the following three conditions:

Initiality $\phi \in \Phi_0$.

Fairness All eventualities are satisfied along Φ in time or, equivalently, all missing invariances are violated along Φ in time; that is, for all $z. \Box\psi \in \text{Closure}^-(\phi)$ and $i \geq 0$,

$$z. \Box\psi \notin \Phi_i \text{ implies } z. \psi^{\delta} \notin \Phi_j \text{ for some } j \geq i \text{ with } \delta = \sum_{i < k \leq j} \delta_{\Phi_k}^-.$$

Progress $\delta_{\Phi_i}^- > 0$ for infinitely many $i \geq 0$.

We can characterize the length of ϕ -paths in the initial tableau for ϕ by reducing every ϕ -path to a special form. The following lemma shows that every ϕ -path is eventually periodic and bounds the length of the period. This will prove to be important to obtain an upper bound on the complexity of TPTL.

Lemma 4.3 (Length of ϕ -paths) *Suppose that the initial tableau $T(\phi)$ for the formula ϕ of TPTL consists of m vertices. If $T(\phi)$ contains a ϕ -path, then it contains a ϕ -path of the form*

$$\Phi_0 \rightarrow \Phi_1 \rightarrow \dots \rightarrow \Phi_i \rightarrow (\Phi_{i+1} \rightarrow \dots \rightarrow \Phi_l)^{\omega}$$

for $l \leq (2n + 1)m$, where n is the number of temporal operators in ϕ .

Proof of Lemma 4.3 Consider the infinite ϕ -path $\Phi = \Phi_0\Phi_1\dots$, and choose i to be the smallest j such that Φ_j occurs infinitely often in Φ . Now Φ_i lacks at most n invariances $z. \Box\psi_k$, each one of which is violated by some vertex Ψ_k of $\Phi_i\Phi_{i+1}\dots$. Let $\Phi^0 = \Phi_0\dots\Phi_i$, $\Phi^{2k-1} = \Phi_i\dots\Psi_k$, and $\Phi^{2k} = \Psi_k\dots\Phi_i$, for all $1 \leq k \leq n$, be finite segments of Φ that contain no other (i.e., inner) occurrences of Φ_i . Delete any loops in every Φ^j , thus obtaining

the finite sequences $\hat{\Phi}^j$, for $0 \leq j \leq 2n$, each of length at most $m + 1$. It is not hard to see that the result of deleting duplicated vertices (i.e., Φ_i) from

$$\hat{\Phi}^0(\hat{\Phi}^1\hat{\Phi}^2 \dots \hat{\Phi}^{2n})^\omega$$

is a ϕ -path of the desired form. ■

Tableau decision procedure

The following main lemma suggests a decision procedure for TPTL: to determine if the TPTL-formula ϕ is satisfiable, construct the initial tableau $T(\phi)$ and check if it contains any ϕ -paths.

Lemma 4.4 (Initial tableau for TPTL)

- (1) [Correctness] *If the initial tableau $T(\phi)$ for the formula ϕ of TPTL contains a ϕ -path, then ϕ is satisfiable.*
 (2) [Completeness] *If ϕ has a Δ -bounded model, then $T(\phi)$ contains a ϕ -path.*

Proof of Lemma 4.4 The proof makes essential use of both directions of the *time-step* lemma, Lemma 4.1.

(1) Given a ϕ -path $\Phi = \Phi_0\Phi_1 \dots$ through the initial tableau $T(\phi)$, define the timed state sequence $\rho = (\sigma, \tau)$ such that, for all $i \geq 0$, $p \in \sigma_i$ iff $z.p \in \Phi_i$, and $\tau_i = \tau_{i-1} + \delta_{\Phi_i}$. Note that the time sequence τ satisfies the *progress* condition because Φ does. We show, by induction on the structure of ϕ , that $\psi \in \Phi_i$ iff $\rho^i \models \psi$ for all $i \geq 0$ and $\psi \in \text{Closure}^*(\phi)$. Since $\phi \in \Phi_0$, it follows that ρ is a model of ϕ .

For a proposition $z.p \in \text{Closure}^*(\phi)$, we have $z.p \in \Phi_i$ iff $p \in \sigma_i$ iff $\rho^i \models z.p$. Let \sim be one of \leq , \geq , \equiv_d , or its negation. By the consistency of Φ_i , $z.(z \sim z + c) \in \Phi_i$ iff $0 \sim c$ iff $\rho^i \models z.(z \sim z + c)$. This completes the base cases.

By the consistency of Φ_i , $z.(\psi_1 \rightarrow \psi_2) \in \Phi_i$ iff either $z.\psi_1 \notin \Phi_i$ or $z.\psi_2 \in \Phi_i$. By the induction hypothesis, this is the case iff either $\rho^i \not\models z.\psi_1$ or $\rho^i \models z.\psi_2$; that is, iff $\rho^i \models z.(\psi_1 \rightarrow \psi_2)$.

Now assume that $z. \bigcirc \psi \in \text{Closure}^*(\phi)$ and let $\delta = \delta_{\Phi_{i+1}}$; that is, $\tau_{i+1} = \tau_i + \delta$. Then $z. \bigcirc \psi \in \Phi_i$ iff $z.\psi^\delta \in \Phi_{i+1}$. By the induction hypothesis, this is the case iff $\rho^{i+1} \models z.\psi^\delta$. By Lemma 4.1, this is the case iff $\rho^{i+1} \models_{[z:=\tau_i]} \psi$; that is, iff $\rho^i \models z. \bigcirc \psi$.

For the case that $z. \Box \psi \in \text{Closure}^*(\phi)$, we first prove that $z. \Box \psi \in \Phi_i$ iff $z. \psi^{\top_j - \top_i} \in \Phi_j$ for all $j \geq i$. Let $\delta_j = \top_j - \top_i$ and note that $\delta = \sum_{i < k \leq j} \delta_{\Phi_k}$ by our choice of \top .

We use induction on j to show that $z. \Box \psi \in \Phi_i$ implies $z. \Box \psi^{\delta_j} \in \Phi_j$ for all $j \geq i$. Suppose that $z. \Box \psi^{\delta_j} \in \Phi_j$ for an arbitrary $j \geq i$. By the consistency of Φ_j , also $z. \bigcirc \Box \psi^{\delta_j} \in \Phi_j$ and therefore $z. \Box \psi^{\delta_{j+1}} \in \Phi_{j+1}$. Invoking again the consistency of Φ_j , we conclude that $z. \psi^{\delta_j} \in \Phi_j$ for all $j \geq i$.

On the other hand, suppose that $z. \Box \psi \notin \Phi_i$. Since Φ is a ϕ -path, there is some $j \geq i$ such that $z. \psi^{\delta_j} \notin \Phi_j$.

By the induction hypothesis, it follows that $z. \Box \psi \in \Phi_i$ iff $\rho^j \models z. \psi^{\delta_j}$ for all $j \geq i$. By Lemma 4.1, this is the case iff $\rho^j \models_{[z := \top_i]} \psi$ for all $j \geq i$; that is, iff $\rho^i \models z. \Box \psi$.

Finally, consider the case that $z. x. \psi \in \text{Closure}^*(\phi)$. In this case, $z. x. \psi \in \Phi_i$ iff $z. \psi[x := z] \in \Phi_i$. By the induction hypothesis, this is the case iff $\rho^i \models z. \psi[x := z]$; that is, iff $\rho^i \models z. x. \psi$.

(2) Let $\rho = (\sigma, \top)$ be a Δ -bounded model of ϕ . The subsets Φ_i , for $i \geq 0$, of $\text{Closure}^*(\phi)$ are defined as follows: $\text{Prev}_{\top_i - \top_{i-1}} \in \Phi_i$, and $\psi \in \Phi_i$ iff $\rho^i \models \psi$ for all $\psi \in \text{Closure}(\phi)$. We show that $\Phi = \Phi_0 \Phi_1 \dots$ is a ϕ -path through the initial tableau $T(\phi)$.

By inspecting the consistency rules, it is evident that every Φ_i is (maximally) consistent. To prove that Φ is an infinite path through $T(\phi)$, we also have to show that there is an edge from Φ_i to Φ_{i+1} for all $i \geq 0$. Suppose that $z. \bigcirc \psi \in \text{Closure}^*(\phi)$ and let $\delta = \top_{i+1} - \top_i$. Then $z. \bigcirc \psi \in \Phi_i$ iff $\rho^i \models z. \bigcirc \psi$ iff $\rho^{i+1} \models_{[z := \top_i]} \psi$. By Lemma 4.1, this is the case iff $\rho^{i+1} \models z. \psi^\delta$; that is, iff $z. \psi^\delta \in \Phi_{i+1}$. Since also $\text{Prev}_\delta \in \Phi_{i+1}$, the initial tableau for ϕ contains an edge from Φ_i to Φ_{i+1} .

We now show that the infinite path Φ is indeed a ϕ -path. It satisfies the *progress* condition because the time sequence \top does. To see that $\phi \in \Phi_0$, observe that ρ is a model of ϕ . It remains to be established that all eventualities in Φ are satisfied in time. Suppose that $z. \Box \psi \in \text{Closure}^*(\phi)$ and $z. \Box \psi \notin \Phi_i$; that is, $\rho^i \models z. \Diamond \neg \psi$ and therefore $\rho^j \models_{[z := \top_i]} \neg \psi$ for some $j \geq i$. Let $\delta = \top_j - \top_i$; thus $\delta = \sum_{i < k \leq j} \delta_{\Phi_k}$. Then $\rho^j \not\models z. \psi^\delta$ by Lemma 4.1, which implies that $z. \psi^\delta \notin \Phi_j$. \blacksquare

The usual techniques for checking if a tableau contains an infinite path along which all eventualities are satisfied, can be straightforwardly adopted to check if the initial tableau

for the formula ϕ contains a ϕ -path: first mark all eventualities that are trivially satisfied, and then repeatedly mark the eventualities that are satisfied in successor states. The state graph that remains if all vertices that are not on a ϕ -path are deleted from $T(\phi)$ is called the *final tableau* for the formula ϕ . Hence we have shown that a TPTL-formula ϕ has a Δ -bounded model iff the final tableau for ϕ is not empty.

The procedure for finding the final tableau is polynomial in the size of the initial tableau, which contains $O(\Delta \cdot 2^{nk})$ vertices, each of size $O(nk)$, where $n-1$ is the number of operators in ϕ and k is the product of all constants that occur in ϕ . Thus, provided that Δ is dominated by 2^{nk} , the initial $T(\phi)$ can be constructed and checked for ϕ -paths in deterministic time exponential in nk . We show next that Δ can indeed be bounded by k .

Bounding the time step-width

Given a TPTL-formula ϕ , we finally determine the bound Δ on the time increase between two successive states such that the satisfiability of ϕ is not affected; that is, we choose the constant $\Delta \in \mathbb{N}$ such that ϕ is satisfiable iff it has a Δ -bounded model. Let c be the largest constant in ϕ that occurs in a subformula of the form $x \leq y + (c-1)$ or $x + (c-1) \leq y$, and let $\equiv_{c_1}, \dots, \equiv_{c_m}$ all the congruence predicates that occur in ϕ . If the time increase δ between two states is greater than or equal to c , it obviously suffices to know the residues of δ modulo c_1, \dots, c_m in order to update, in a tableau, all timing constraints correctly. Indeed, for checking the satisfiability of ϕ , the arbitrary step-width δ can be bounded by taking the smallest representative for each of the finitely many congruence classes.

Lemma 4.5 (Bounded time increase) *If the formula ϕ of TPTL is satisfiable, then $\rho \models \phi$ for some $\rho = (\sigma, T)$ with $T_i \leq T_{i-1} + k$ for all $i \geq 0$, where k is the product of all constants that occur in ϕ .*

Proof of Lemma 4.5 We can, in fact, derive the tighter bound $c + k' \leq k$, for the least common multiple k' of all c_i , $1 \leq i \leq m$. Given a model $\rho = (\sigma, T)$ of ϕ , let the time sequence T' be such that, for all $i \geq 0$, $T_i = T'_{i-1} + (T_i - T_{i-1})$ if $T_{i+1} - T_i < c$; else choose T'_i to be the smallest $\delta \geq T'_i + c$ with $\delta \equiv_{k'} T_i$. It is easy to see that $\rho' = (\sigma, T')$ is also a model of ϕ . ■

Combining this result with the tableau method developed above, we arrive at the conclusion that the satisfiability of the TPTL-formula ϕ is decidable in deterministic time

exponential in nk . Moreover, Lemma 4.3 implies that every satisfiable formula ϕ is satisfiable in a model whose size is, in the sense mentioned above, exponential in nk . Remember that we have restricted ϕ to contain no *until* operators and no absolute time references. We now show that both assumptions can be relaxed.

Until operators

First, let us address TPTL-formulas that include *until* operators. We take the *closure* of a formula ϕ with *until* operators to be closed under the operation

$$\text{Sub}(z.(\psi_1 \mathcal{U} \psi_2)) = \{z.\psi_1, z.\psi_2, z. \bigcirc (\psi_1 \mathcal{U} \psi_2)\}$$

and add the following condition on the local *consistency* of a set $\Phi \subseteq \text{Closure}^*(\phi)$ of formulas:

$z.(\psi_1 \mathcal{U} \psi_2)$ is in Φ iff
either $z.\psi_2$ is in Φ , or both $z.\psi_1$ and $z. \bigcirc (\psi_1 \mathcal{U} \psi_2)$ are in Φ .

Finally, the *fairness* requirement on a ϕ -path $\Phi_0 \Phi_1 \Phi_2 \dots$ is generalized to

$$z.(\psi_1 \mathcal{U} \psi_2) \in \Phi_i \text{ implies } z.\psi_2^{\delta} \in \Phi_j \text{ for some } j \geq i \text{ with } \delta = \sum_{i < k \leq j} \delta_{\Phi_k}^-$$

for all $i \geq 0$. It is not hard to see that the Lemmas 4.2, 4.3, 4.4, and 4.5 allow the addition of *until* operators in this way.

Absolute time references

Secondly, let us accommodate absolute time references. Instead of generalizing the tableau method to constant terms, which contain no variable, we can use a simple observation. Suppose that we test the TPTL-formula ϕ for satisfiability. Let x be a variable that does not occur in ϕ and replace every variable-free term c in ϕ by the term $x + c$, thus obtaining the new formula ϕ^R (which may contain free occurrences of x). The following lemma allows us to reduce the satisfiability problem for ϕ to the satisfiability problem for the formula $x. \bigcirc \phi^R$, which contains no absolute time references.

Lemma 4.6 (Absolute time references) *A formula ϕ of TPTL is satisfiable iff the formula $x. \bigcirc \phi^R$ is satisfiable, where x does not occur in ϕ .*

Proof of Lemma 4.6 (1) Let $\rho = (\sigma, T)$ be a timed state sequence. We define the timed state sequence $\rho' = (\sigma', T')$ such that $T'_0 = 0$, and $\sigma'_{i+1} = \sigma_i$ and $T'_{i+1} = T_i$ for all $i \geq 0$. Clearly, if $\rho \models \phi$, then $\rho' \models \phi$. \square

(2) Let $\rho = (\sigma, T)$ be a timed state sequence. We define the timed state sequence $\rho' = (\sigma', T')$ such that $\sigma'_i = \sigma_{i+1}$ and $T'_i = T_{i+1} - T_0$ for all $i \geq 0$. Clearly, if $\rho \models \phi$, then $\rho' \models \phi$. \blacksquare

Note that the transformation from ϕ to ϕ^R does not increase the number of operators in ϕ nor the product of all constants that occur in ϕ . The following theorem summarizes our results about the tableau method.

Theorem 4.1 (Deciding TPTL) *The validity problem for a formula ϕ of TPTL can be decided in deterministic time exponential in nk , where $n - 1$ is the number of boolean, temporal, and freeze operators in ϕ , and k is the product of all constants that occur in ϕ .*

Note that the length ℓ of any TPTL-formula ϕ , whose constants are presented in a logarithmic (e.g., binary) encoding, is within constant factors of $n + \log k$. Thus we have a decision procedure for TPTL that is doubly exponential in ℓ (although only singly exponential in n , the "untimed" part and, therefore, singly exponential for PTL). The algorithm we have outlined can, of course, be improved in many ways. In particular, we may avoid the construction of the entire initial tableau by starting with the initial state, which contains ϕ , and successively adding new states only when needed [130]. This stepwise procedure, however, does not lower the doubly exponential deterministic-time bound; in fact, as we will show in the following subsection, the decision problem for TPTL is EXPSPACE-hard.

We also remark that while the *monotonicity* condition on timed state sequences is essential for the tableau method to work, the *progress* condition on timed state sequences (and ϕ -paths) can be omitted.

4.1.2 Complexity of timed temporal logic

We prove the following theorem, which establishes TPTL as being exponentially harder to decide than its untimed base PTL, which has a PSPACE-complete decision problem [119].

Theorem 4.2 (Complexity of TPTL) *The validity problem for TPTL is EXPSPACE-complete (with respect to polynomial-time reduction).*

Proof of Theorem 4.2 The proof proceeds in two parts; we first show that TPTL is in EXPSPACE, and then that it is EXPSPACE-hard. The first part follows the argument that PTL is in PSPACE, which builds on a nondeterministic version of the tableau-based decision procedure [130]; the hardness part is patterned after the proof by Hopcroft and Ullman that the universality problem of regular expressions with exponentiation is EXPSPACE-hard [62].

[EXPSPACE] It suffices to show that the complementary problem of checking the satisfiability of a TPTL-formula is in nondeterministic EXPSPACE and, hence, by Savitch's theorem, in (deterministic) EXPSPACE.

In particular, it can be checked in nondeterministic singly exponential space if the initial tableau $T(\phi)$ contains a ϕ -path of the form stated in Lemma 4.3. In trying to construct such a ϕ -path nondeterministically, at each stage only the current vertex, the "loop-back" vertex, and a vertex counter have to be retained in order to construct a successor vertex, loop back, or, if the vertex counter exceeds the maximal length of the loop, fail. Since both the size of each vertex and the length of the loop have, by Lemma 4.2 and Lemma 4.3, respectively, (singly) exponential representations in the length of ϕ , it follows that this nondeterministic procedure requires only exponential space.

[EXPSPACE-hardness] Consider a deterministic 2^n -space-bounded Turing machine M . For each input X of length n , we construct a TPTL-formula ϕ_X of length $O(n \cdot \log n)$ that is valid iff M accepts X . By a standard complexity-theoretic argument, using the hierarchy theorem for space, it follows that there is a constant $c > 0$ such that every Turing machine that solves the validity problem for TPTL-formulas ϕ of length ℓ takes space $S(\ell) \geq 2^{c\ell/\log \ell}$ infinitely often. Thus it suffices to construct, given the initial tape contents X ,

1. a sufficiently succinct formula ϕ_X that describes the (unique) computation of M on X as an infinite sequence of propositions, and
2. a sufficiently succinct formula ϕ_{ACCEPT} that characterizes the computation of M on X as accepting.

Then the implication

$$\phi_X \rightarrow \phi_{ACCEPT}$$

is valid iff the machine M accepts the input X .

We use a proposition p_i and a proposition q_j for every tape symbol i and machine state j of M , respectively. In particular, p_0 and q_0 correspond to the special tape symbol "blank" and the initial state of M . We use the following abbreviations for formulas:

$$\begin{aligned}\hat{p}_i &: p_i \wedge \bigwedge_{i' \neq i} \neg p_{i'} \wedge \bigwedge \neg q_j, \\ \tau_{i,j} &: p_i \wedge q_j \wedge \bigwedge_{i' \neq i} \neg p_{i'} \wedge \bigwedge_{j' \neq j} \neg q_{j'}, \\ s &: \bigwedge \neg p_{i'} \wedge \bigwedge \neg q_j.\end{aligned}$$

We represent configurations of M by \hat{p} -state sequences of length 2^n that are separated by s -states; the position of the read-write head is marked by an τ -state. The computation of M on X is completely determined by the following two conditions:

- (1) it starts with the initial configuration, and
- (2) every configuration follows from the previous one by a move of M .

Both conditions can be expressed in TPTL. Take ϕ_X to consist of $\Box x. \bigcirc y. y = x + 1$, forcing time to resemble a state counter, and the following two conjuncts, which correspond to the requirements (1) and (2), respectively:

$$\begin{aligned}\phi_{INITIAL}: & s \wedge \bigcirc \tau_{X_1,0} \wedge x. \bigwedge_{2 \leq i \leq n} \Box y. (y = x + i \rightarrow \hat{p}_{X_i}) \wedge \\ & x. \Box y. (x + n < y \leq x + 2^n \rightarrow \hat{p}_0),\end{aligned}$$

$$\begin{aligned}\phi_{MOVE}: & \Box x. (s \rightarrow \bigcirc y. (y = x + 2^n + 1 \wedge s)) \wedge \\ & \bigwedge_{P,Q,R} \Box x. (P \wedge \bigcirc Q \wedge \bigcirc \bigcirc R \rightarrow \bigcirc y. (y = x + 2^n + 2 \wedge f_M(P, Q, R))).\end{aligned}$$

Here P , Q , and R each range over the propositions \hat{p}_i , $\tau_{i,j}$, and s , and $f_M(P, Q, R)$ refers to the transition function of M . For instance, if M writes, in state j on input i' , the symbol k onto the tape, moves to the right, and enters state j' , then $f_M(\hat{p}_{i'}, \tau_{i',j}, \hat{p}_{i''}) = \hat{p}_k$ and $f_M(\tau_{i',j}, \hat{p}_{i''}, \hat{p}_{i'''}) = \tau_{i''',j'}$.

The computation of M on X is accepting iff it contains the accepting state F , which is expressible in TPTL by the formula

$$\phi_{ACCEPT}: \bigcirc \bigvee_i \tau_{i,F}.$$

The lengths of $\phi_{INITIAL}$, ϕ_{MOVE} , and ϕ_{ACCEPT} are $O(n \cdot \log n)$, $O(n)$, and $O(1)$, respectively (recall that constants are represented in binary), thus implying the desired $O(n \cdot \log n)$ -bound for ϕ_X . ■

4.1.3 Timed extended temporal logic

By putting together the tableau methods for ETL [130] and TPTL, we develop a doubly-exponential-time decision procedure for TETL. We are given a formula ϕ of TETL and wish to determine if ϕ is satisfiable. For the sake of keeping the presentation simple, we assume that all grammar operators in ϕ correspond to productions of the form

$$\mathcal{G}(a_1, \dots, a_m) \longrightarrow a_{i_1} \mid a_{i_2} \mathcal{G}'(a_{j_1}, \dots, a_{j_n}).$$

We also assume that ϕ is of the form $z. \phi'$, as usual.

First, we observe that the Lemmas 4.1, 4.5, and 4.6 hold for TETL as they do for TPTL. It follows that

Lemma 4.6 Absolute time references can be handled as in TPTL.

Lemma 4.5 For checking the satisfiability of ϕ , we may restrict ourselves to k -bounded timed state sequences, where k is the product of all constants that occur in ϕ . All time information can, therefore, again be modeled by k time-difference propositions, $Prev_\delta$ for all $0 \leq \delta \leq k$.

Lemma 4.1 The TETL-formula $z. \psi^\delta$ updates all references in $z. \psi$ to the initial time z by the time difference δ .

Next, we define the closure for grammar operators. The *closure set* $Closure(z. \phi')$ of the TETL-formula $z. \phi'$ is the smallest set of formulas containing $z. \phi'$ that is closed under the operation Sub , whose effect on grammar operators is defined by the clause

$$Sub(z. \mathcal{G}(\psi_1, \dots, \psi_m)) = \{z. \psi_{i_1}, z. \psi_{i_2}, z. \bigcirc \mathcal{G}'(\psi_{j_1}, \dots, \psi_{j_n})\}$$

To determine the number of symbols in a TETL-formula, we count grammar operator to contribute the number of nonterminal symbols in the corresponding grammar. For this purpose, we say that a nonterminal symbol occurs in a TETL-formula ϕ iff it occurs in a grammar that is denoted by a grammar operator in ϕ . Let $n - 1$ be the number of boolean connectives, freeze quantifiers, and nonterminal symbols in ϕ , and let k be the product of all constants that occur in ϕ . By induction on the structure of ϕ , we can show the analog to Lemma 4.2 for TETL, namely that

$$|Closure(\phi)| \leq 2nk$$

for every TETL-formula ϕ .

The universe $Closure^*(\phi)$ and the edge relation of the initial tableau $T(\phi)$ for the TETL-formula ϕ is defined as in the case of TPTL; a subset Φ of $Closure^*(\phi)$ is (maximally) consistent iff it satisfies, in addition, the following condition on grammar operators:

$z.G(\psi_1, \dots, \psi_m)$ is in Φ iff
either $z.\psi_{i_1}$ is in Φ , or both $z.\psi_{i_2}$ and $z.O'(\psi_{j_1}, \dots, \psi_{j_n})$ are in Φ .

We show that every model of the TETL-formula ϕ corresponds to an infinite path through the initial tableau $T(\phi)$ for ϕ along which all eventualities are satisfied in time, and vice versa. An eventuality " $\neg z.G(\psi_1, \dots, \psi_m)$ " is called *fulfilled* along the finite path $\Phi_0\Phi_1 \dots \Phi_k$ through a tableau iff either $z.\psi_{i_2} \notin \Phi_0$, or $k \geq 1$ and $\neg z.G'(\psi_{j_1}, \dots, \psi_{j_n})^{\Phi_1}$ is fulfilled along the path $\Phi_1\Phi_2 \dots \Phi_k$. An infinite path

$$\Phi: \Phi_0 \rightarrow \Phi_1 \rightarrow \Phi_2 \rightarrow \dots$$

through a tableau is a ϕ -path iff it satisfies, in addition to the *initiality* and *progress* conditions, the following *fairness* requirement: for all formulas $z.G(\psi_1, \dots, \psi_m)$ in $Closure^*(\phi)$ and all $i \geq 0$,

if $z.G(\psi_1, \dots, \psi_m) \notin \Phi_i$, then the eventuality $\neg z.G(\psi_1, \dots, \psi_m)$ is fulfilled along some finite segment $\Phi_i\Phi_{i+1} \dots \Phi_k$, with $k \geq i$, of Φ .

By combining the corresponding arguments for ETL and TPTL, we can prove for TETL both Lemma 4.3, which bounds the size of models, and Lemma 4.4, which establishes the correctness of the tableau construction. Thus we have a decision procedure for TETL:

Lemma 4.7 (Initial tableau for TETL) *The formula ϕ of TETL is satisfiable iff the initial tableau $T(\phi)$ for ϕ contains a ϕ -path.*

Since the initial tableau contains $O(k \cdot 2^{nk})$ vertices, each of size $O(nk)$, the initial tableau $T(\phi)$ can be constructed and checked for ϕ -paths in deterministic time exponential in $O(nk)$. It follows that

Theorem 4.3 (Deciding TETL) *The validity problem for a formula ϕ of TETL can be decided in deterministic time exponential in $O(nk)$, where $n-1$ is the number of connectives,*

quantifiers, and nonterminal symbols in ϕ , and k is the product of all constants that occur in ϕ .

Recall that a nonterminal symbol is counted to "occur" in a TETL-formula ϕ even if the corresponding grammar operator does not literally occur in ϕ , but the nonterminal symbol is contained in a grammar whose starting nonterminal symbol occurs in ϕ . Also note that the length of a TETL-formula whose constants are represented in binary, is $O(n + \log k)$. Consequently, the tableau-based decision procedure for TETL is, as in the case of TPTL, doubly exponential in the length of the input formula (although only singly exponential in n and, therefore, singly exponential for ETL).

Complexity of TETL

The following theorem shows that TETL is no harder to decide than TPTL.

Theorem 4.4 (Complexity of TETL) *The validity problem for TETL is EXPSPACE-complete (with respect to polynomial-time reduction).*

Proof of Theorem 4.4 (1) To show that TETL is in EXPSPACE, we first use Lemma 4.3 to develop a nondeterministic exponential-space version of the tableau procedure that determines if a TETL-formula is satisfiable, and then apply Savitch's theorem. We refer to the corresponding proof for TPTL for details.

(2) The EXPSPACE-hardness of TETL follows immediately from the corresponding result for TPTL. ■

4.2 Deciding Metric Temporal Logic

We present a doubly-exponential-time, tableau-based decision procedure for MTL and show that the satisfiability problem for TPTL is EXPSPACE-complete. This result establishes that the decision problem for MTL is equally hard as the decision problem for TPTL, and that MTL, too, corresponds to an elementary fragment of the nonelementary first-order language \mathcal{L}_T . In Section 4.3, we will demonstrate how the tableau techniques can be applied to verify MTL-properties of finite-state timed transition systems.

4.2.1 Metric tableaux

The tableau algorithm for MTL uses the techniques that we have developed for TPTL. The crucial property that guarantees the finiteness of tableaux is that, in both cases, the temporal precedence between any two temporal contexts that are related by a timing constraint is uniquely determined. For TPTL-formulas, which contain only future operators, this property is guaranteed by the monotonicity of time; for MTL-formulas, which may contain past operators, it is due to the fact that MTL can relate only adjacent temporal contexts.

Before giving a formal definition of the tableau method for MTL, we indicate first how the algorithm proceeds for a sample input. Suppose that the time increases by one unit from a state to its successor (in general, the time increase between states can be bounded for any given formula, and thus reduced to a finite number of different cases). In order to satisfy, say, the formula $\Diamond_{<c} \psi$ in the current state, we have to satisfy either ψ now, or $\Diamond_{<c-1} \psi$ in the succeeding state. Continuing this splitting of requirements into a present and a future part, we will eventually arrive at the condition $\Diamond_{<1} \psi$, which forces ψ to be satisfied in the current state. Since every input formula ϕ generates only a finite number of requirements on states in the described fashion, ψ is satisfiable iff it is satisfiable in a finite tableau. By bounding the maximal size of this tableau, we obtain the following result.

Theorem 4.5 (Deciding MTL) *The validity problem for a formula ϕ of MTL can be decided in deterministic time exponential in $O(nC)$, where $n - 1$ is the number of boolean and temporal operators in ϕ , and $C - 1$ is the largest constant that occurs in ϕ as an interval end-point.*

Proof of Theorem 4.5 Suppose we are given an MTL-formula ϕ and wish to determine iff ϕ is valid; that is, iff its negation $\neg\phi$ is satisfiable.

We define the *closure set* $Closure(\phi)$ of the MTL-formula ϕ to be the smallest set containing ϕ that is closed under the following operation *Sub*:

$$\begin{aligned} Sub(\psi_1 \rightarrow \psi_2) &= \{\psi_1, \psi_2\}, \\ Sub(\odot_I \psi) &= \{\psi\}, \\ Sub(\odot_I \psi) &= \{\psi\}, \\ Sub(\psi_1 \mathcal{U} \psi_2) &= \{\psi_1, \psi_2\} \cup \{\odot(\psi_1 \mathcal{U}_{I-\delta} \psi_2) \mid \delta \geq 0\}, \end{aligned}$$

$$\text{Sub}(\psi_1 \mathcal{S}_I \psi_2) = \{\psi_1, \psi_2\} \cup \{\odot(\psi_1 \mathcal{S}_{I-\delta} \psi_2) \mid \delta \geq 0\}$$

such that

1. If I is an interval, then the expression $I - \delta$ stands for the interval $\{t \in \mathbb{N} \mid t + \delta \in I\}$. Note that if I is bounded by the right end-point d , then $I - \delta = \emptyset$ for all $\delta > d$, and if I is unbounded and has the left end-point c , then $I - \delta = \mathbb{N}$ for all $\delta > c$. This observation shows that the closure set of ϕ is finite.
2. If I is a congruence expression the form $\equiv_d c$, then the expression $I - \delta$ stands for the unchanged congruence expression $x \equiv_d c$.

Let $n - 1$ be the number of boolean and temporal operators in ϕ , and let $C - 1$ be the largest constant that occurs in ϕ as an interval end-point. It is not hard to see that

$$|\text{Closure}(\phi)| \leq 2nC.$$

As in TPTL, for checking the satisfiability of ϕ , we may restrict ourselves to timed state sequences $\rho = (\sigma, \mathbb{T})$ all of whose time steps $\mathbb{T}_i - \mathbb{T}_{i-1}$, for $i \geq 0$, are bounded by $C + k'$, where k' is the least common multiple of all d such a the congruence expression of the form $\equiv_d c$ occurs in ϕ . Let $\Delta = C + k'$. The time information in ρ has, therefore, finite-state character and can be modeled by the (new) time-difference propositions Prev_δ and $\text{Cong}_{\Delta, \delta'}$, for $0 \leq \delta \leq \Delta$ and $0 \leq \delta' < \Delta$. As usual, the proposition Prev_δ represents, in the initial state, the initial time δ and, in all other states, the time difference δ from the predecessor state; the proposition $\text{Cong}_{\Delta, \delta'}$ represents, in any state, the remainder δ' modulo Δ of the current time. For ease of presentation we use, in addition, the time-difference propositions Next_δ , for $0 \leq \delta \leq \Delta$, to represent, in any state, the time difference δ to the successor state.

Let $\text{Closure}^*(\phi)$ denote the set that is obtained from $\text{Closure}(\phi)$ by adding all of the new propositions Prev_δ , Next_δ , and $\text{Cong}_{\Delta, \delta'}$. A subset Φ of $\text{Closure}^*(\phi)$ is called (maximally) *consistent* iff it satisfies the following conditions, where all formulas range only over $\text{Closure}^*(\phi)$ (let I be an interval):

- $\text{Prev}_\delta \in \Phi$ for precisely one δ with $0 \leq \delta \leq \Delta$; this $\delta \in \mathbb{N}$ is referred to as δ_Φ^- .
- $\text{Next}_\delta \in \Phi$ for precisely one δ with $0 \leq \delta \leq \Delta$; this $\delta \in \mathbb{N}$ is referred to as δ_Φ^+ .
- $\text{Cong}_{\Delta, \delta} \in \Phi$ for precisely one δ with $0 \leq \delta < \Delta$; this $\delta \in \mathbb{N}$ is referred to as γ_Φ .

- $\text{false} \notin \Phi$.
- $\psi_1 \rightarrow \psi_2 \in \Phi$ iff either $\psi_1 \notin \Phi$ or $\psi_2 \in \Phi$.
- $\psi_1 \mathcal{U}_I \psi_2 \in \Phi$ iff
 - (1) $I \neq \emptyset$ and
 - (2) either $0 \in I$ and $\psi_2 \in \Phi$, or $\psi_1 \in \Phi$ and $\odot(\psi_1 \mathcal{U}_{I-\delta_\Phi^+} \psi_2) \in \Phi$.
- $\psi_1 \mathcal{U}_{\equiv_d c} \psi_2 \in \Phi$ iff either $\gamma_\Phi \equiv_d c$ and $\psi_2 \in \Phi$, or $\psi_1 \in \Phi$ and $\odot(\psi_1 \mathcal{U}_{\equiv_d c} \psi_2) \in \Phi$.
- $\psi_1 \mathcal{S}_I \psi_2 \in \Phi$ iff
 - (1) $I \neq \emptyset$ and
 - (2) either $0 \in I$ and $\psi_2 \in \Phi$, or $\psi_1 \in \Phi$ and $\odot(\psi_1 \mathcal{S}_{I-\delta_\Phi^+} \psi_2) \in \Phi$.
- $\psi_1 \mathcal{S}_{\equiv_d c} \psi_2 \in \Phi$ iff either $\gamma_\Phi \equiv_d c$ and $\psi_2 \in \Phi$, or $\psi_1 \in \Phi$ and $\odot(\psi_1 \mathcal{S}_{\equiv_d c} \psi_2) \in \Phi$.

The *initial tableau* $T(\phi)$ for the MTL-formula ϕ is a (finite) directed graph whose vertices are the consistent subsets of $\text{Closure}^-(\phi)$, and which contains an edge from Φ to Ψ iff all of the following conditions are met (let I be an interval):

- $\delta_\Phi^+ = \delta_\Psi^-$.
- $\gamma_\Psi \equiv_\Delta \gamma_\Phi + \delta_\Phi^+$.
- For all $\odot_I \psi \in \text{Closure}(\phi)$, $\odot_I \psi \in \Phi$ iff $\delta_\Phi^+ \in I$ and $\psi \in \Psi$.
- For all $\odot_{\equiv_d c} \psi \in \text{Closure}(\phi)$, $\odot_{\equiv_d c} \psi \in \Phi$ iff $\gamma(\Psi) \equiv_d c$ and $\psi \in \Psi$.
- For all $\odot_I \psi \in \text{Closure}(\phi)$, $\odot_I \psi \in \Psi$ iff $\delta_\Psi^- \in I$ and $\psi \in \Phi$.
- For all $\odot_{\equiv_d c} \psi \in \text{Closure}(\phi)$, $\odot_{\equiv_d c} \psi \in \Psi$ iff $\gamma(\Phi) \equiv_d c$ and $\psi \in \Phi$.

We show that all models of ϕ correspond to certain infinite paths through the initial tableau $T(\phi)$ for ϕ , and vice versa. An infinite path

$$\Phi : \Phi_0 \rightarrow \Phi_1 \rightarrow \Phi_2 \rightarrow \dots$$

through a tableau is called a ϕ -*path* iff it satisfies the following conditions (let I be an interval):

- $\phi \in \Phi_0$,
- Φ_0 contains no \odot -formula,
- For all $i \geq 0$, $\psi_1 \mathcal{U}_I \psi_2 \in \Phi_i$ implies $\psi_2 \in \Phi_j$ for some $j \geq i$ with $\Sigma_{i \leq k < j} \delta^+(\Phi_k) \in I$.
- For all $i \geq 0$, $\psi_1 \mathcal{U}_{\equiv_d c} \psi_2 \in \Phi_i$ implies $\psi_2 \in \Phi_j$ for some $j \geq i$ with $\gamma(\Phi_j) \equiv_d c$.
- $\delta_{\Phi_i}^- > 0$ for infinitely many $i \geq 0$.

It is not difficult to show that an MTL-formula ϕ is satisfiable iff the initial tableau $T(\phi)$ for ϕ contains a ϕ -path; the proof is similar to the corresponding argument for TPPTL. Since the $T(\phi)$ contains $O(k \cdot 2^{nC})$ vertices, each of size $O(nC)$, where k is the product of all constants that occur in ϕ , the initial tableau for ϕ can be constructed and checked for ϕ -paths in deterministic time exponential in $O(nC)$. ■

4.2.2 Complexity of metric temporal logic

Note that although the (worst-case) running time of the tableau algorithm is slightly faster for MTL than for TPPTL (for which the product of all constants appears in the exponent), it is still doubly exponential in the length of the input formula (under the assumption of binary encoding of constants). We show that the decision problem for MTL is indeed as hard as the decision problem for TPPTL.

Theorem 4.6 (Complexity of MTL) *The validity problem for MTL with or without past temporal operators is EXPSPACE-complete.*

Proof of Theorem 4.6 From a nondeterministic version of the tableau algorithm, it follows that MTL is in EXPSPACE. The corresponding lower bound can be shown similarly to the analogous result for TPPTL, by simulating EXPSPACE-bounded Turing machines. In particular, both of the TPPTL-formulas $\phi_{INITIAL}$ and ϕ_{MOVE} that are used in the proof of Theorem 4.2 can be directly translated into MTL without using past operators. ■

We point out that while PTL is PSPACE-complete, both TPPTL and MTL are exponentially more expensive. In fact, a closer look at our proof of the EXPSPACE-hardness of

TPTL suggests that any extension of PTL that allows the expression of a certain minimal set of timing constraints such as

$$\Box(p \rightarrow \Diamond_{=c} q),$$

which enforces that "the time of one state is a constant distance c from the time of another state," is EXPSPACE-hard, provided that all time constants are encoded in binary. Even the simplification of the digital-clock semantics that identifies *next-time* with *next-state* (i.e., time as a state counter) is of no help in complexity: our techniques can be used to show that already the introduction of the abbreviation \bigcirc^k for a sequence of k consecutive *next* operators makes PTL EXPSPACE-hard. It follows that the succinct encoding of time constants makes real-time reasoning (at least) exponentially more expensive than untimed reasoning.

4.3 Model Checking

Model checking is a powerful and well-established technique for the automatic verification of finite-state systems that compares a propositional temporal-logic specification of a system against a state-graph description of the system (for a survey of model checking and its applications, see, for example, [29]). Model-checking algorithms were first developed for branching-time logics [28, 36]. We build on the model-checking algorithm by Lichtenstein and Pnueli for the linear-time propositional temporal logic PTL [83].

Let us briefly review the main ideas that underlie model checking in the untimed case. Suppose that a system S is represented as a finite state graph (Kripke structure) $T(S)$; that is, all possible runs of S correspond to infinite paths through $T(S)$ that satisfy certain fairness requirements. Furthermore, suppose that the specification of S is given as a formula ϕ of PTL. The tableau construction for testing the satisfiability of the negated formula $\neg\phi$ can then be used to solve the verification problem:

Do all possible runs of the system S satisfy the specification ϕ ?

The algorithm proceeds in two steps:

1. We construct the initial tableau $T(\neg\phi)$ for $\neg\phi$, which captures precisely the models of $\neg\phi$. Then we can reformulate the verification question as follows: is there an

infinite path that is common to both finite state graphs, $T(S)$ and $T(\neg\phi)$, and that corresponds both to a possible run of S and a model of $\neg\phi$? Clearly, the system S meets the specification ϕ if and only if this is not the case.

2. The reformulated verification problem can be solved by a product construction on finite Kripke structures. We construct the product of the two state graphs $T(S)$ and $T(\neg\phi)$ and check if it contains an infinite path that satisfies certain fairness conditions.

The tableau methods for TPTL and MTL allow us to generalize the model-checking approach to real-time systems and real-time properties. We define *timed* state graphs in a way so that they subsume our system model — finite-state timed transition systems. Then we define the product of such timed structures, which leads to an algorithm that determines if a finite-state timed transition system meets a specification that is given in TPTL or MTL. We also show that the problem of checking if a formula ϕ of TPTL or MTL is satisfied in a given structure is EXPSPACE-complete and thus, in general, equally hard as deciding if ϕ is satisfiable in any structure. The complexity of the model-checking problem, however, is doubly exponential only in the size of the formula, which is usually much smaller than the size of the structure.

4.3.1 Finite-state real-time systems

Let us define the notion of finite-state real-time system as a finite state graph (Kripke structure) that contains finitely many time-difference propositions. We call these timed structures *tableaux*, and show that finite-state timed transition systems can, just like formulas of TPTL and MTL, be represented as tableaux.

Timed Kripke structures

Let P_S be a finite set of propositions. We represent finite-state real-time systems by finite, directed state graphs whose vertices are labeled by sets of propositions. Each vertex contains finite

State information A proposition $p \in P_S$ holds at vertex v iff v is labeled with p .

Time information Every vertex is labeled with precisely one time-difference proposition $Prev_\delta$ or $Prev_{\geq\delta}$ that indicates that the time difference from the predecessor vertices is exactly δ time units or at least δ time units, respectively.

Formally, a (*real-time*) tableau $T = \langle V, \sigma, \delta^-, V_0, E \rangle$ over the set P_S of propositions consists of

- a finite set V of vertices,
- a *state* labeling function $\sigma: V \rightarrow 2^{P_S}$ that labels every vertex v with a state $\sigma_v \subseteq P_S$,
- a *time* labeling function $\delta^-: V \rightarrow \mathbb{N} \times 2$ that labels every vertex v with a time-difference proposition δ_v^- , which is either $Prev_\delta$ or $Prev_{\geq\delta}$ for some $\delta \in \mathbb{N}$,
- a set $V_0 \subseteq V$ of initial vertices,
- a set $E \subseteq V^2$ of edges.

In accordance with the intuitive operational semantics that is associated with a tableau, we say that a timed state sequence $\rho = (\sigma, T)$ over \mathbb{N} is a *computation* of the tableau T iff there is an infinite path $v_0 v_1 v_2 \dots$ through T such that for all $i \geq 0$,

- (1) $\sigma_i = \sigma_{v_i}$ and
- (2) if $\delta_{v_i}^- = Prev_\delta$, then $T_i = T_{i-1} + \delta$; otherwise $T_i \geq T_{i-1} + \delta_{v_i}^-$.

Thus every tableau T defines a digital real-time property — the set $\Pi(T)$ of computations of T . We say that the tableau T *satisfies* the formula ϕ of TPTL or MTL iff there is a computation of T that is a model of ϕ . Dually, the formula ϕ is called *valid* over the tableau T iff all computations of T are models of ϕ :

$$\Pi(T) \subseteq \Pi(\phi).$$

The problem of *model checking* is to determine iff a formula is valid over a tableau. By representing finite-state timed transition systems as tableaux, we can subject them to model-checking algorithms.

Finite-state timed transition systems

In Chapter 2, we have introduced timed transition systems as our model for real-time systems. Here we associate a tableau $T(S)$ with every finite-state timed transition system S such that the computations of $T(S)$ are exactly the runs of S :

$$\Pi(T(S)) = \text{Det}(\Pi(S)).$$

Then we can use model checking to verify that all runs of a finite-state timed transition system satisfy a formula of TPTL or MTL.

Let $S = \langle \Sigma, \Theta, \mathcal{T}, l, u \rangle$ be a finite-state timed transition system; that is, the set Σ of states is finite. We may therefore assume that every state of S is uniquely characterized by a subset of a suitable finite set P_S of propositions (i.e., $\Sigma = 2^{P_S}$). For instance, every timed transition system that is defined in the transition diagram language of Chapter 2 uses the interpretations of a finite set of variables as states; if each of these variables ranges over a finite domain, then S is a finite-state transition system. We construct the tableau $T(S)$ in two steps:

1. First, we dispense with all timing requirements of the timed transition system S by adding the timing information to the states. For this purpose, we have defined, in Subsection 2.1.2, the explicit-clock transition system $S' = \langle \Sigma', \Theta', \mathcal{T}' \rangle$ that is associated with S .
2. Secondly, we merge the infinite number of states of S' into a finite number of equivalence classes.

The explicit-clock transition system S' contains infinitely many states, because the clock variable t and the delay counters d_r range over the infinite domain \mathbb{N} . Let $C_S - 1 \in \mathbb{N}$ be the largest constant of all finite minimal and maximal delays of S . To reduce the information that is provided by the clock variable t to a finite domain, we use the following simple observation. Let $\rho = (\sigma, T)$ and $\rho' = (\sigma, T')$ be two timed state sequences such that for all $i \geq 0$, either $T_i - T_{i-1} = T'_i - T'_{i-1}$ or both $T_i - T_{i-1} \geq C_S$ and $T'_i - T'_{i-1} \geq C_S$. Then ρ is a run of S iff ρ' is a run of S . Thus it suffices to model time by the finite set

$$\text{Prev} = \{\text{Prev}_0, \dots, \text{Prev}_{C_S-1}, \text{Prev}_{\geq C_S}\}$$

of time-difference propositions. The information that is provided by the delay counters has also finite-state character, because if the value of a delay counter is, in any state, at least C_S , then its actual value is of no importance.

Thus we define the equivalence relation \sim on the states of S' such that $\sigma \sim \sigma'$ iff

- (1) $\sigma^- = \sigma'^-$ and
- (2) for all transitions $\tau \in \mathcal{T}$, either $\sigma(d_\tau) = \sigma'(d_\tau) < C_S$ or both $\sigma(d_\tau) \geq C_S$ and $\sigma'(d_\tau) \geq C_S$

for all $\sigma, \sigma' \in \Sigma^s$. The tableau $T(S) = \langle V, \sigma, \delta^-, V_0, E \rangle$ for S is defined as follows:

- Every vertex $v = (s, t)$ of $T(S)$ is a pair that consists of an equivalence class s of states of S' and a time-difference proposition t :

$$V = \{[\sigma]_{\sim} \mid \sigma \in \Sigma^s\} \times \text{Prev}.$$

- If $v = ([\sigma]_{\sim}, t)$ for $\sigma \in \Sigma^s$, then $\sigma_v = \sigma^-$.
- If $v = (s, t)$, then $\delta_v^- = t$.
- A vertex $v = ([\sigma]_{\sim}, t)$, for $\sigma \in \Sigma^s$, is initial iff $\sigma^- \in \Theta$.
- There is an edge from the vertex $([\sigma]_{\sim}, t)$ to the vertex $([\sigma']_{\sim}, t')$, for $\sigma, \sigma' \in \Sigma^s$, iff
 - either there is a transition $\tau \in \mathcal{T}$ such that $(\sigma, \sigma') \in \tau^s$ and $t' = \text{Prev}_0$,
 - or $(\sigma, \sigma') \in \tau_T^s$ and $t' = \text{Prev}_{\sigma'(t) - \sigma(t)}$,
 - or $(\sigma, \sigma') \in \tau_T^s$ and $\sigma'(t) - \sigma(t) \geq C_S$ and $t' = \text{Prev}_{\geq C_S}$.

It is not hard to see that if the timed transition system S contains no maximal delays ∞ , then the tableau $T(S)$ for S defines the set $\text{Det}(\Pi(S))$ of runs of S . We have not provided any means to place fairness assumptions on tableaux, which is necessary to handle infinite upper-bound requirements and can be done analogously to the untimed case [83].

We remark that the tableau $T(S)$ for S contains $O(|\Sigma| \cdot C_S^{|\mathcal{T}|})$ vertices, because there are $|\mathcal{T}|$ delay counters each of which ranges over a domain of size $C_S + 1$. Thus there is an exponential blow-up in moving from the description of a timed transition system to a tableau (independent of whether the maximal and minimal delays of the system are given in a unary or a binary encoding).

4.3.2 Model-checking algorithms

Suppose that we are given

1. a finite-state real-time system in form of a tableau T_S over the set P_S of propositions. Let C_S be the largest constant δ for which T_S contains a time-difference proposition $Prev_\delta$ or $Prev_{\geq\delta}$.
2. a formula ϕ of TPTL or MTL over the set P of propositions. Recall that the product k of all constants in ϕ is the largest constant δ for which the initial tableau $T(\phi)$ for ϕ contains a time-difference proposition $Prev_\delta$.

We define the *product* $T = T_S \times T(\phi)$ of the two structures T_S and $T(\phi)$ to be a finite, directed graph whose vertices are pairs of T_S -vertices and $T(\phi)$ -vertices:

- Each vertex (v, Φ) of T consists of a vertex v of T_S and a vertex Φ of $T(\phi)$ such that
 1. the *state* information in v and Φ is compatible; that is, $p \in \sigma_v$ iff $p \in \Phi$ (or $z.p \in \Phi$, in the case of TPTL) for all propositions $p \in P_S \cap P$.
 2. the *time* information in v and Φ is compatible; that is, either $\delta_v^- = \delta_\Phi^-$, or $\delta_\Phi^- = Prev_k$ and $\delta_v^- \in \{Prev_\delta, Prev_{\geq\delta}\}$ for some $\delta \geq k$.
- T contains an edge from the vertex (v_1, Φ_1) to the vertex (v_2, Φ_2) iff T_S contains an edge from v_1 to v_2 and $T(\phi)$ contains an edge from Φ_1 to Φ_2 .

The size of the product $T_S \times T(\phi)$ is clearly linear in the product of the sizes of T_S and $T(\phi)$.

For both TPTL and MTL we say that an infinite path through the product $T_S \times T(\phi)$ is a ϕ -path iff its second projection is a ϕ -path through $T(\phi)$; it is an *initialized* ϕ -path iff, in addition, it starts at a vertex whose first projection is an initial vertex of T_S . The following lemma, which follows immediately from Lemma 4.4 and the proof of Theorem 4.5, confirms that our product construction has the intended effect.

Lemma 4.8 (Tableau product) *The tableau T_S satisfies the formula ϕ of TPTL or MTL iff the product $T_S \times T(\phi)$ contains an initialized ϕ -path.*

This lemma suggests a model-checking algorithm. To see if all runs of a finite-state timed transition system S satisfy a formula ϕ of TPTL or MTL:

1. Construct the tableau $T(S)$ for S .
2. Construct the initial tableau $T(\neg\phi)$ for the negated formula $\neg\phi$.
3. Construct the tableau product $T = T(S) \times T(\neg\phi)$.
4. Check if T contains a $\neg\phi$ -path. The system S meets the specification ϕ iff this is not the case.

According to different versions of fairness, various variants of the notion of ϕ -paths through the tableau product can be defined, and checked for, as in the untimed case [83]. This allows an extension of the method to arbitrary finite-state timed transition systems, which may contain maximal delays of ∞ . Since a structure can be checked for ϕ -paths in polynomial time, the running time of the algorithm is determined by the size of the tableau product T , which contains $O(|T(S)| \cdot |T(\phi)|)$ vertices. Recall that the size of $T(S)$ may be exponentially larger than the description of S , and the size of $T(\phi)$ may be two exponentials larger than ϕ itself. Thus

Proposition 4.1 (Model checking) *The problem if all runs of a finite-state timed transition system S are models of a formula ϕ of TPTL or MTL can be decided in deterministic time exponential in $|S| \cdot 2^{|\phi|}$, where $|S|$ is the size of the system description and $|\phi|$ is the length of the formula.*

The model-checking algorithm we have outlined can, of course, be streamlined in myriad ways. To begin with, it is not necessary to construct both tableaux and the product in their entirety, but all four steps of the algorithm can be overlapped. For details, we refer the reader to recent developments in untimed model-checking procedures (see, for example, [24]). We also remark that the product construction of timed structures can be used to check if one finite-state timed transition system implements (refines) another finite-state timed transition system.

4.3.3 Complexity of model checking

We show that the problem of determining if a formula ϕ of TPTL or MTL is valid in a given structure is, in general, equally hard as the problem of determining if ϕ is unconditionally valid.

Theorem 4.7 (Complexity of model checking) *The problem of deciding if a formula of TPTL or MTL (with or without past temporal operators) is valid over a tableau is EXPSPACE-complete.*

Proof of Theorem 4.7 [EXPSPACE] As is the case for PTL [119], timed and metric model checking are polynomial-time reducible to the validity problems for TPTL and MTL, respectively. Given the formula ϕ and the tableau $T = (V, \sigma, \delta^-, V_0, E)$, we construct a formula ϕ_T , whose length depends polynomially on the sizes of both T and ϕ , and which is valid iff ϕ is valid over T . For every vertex $v_i \in V$, we introduce a new proposition p_i and build the MTL-formula

$$p_i \rightarrow \left(\bigwedge_{q \in \sigma_{v_i}} q \wedge \bigwedge_{q \notin \sigma_{v_i}} \neg q \wedge \bigvee_{(v_i, v_j) \in E} \odot_{\delta_{v_i}^-} p_j \right), \quad (\psi_i)$$

where $\odot_{\delta_{v_i}^-}$ stands for the time-bounded operator $\odot_{=\delta}$ if $\delta_{v_i}^- = Prev_{\delta}$, and for $\odot_{\geq \delta}$ if $\delta_{v_i}^- = Prev_{\geq \delta}$. Furthermore, let the formula ψ assert that exactly one of the propositions p_i , for $v_i \in V$, is true. It is not hard to see that the formula

$$\left(\bigvee_{v_i \in V_0} p_i \wedge \Box(\psi \wedge \bigwedge_{v_i \in V} \psi_i) \right) \rightarrow \phi \quad (\phi_T)$$

has the desired properties. Note that the antecedent of the formula ϕ_T contains no past operators and, thus, can be directly translated into TPTL.

[EXPSPACE-hardness] To reduce the validity problems for TPTL and MTL to model checking, it suffices to give a tableau T of constant size such that formula ϕ is valid iff ϕ is valid over T . Simply choose $T = (V, \sigma, \delta^-, V, V^2)$ to be the complete graph over all subsets of P , the propositions that occur in ϕ , and label all vertices by the time-difference proposition $Prev_{\geq 0}$. ■

Since our translation from finite-state timed transition systems to tableaux involves an exponential blow-up in size, Theorem 4.7 does not locate the exact complexity of verifying finite-state timed transition systems. Indeed, as we shall see next, a finite-state timed transition system can be directly encoded in TPTL or MTL without an exponential factor.

Encoding finite-state transition systems

Let P be a finite set of propositions and $S = (\Sigma, \Theta, T, l, u)$ a timed transition system with $\Sigma = 2^P$. We construct a formula ϕ_S of MTL without past operators, the *logical*

representation of the finite-state system S , such that the possible runs of S are exactly the models of ϕ_S ; that is,

$$Det(\Pi(S)) = \Pi(\phi_S).$$

Since ϕ_S will contain no past operators, it can be directly translated into TPTL.

For all states $\sigma \subseteq P$ of S , let ϕ_σ be the conjunction

$$\bigwedge_{q \in \sigma} q \wedge \bigwedge_{q \notin \sigma} \neg q$$

that characterizes the state σ . The condition that a transition $\tau \in \mathcal{T}$ is enabled is then expressed by the formula

$$enabled(\tau) : \bigvee_{\exists \sigma' \in \Sigma. (\sigma, \sigma') \in \tau} \phi_{\sigma'}.$$

Recall that all runs are deterministic timed state sequences; that is,

$$\Box \bigwedge_{\sigma \in \Sigma} (\phi_\sigma \rightarrow (\odot_{=0} true \vee \odot \phi_\sigma)). \quad (\phi_{DET})$$

For every transition $\tau \in \mathcal{T}$, we use a new proposition $taken_\tau$ that indicates, in any state of a run of S , if the transition τ is taken. Consequently, we require that

$$taken_\tau \rightarrow \bigwedge_{\tau' \neq \tau} \neg taken_{\tau'}, \quad (\phi_\tau^1)$$

$$taken_\tau \rightarrow \bigvee_{(\sigma, \sigma') \in \tau} (\phi_\sigma \wedge \odot \phi_{\sigma'}). \quad (\phi_\tau^2)$$

The formula ϕ_S consists, in addition to the conjunct

$$\phi_{DET} \wedge \Box \bigwedge_{\tau \in \mathcal{T}} (\phi_\tau^1 \wedge \phi_\tau^2),$$

of the following four conjunctive parts:

Initiality	$\bigvee_{\sigma \in \Theta} \phi_\sigma.$
Consecution	$\Box \bigvee_{\tau \in \mathcal{T}} taken_\tau.$
Deterministic lower bound	$\bigwedge_{\tau \in \mathcal{T}} \Box (\neg enabled(\tau) \rightarrow \Box_{< t_\tau} \neg taken_\tau).$
Deterministic upper bound	$\bigwedge_{\tau \in \mathcal{T}} \Box (enabled(\tau) \rightarrow \Diamond_{\leq u_\tau} (taken_\tau \vee \neg enabled(\tau))).$

It is not hard to see that a timed state sequence ρ is indeed a model of ϕ_S iff ρ is a run of the system S . Three additional remarks are in order.

1. The logical representation reveals that every real-time property that is defined by a timed transition system is an intersection of the time-invariant property that is defined by the underlying untimed transition system, bounded-invariance properties that ensure all lower-bound requirements, and bounded-response properties that ensure the upper-bound requirements. It follows that timed transition systems can be encoded by any logic that can express *bounded-invariance* and *bounded-response* properties. Also, the digitizability of the set of runs of S follows as a corollary to Proposition 3.1.
2. If every pair of states in Σ is related by at most one transition in \mathcal{T} , then the auxiliary propositions $taken_\tau$ are not needed and the formula ϕ_S employs, just like S , only propositions from P :

$$taken_\tau \leftrightarrow \bigvee_{(\sigma, \sigma') \in \tau} (\phi_\sigma \wedge \odot \phi_{\sigma'}).$$

We say that a timed transition system that satisfies this condition is *terse*. The terseness of S ensures that every run ρ of S uniquely determines the infinite sequence of transitions that are taken along ρ . Note that terseness, however, does not imply determinism. We point out that, for example, all timed transition systems that are defined in the timed transition diagram language are terse.

3. Note that the length of the formula ϕ_S is polynomially related to the size of the description of S (provided that all natural number constants in ϕ_S and S are given by the same encoding). By an argument similar to the proof of Theorem 4.7, it follows that the problem of verifying a finite-state timed transition system with respect to a TPTL-specification or an MTL-specification is in EXPSPACE.

Chapter 5

Deductive Verification: General Part

In Chapter 4, we have shown that the real-time temporal logic TPTL yields to algorithmic techniques for the verification of finite-state systems. For a given timed transition system S , the tableau-based methods, however, may not be

1. *applicable*, because the state space of S is infinite. For example, this is the case for all programs that contain variables that range over infinite domains, such as the integers.
2. *feasible*, because the state space of S is too large. This problem, which is known as the "state explosion problem," arises especially with highly parallel systems, as the number of states in a transition system grows exponentially with the number of parallel processes. The growth of the state space is particularly hindering for state-based *real-time* verification, which is, as we have seen, exponentially more expensive than untimed verification.

Thus it is imperative that a formal approach to real-time verification provides both semantic (i.e., algorithmic) as well as syntactic (i.e., deductive) methods. In this chapter, we develop a complete proof system for TPTL to complement the model-checking algorithm. The proof system can be used for the deductive verification of real-time systems even if they cannot be represented as finite-state graphs.

Just as proof systems for the propositional temporal logic PTL consist of a general, modal, part and special axioms for linear structures, we arrive at the proof system for

TPTL in two steps. First (in Section 5.1), we axiomatize the freeze quantifier for arbitrary modal logics that are interpreted over Kripke structures in which a value is associated with every possible world, completely independent of the notion of "time." Since these modal logics are fragments of the corresponding first-order versions, we dub them *half-order*. We also show that half-order modal logic generalizes the classical first-order predicate calculus, which can be embedded.

Secondly (in Section 5.2), we obtain half-order *temporal* logic by restricting ourselves to certain linear structures. We interpret possible worlds as system states, the accessibility relation as temporal ordering between states, and the value that is associated with a state as its time. The resulting structures are timed state sequences and, hence, precisely the interpretations for TPTL. By adding appropriate axioms for linear structures and the timing constraints that are admitted in TPTL, we obtain a complete proof system for TPTL. In fact, the choice of timing constraints of TPTL turns out to be crucial; we show that half-order temporal logic in general is Π_1^1 -hard, and therefore not axiomatizable.

Finally (in Section 5.3), we indicate how the proof system for TPTL can be used to verify TPTL-specifications of timed transition systems.

5.1 Half-order Modal Logic

We introduce modal logics that are interpreted over Kripke structures each of whose possible worlds (states) s has a value $|s|$ associated with it. Ordinary first-order function and relation symbols, including equality, perform operations and tests on these values. However, instead of ordinary universal (and existential) quantification, the access to values is kept extremely local: the freeze quantifier " x ." binds x to the value that is associated with the current state. For example, the formula $x. \Diamond y. p(x, y)$ is true in an interpretation with initial state s iff there is a state t accessible from s such that the relation p , as interpreted in t , holds between the value $|s|$ associated with s and the value $|t|$ associated with t .

There is a wide, and largely confusing, variety of different ways to add conventional quantification to modal logic, for only some of which completeness results have been achieved, and some of which are known to be incomplete (see [43] for an excellent survey). The situation for the freeze quantifier is, fortunately, much cleaner: we show that modal logics with freeze quantification are axiomatizable, yet not necessarily decidable, fragments of

certain corresponding first-order modal logics. This explains the attribute "half-order" for the freeze quantifier.

5.1.1 Syntax and semantics

The formulas of half-order modal logic are built from first-order atoms, which include equations, by boolean connectives, the modal operator \Box , and the freeze quantifier. Let V be an infinite set of variables, and F and R be sets of function and relation symbols, respectively. We assume that all of these sets can be effectively enumerated. The *terms* π , *atomic formulas* α , and *formulas* ϕ of half-order modal logic are inductively defined as follows:

$$\pi := x \mid f\bar{\pi}$$

$$\alpha := \pi_1 = \pi_2 \mid p\bar{\pi}$$

$$\phi := \alpha \mid \text{false} \mid \phi_1 \rightarrow \phi_2 \mid \Box\phi \mid x.\phi$$

for $x \in V$, $f \in F$, and $p \in R$. We write $\bar{\pi}$ to denote a tuple of terms; for example, if f is a binary function symbol then $f\bar{\pi}$ stands for $f\pi_1\pi_2$, or $f(\pi_1, \pi_2)$. The boolean connectives *true*, \neg , \wedge , \vee , and \leftrightarrow are defined in terms of *false* and \rightarrow in the standard way; $\Diamond\phi$ is an abbreviation for $\neg\Box\neg\phi$. Throughout this chapter, we use π and α (possibly subscripted) to stand for arbitrary terms and atomic formulas, respectively; formulas are, as usual, denoted by ϕ , ψ , and φ .

An *interpretation*

$$\mathcal{M} = (\Sigma, \rightarrow_{\Box}, \Upsilon, \|\cdot\|, [x]_{x \in V}, [f]_{f \in F}, [p]_{p \in R}, s_0)$$

for half-order modal logic consists of

- a set Σ of states,
- an accessibility relation $\rightarrow_{\Box} \subseteq \Sigma^2$ on the states,
- a set Υ of values,
- a value function $\|\cdot\|: \Sigma \rightarrow \Upsilon$ that associates a value $\|s\|$ with every state s ,
- a rigid assignment function $[x] \in \Upsilon$ for all variables $x \in V$,
- a rigid assignment function $[f]: \bar{\Upsilon} \rightarrow \Upsilon$ for all function symbols $f \in F$,
- a flexible assignment function $[p]: \Sigma \rightarrow 2^{\Upsilon}$ for all relation symbols $p \in R$,
- an initial state $s_0 \in \Sigma$.

Note that all terms are given a state-independent (rigid) meaning, while the interpretation of relation symbols is state-dependent (flexible). The rigidity restriction on terms is required for the completeness proof; the flexibility of relation symbols is necessary to cover TPTL, whose propositions are interpreted in a state-dependent fashion.

The interpretation \mathcal{M} is a *model* of the formula ϕ iff $\mathcal{M} \models \phi$, for the following inductive definition of the satisfaction relation \models :

$$\mathcal{M} \models \pi_1 = \pi_2 \text{ iff } \llbracket \pi_1 \rrbracket = \llbracket \pi_2 \rrbracket, \text{ for } \llbracket f\bar{\pi} \rrbracket = \llbracket f \rrbracket(\llbracket \bar{\pi} \rrbracket).$$

$$\mathcal{M} \models p\bar{\pi} \text{ iff } \llbracket \bar{\pi} \rrbracket \in \llbracket p \rrbracket(s_0).$$

$$\mathcal{M} \not\models \text{false}.$$

$$\mathcal{M} \models \phi_1 \rightarrow \phi_2 \text{ iff } \mathcal{M} \models \phi_1 \text{ implies } \mathcal{M} \models \phi_2.$$

$$\mathcal{M} \models \Box \phi \text{ iff } \mathcal{M}[s_0 := s] \models \phi \text{ for all } s \in \Sigma \text{ with } s_0 \rightarrow \Box s.$$

$$\mathcal{M} \models x.\phi \text{ iff } \mathcal{M}[\llbracket x \rrbracket := |s_0|] \models \phi.$$

Here $\mathcal{M}[s_0 := s]$ denotes the interpretation that differs from \mathcal{M} only in its initial state, s ; the interpretation $\mathcal{M}[\llbracket x \rrbracket := |s_0|]$ differs from \mathcal{M} only in its assignment function for x . Thus, the semantic clause for formulas of the form $\Box \phi$ specifies that ϕ is interpreted in all states accessible from the current state. The clause for $x.\phi$ asserts that all occurrences of x in ϕ refer to the value that is associated with the current state.

The formula ϕ is *satisfiable* (*valid*) iff some (every) interpretation is a model of ϕ . We write $\models \phi$ to denote that ϕ is valid. Two formulas are *equivalent* iff they have the same models. While it is customary in modal logic to separate the initial state from a Kripke structure, we have merged both components of an interpretation, because we are only interested in general validity, not validity in a given structure.

Observe that we can faithfully embed half-order modal logic into a first-order modal logic with the set T of values as constant domain for all states, and rigid terms, with the exception of *one* flexible constant symbol, say t , that denotes, in every state, the value associated with that state; that is, $\llbracket t \rrbracket(s) = |s|$. The freeze quantifier $x.\phi$ is translated as

$$\forall x. (x = t \rightarrow \phi)$$

(or, equivalently, $\exists x.(x = t \wedge \phi)$). In other words, half-order modal logic captures the fragment of first-order modal logic in which every rigid (global) variable is, immediately upon introduction, bound to the current value of the flexible (state) variable t . This makes the conventional first-order quantifiers superfluous.

5.1.2 Proof system

Let K_P be the propositional modal logic that is determined by the class of all Kripke structures. We extend the proof system for K_P by axioms and inference rules for both the freeze quantifier and equality. Recall that the deductive calculus for K_P consists of a complete proof system **PROP** for propositional logic, say,

PROP1 all tautologies are axioms,
 from ϕ_1
PROP2 and $\frac{\phi_1 \rightarrow \phi_2}{\text{infer } \phi_2}$

as well as the following axiom schema and necessitation rule, which completely characterize the modal operator \Box with respect to Kripke semantics [81]:

K1 $\Box(\phi_1 \rightarrow \phi_2) \rightarrow \Box\phi_1 \rightarrow \Box\phi_2$,
K2 from $\frac{\phi}{\text{infer } \Box\phi}$

Let us abbreviate the formula $\phi \rightarrow \Box\psi$ to $\phi \Rightarrow \psi$ (this convention will serve our purposes better than the standard interpretation of $\phi \Rightarrow \psi$ as $\Box(\phi \rightarrow \psi)$); the operator \Rightarrow associates to the right, as does \rightarrow . The freeze quantifier is characterized by an axiom schema that asserts its functionality,

Q1 $x.(\phi_1 \rightarrow \phi_2) \leftrightarrow (x.\phi_1 \rightarrow x.\phi_2)$,

and an introduction rule for every modal context,

Q2 from $\frac{\phi_1 \Rightarrow \dots \Rightarrow \phi_n \Rightarrow \psi}{\text{infer } \phi_1 \Rightarrow \dots \Rightarrow \phi_n \Rightarrow x.\psi}$, provided that $x \notin \phi_i$ for all $1 \leq i \leq n$

(we write $x \in \phi$ iff the variable x occurs freely in ϕ). The simplest instances of this rule (take $n = 0$) are of the form

$$\text{Q2}^* \quad \frac{\text{from } \phi}{\text{infer } x.\phi.}$$

Only the elimination of vacuous quantifier occurrences is sound:

$$\text{Q3} \quad x.\phi \leftrightarrow \phi \text{ if } x \notin \phi.$$

Let $\phi(\pi_1 := \pi_2)$ (and $\phi[\pi_1 := \pi_2]$) denote a formula that results from ϕ by safely replacing zero, one, or more (all, respectively) free occurrences of π_1 by π_2 . *Safe replacement* means, as usual, that no free occurrence of π_1 that is replaced, is within the scope of a quantifier binding a variable of π_2 ; whenever we write $\phi[\pi_1 := \pi_2]$, there is the implicit condition that *all* free occurrences of π_1 in ϕ can be safely replaced by π_2 . We add conventional congruence axioms for equality, for instance,

$$\text{EQ1} \quad \pi = \pi,$$

$$\text{EQ2} \quad \pi_1 = \pi_2 \rightarrow \alpha \rightarrow \alpha(\pi_1 := \pi_2),$$

two axiom schemata that assert the rigidity of terms,

$$\text{RIG1} \quad \pi_1 = \pi_2 \rightarrow \Box \pi_1 = \pi_2,$$

$$\text{RIG2} \quad \pi_1 \neq \pi_2 \rightarrow \Box \pi_1 \neq \pi_2,$$

and an axiom schema that states that the value associated with every state is unique,

$$\text{QEQ} \quad x.y.z = y.$$

To summarize, we are given the *logical axioms* PROP1, K1, Q1, Q3, EQ1-2, RIG1-2, and QEQ, and the *inference rules* PROP2, K2, and Q2. A *half-order normal logic* is a set of formulas that contains all logical axioms and is closed under all inference rules. Note that, since the proof system includes PROP and K1-2, every normal logic contains all instances of valid schemata of the propositional modal logic K_P .

Sample proofs

We demonstrate the use of the proof system by deriving some additional theorems of half-order modal logic that will be useful later.

Lemma 5.1 (Sample theorems) *Any normal logic contains the following formulas:*

$$\text{Q4} \quad x. \neg \phi \leftrightarrow \neg x. \phi.$$

$$\text{EQ3} \quad \pi_1 = \pi_2 \rightarrow \phi \rightarrow \phi(\pi_1 := \pi_2).$$

$$\text{EQ4} \quad x. x = \pi \rightarrow (x. \phi \leftrightarrow \phi[x := \pi]).$$

$$\text{VAR1} \quad x. \phi \rightarrow y. \phi[x := y] \text{ if } y \notin \phi.$$

$$\text{VAR2} \quad \phi \leftrightarrow \phi' \text{ if } \phi' \text{ results from } \phi \text{ by safe renaming of bound variables.}$$

$$\text{VAR3} \quad x. y. \phi \leftrightarrow x. \phi[y := x].$$

Proof of Lemma 5.1 Q4 states that the freeze quantifier is its own dual. It follows from Q1:

$$x. (\phi_1 \rightarrow \text{false}) \leftrightarrow (x. \phi_1 \rightarrow x. \text{false})$$

by Q3 and PROP.

EQ3 generalizes the equality axiom EQ2 to arbitrary formulas ϕ . To establish it, we use induction on the structure of ϕ . The atomic base case holds by EQ2. The propositional cases follow from the induction hypothesis by PROP. Observe that

$$\pi_1 = \pi_2 \rightarrow \Box \phi \rightarrow (\Box \phi)(\pi_1 := \pi_2)$$

equals

$$\pi_1 = \pi_2 \rightarrow \Box \phi \rightarrow \Box \phi(\pi_1 := \pi_2),$$

which follows from the induction hypothesis by K2, K1, RIG1, and PROP. If $x \notin \pi_1$ and $x \notin \pi_2$, then $(x. \phi)(\pi_1 := \pi_2)$ equals $x. \phi(\pi_1 := \pi_2)$, and

$$\pi_1 = \pi_2 \rightarrow x. \phi \rightarrow x. \phi(\pi_1 := \pi_2),$$

follows from the induction hypothesis by Q2", Q1, Q3, and PROP. If, on the other hand, $x \in \pi_1$ or $x \in \pi_2$, then $(x. \phi)(\pi_1 := \pi_2)$ must be $x. \phi$; the corresponding inductive step holds by PROP.

From now on, we will often omit to mention applications of PROP explicitly. Similarly to our use of PROP, we may refer to any of K1-2 and Q1-4 simply by K or Q. We write Q^* if all applications of the rule Q2 are instances of $Q2^*$.

EQ4 will be used extensively in the completeness proof, and can be derived from EQ3 by Q^* .

VAR1 shows that bound variables can be renamed at the top level of formulas; it follows from EQ4 by Q^* and QEQ.

VAR2 generalizes VAR1 and is shown by structural induction.

VAR3 demonstrates that adjacent quantifiers can be combined; it follows from EQ4 by Q^* and QEQ. Together with Q1, VAR3 implies that every formula ϕ is equivalent to some formula $\alpha \cdot \phi'$ that contains at most one quantifier per modal level; that is, every quantifier in ϕ' follows a modal operator. Alternatively, any formula can be put into a normal form in which every quantifier precedes a modal operator or an atomic formula. ■

Soundness

Let Φ be a set of formulas and Λ_Φ the intersection of all normal logics containing Φ . Clearly, Λ_Φ is again a normal logic; the formulas of Φ are called the *nonlogical axioms* of Λ_Φ . In particular, $K = \Lambda_\emptyset$ is the smallest normal logic, and $\perp = \Lambda_{\{\text{false}\}}$, the set of all formulas, is the largest one. Our goal is to show that K is precisely the set of all valid formulas; that is, that the given proof system is both sound and complete for half-order modal logic. First we show that every formula of K is true under every interpretation; the completeness proof is deferred to the next subsection.

Lemma 5.2 (Soundness) *The logical axioms are valid. If all of the antecedents of an inference rule are valid, then so is the consequent.*

Proof of Lemma 5.2 The soundness of PROP, K1, Q1, EQ1-2, RIG1-2, and QEQ follows immediately from the definition of truth under an interpretation. The argument for K2 is the same as in propositional modal logic.

To show Q3 to be valid, we use the following fact:

$$\mathcal{M} \models \phi \text{ iff } \mathcal{M}[[z] := u] \models \phi \text{ for all } z \notin \phi \text{ and values } u \text{ of } \mathcal{M}. \quad (\dagger)$$

This can be established by induction on the structure of ϕ .

It remains to be shown that Q2 is sound. Suppose that $\phi_1 \Rightarrow \dots \Rightarrow \phi_n \Rightarrow \psi$ is valid and $x \notin \phi_i$ for all $1 \leq i \leq n$. Now consider an arbitrary interpretation \mathcal{M} , and show that $\phi_1 \Rightarrow \dots \Rightarrow \phi_n \Rightarrow x.\psi$ is true under \mathcal{M} . Let s_0 be the initial state of \mathcal{M} , and let

$$s_0 \rightarrow_{\square} s_1 \rightarrow_{\square} \dots \rightarrow_{\square} s_n.$$

Assume that the interpretation $\mathcal{M}[s_0 := s_i] \models \phi_i$ for all $1 \leq i < n$, and show that $x.\psi$ is true under $\mathcal{M}[s_0 := s_n]$. Since $\phi_1 \Rightarrow \dots \Rightarrow \phi_n \Rightarrow \psi$ is valid, it is in particular true under $\mathcal{M}[[x] := |s_n|]$. By (†), we infer from $\mathcal{M}[s_0 := s_i] \models \phi_i$ and $x \notin \phi_i$ that

$$\mathcal{M}[s_0 := s_i, [x] := |s_n|] \models \phi_i$$

for all $1 \leq i < n$. The desired conclusion $\mathcal{M}[s_0 := s_n, [x] := |s_n|] \models \psi$ follows. ■

5.1.3 Completeness

Our completeness proof is typical for quantified modal logics, and combines techniques from both propositional modal logic and classical first-order logic. The organization of the proof follows largely similar proofs that are surveyed by Garson [43].

- As is common in propositional modal logic, we construct, for any consistent formula, a model whose states are maximally consistent sets. The basic idea of the construction of this “canonical” model is to guarantee that all formulas that are contained in a state, are true in that state.
- As usual in Henkin-type proofs of the completeness of classical first-order logic, we take as the set of values the equivalence classes of terms under equality (which requires the rigidity of terms) [55].

A completeness condition has to be put on the maximally consistent sets to assure that all of them have values associated with them. So in order to give the canonical-model construction, we have to develop the notions of consistency and completeness of sets of formulas first.

Consistent sets

Given a normal logic Λ , we write $\Phi \vdash_{\Lambda} \phi$ iff

$$\phi_1 \rightarrow \cdots \rightarrow \phi_n \rightarrow \phi \in \Lambda$$

for some finite subset $\{\phi_i \mid 1 \leq i \leq n\}$ of Φ , and $\vdash_{\Lambda} \phi$ iff $\phi \in \Lambda$. Throughout this subsection, we assume $\Lambda \neq \perp$ to be fixed, and suppress the subscript of the derivability relation \vdash . The concept of consistency is the familiar one. A set Φ of formulas is called *consistent* iff $\Phi \not\vdash \text{false}$; it is *maximally consistent* iff, furthermore, either $\phi \in \Phi$ or $\neg\phi \in \Phi$ for all formulas ϕ . The following lemma is, as usual, established using PROP.

Lemma 5.3 (Consistency) *For every set Φ of formulas and formula ϕ :*

- (1) $\Phi \cup \{\neg\phi\}$ is consistent iff $\Phi \not\vdash \phi$.
- (2) If Φ is consistent, then either $\Phi \cup \{\phi\}$ or $\Phi \cup \{\neg\phi\}$ is consistent.
- (3) For any maximally consistent set Φ , $\phi_1 \rightarrow \phi_2 \in \Phi$ iff $\phi_1 \notin \Phi$ or $\phi_2 \in \Phi$.

Complete sets

By $\phi - x$ we denote a set of formulas that result from binding all free occurrences of x in ϕ by a single quantifier. More precisely, $\phi - x$ is the smallest set satisfying the following condition: if ϕ is of the form

$$\phi_1 \Rightarrow \cdots \Rightarrow \phi_n \Rightarrow \psi,$$

where $x \notin \phi_i$ for all $1 \leq i \leq n$, then

$$\phi_1 \Rightarrow \cdots \Rightarrow \phi_n \Rightarrow x.\psi \in \phi - x.$$

Note that, in particular, $x.\phi \in \phi - x$.

A set Φ of formulas is *complete* iff $\Phi \vdash \phi[x := \pi]$ for all terms π implies $\Phi \vdash \phi'$ for all $\phi' \in \phi - x$. By part (1) of Lemma 5.3, it follows that completeness is equivalent to the condition that, if $\Phi \cup \{\neg\phi'\}$ is consistent and $\phi' \in \phi - x$ for some formula ϕ and variable x , then there is a term π such that $\Phi \cup \{\neg\phi[x := \pi]\}$ is consistent. In particular, the completeness of a set Φ ensures that, whenever Φ contains the formulas $\phi[x := \pi]$ for all terms π , then Φ entails $x.\phi$. This means, intuitively speaking, that some term π is

interpreted as the value of the current state. The general form of the completeness condition is necessary to guarantee this property, which allows us to assign terms as values to complete sets, for all states in the canonical model.

Saturated sets

Maximally consistent, complete sets of formulas are called *saturated*. Given a consistent formula ϕ_0 , we will define a model for ϕ_0 whose states are saturated sets. Part (1) of the following lemma ensures that ϕ_0 is contained in some saturated set, while part (2) guarantees that there are enough such sets for constructing a model. Note that the general form of the quantifier introduction rule Q2 is needed to show the former.

Lemma 5.4 (Existence of saturated extensions)

- (1) *Every finite consistent set of formulas has a saturated extension.*
- (2) *Every complete consistent set of formulas has a saturated extension.*

Proof of Lemma 5.4 Both parts are shown by applying variants of the Lindenbaum procedure to extend a consistent set Φ_0 to a maximally consistent set Φ . Enumerate all formulas ϕ_1, ϕ_2, \dots , and let Φ_{i+1} , for all $i \geq 0$, be either $\Phi \cup \{\phi_i\}$, if this set is still consistent, or $\Phi \cup \{\neg\phi_i\}$, otherwise. Each set Φ_i is consistent by part (2) of Lemma 5.3, implying that

$$\Phi = \bigcup_{i \geq 0} \Phi_i$$

is maximally consistent.

(1) Suppose that Φ_0 is finite. Then we can guarantee the completeness of Φ as follows: whenever Φ_i is extended by a formula $\neg\phi'$ and $\phi' \in \phi - x$ for some formula ϕ and variable x , then we add also $\neg\phi[x := y]$, for some new variable $y \notin \Phi_i \cup \{\phi\}$. Although at every stage several new formulas may be added, each set Φ_i is finite, which assures the existence of new variables at all future stages. It remains to be shown that this process preserves consistency. Assume that $\Phi \cup \{\neg\phi', \neg\phi[x := y]\}$ is inconsistent; that is,

$$\vdash \Phi_j \rightarrow \neg\phi' \rightarrow \neg\phi[x := y] \rightarrow \text{false} \quad (\dagger)$$

for some conjunction Φ_j of formulas in Φ . We derive a contradiction, by showing that, in this case, already

$$\vdash \Phi_j \rightarrow \neg\phi' \rightarrow \text{false}, \quad (\ddagger)$$

implying the inconsistency of $\Phi \cup \{\neg\phi'\}$. Let ϕ' be $\phi_1 \Rightarrow \dots \Rightarrow \phi_n \Rightarrow x.\psi$, with $x \notin \phi_i$ for all $1 \leq i \leq n$. Use Q to infer from (†) that

$$\vdash \Phi_f \rightarrow \neg\phi' \rightarrow \phi_1 \Rightarrow \dots \Rightarrow \phi_n \Rightarrow y.\psi[x := y],$$

and conclude (†) by VAR2 and PROP.

(2) First we show that, if a set Φ is complete, then so is $\Phi \cup \{\phi\}$. Assume that

$$\Phi \cup \{\phi\} \vdash \varphi[x := \pi]$$

for all terms π , let x be such that $x \notin \phi$ and $\varphi' \in \varphi - x$; by VAR2 it suffices to show that $\Phi \cup \{\phi\} \vdash \varphi'$. From our assumption, it follows by PROP that

$$\Phi \vdash \phi \rightarrow \varphi[x := \pi]$$

for all terms π ; hence $\Phi \vdash \phi \rightarrow \varphi'$ by the completeness of Φ (and Q*, in case φ' equals $x.\varphi$). The desired conclusion follows.

Now suppose that Φ_0 is complete. Then all the finite extensions Φ_i , $i \geq 0$, are complete; that is, whenever Φ_i is about to be extended by the formula $\neg\phi'$ and $\phi' \in \phi - x$ for some formula ϕ , there exists a term π such that $\Phi_i \cup \{\neg\phi[x := \pi]\}$ is consistent. We extend Φ_i in this fashion, and continue the Lindenbaum process by checking whether $\neg\phi'$ can still be added consistently. Since every formula contains only a finite number of quantifiers, each stage is completed within a finite number of steps. ■

In the canonical model, the value associated with a state (saturated set) s containing $x.\tau = \pi$ will be the equivalence class of the term π under equality. The next lemma guarantees that such a term exists (part (4)), and that equality is a congruence relation (parts (1) to (3)).

Lemma 5.5 (Value of saturated sets) *For every saturated set s of formulas:*

- (1) $\approx_s = \{(\pi_1, \pi_2) \mid \pi_1 = \pi_2 \in s\}$ is an equivalence relation.
- (2) $\bar{\pi}_1 \approx_s \bar{\pi}_2$ implies that $f\bar{\pi}_1 \approx_s f\bar{\pi}_2$ for all $f \in F$.
- (3) $\bar{\pi}_1 \approx_s \bar{\pi}_2$ and $p\bar{\pi}_1 \in s$ implies $p\bar{\pi}_2 \in s$ for all $p \in R$.
- (4) $\{\pi \mid x.x = \pi \in s\} = [\pi]_{\approx_s}$ for some term π .

Proof of Lemma 5.5 Parts (1), (2), and (3) follow from EQ.

(4) QEQ and Q4 imply that $\neg y. \neg x. y = x \in s$. Since s is complete, $x. \pi_s = x \in s$ for some term π_s , not containing x ; thus $x. x = \pi_s \in s$ by EQ and Q^{*}. From EQ4:

$$x. x = \pi_1 \rightarrow (x. x = \pi_2 \leftrightarrow \pi_1 = \pi_2),$$

we infer that $[\pi_s]_{\approx_s} = \{\pi \mid x. x = \pi \in s\}$. ■

Canonical model

Now we are ready to define the *canonical model* for a given consistent formula ϕ_0 . Let

$$\mathcal{M}(\phi_0) = (\Sigma, \rightarrow_{\square}, \Upsilon, ||, [[x]]_{x \in V}, [[f]]_{f \in F}, [[p]]_{p \in R}, s_0)$$

be an interpretation such that:

- s_0 is a saturated extension of $\{\phi_0\}$. By \approx we denote \approx_{s_0} .
- Σ is the set of saturated sets s with $\approx_s = \approx$.
- $s \rightarrow_{\square} t$ iff $\phi \in t$ for all $\square \phi \in s$.
- Υ is the set of equivalence classes $[\pi]_{\approx}$.
- $|s| = \{\pi \mid x. x = \pi \in s\}$.
- $[[x]] = [x]_{\approx}$.
- $[[f]]([[\pi]_{\approx}]) = [f\pi]_{\approx}$.
- $[[\bar{\pi}]_{\approx} \in [[p]](s)$ iff $p\bar{\pi} \in s$.

Note that the interpretation $\mathcal{M}(\phi_0)$ is well-defined: a saturated extension of $\{\phi_0\}$ exists by part (1) of Lemma 5.4; Lemma 5.5 guarantees that \approx is an equivalence relation, that $|s| \in \Upsilon$, and that $[[f]]$ and $[[p]]$ are properly defined.

The values of the canonical model are equivalence classes of terms under equality. It is straightforward to show, by structural induction, that all terms are interpreted as themselves, modulo equality.

Lemma 5.6 (Term model) $[[\pi]] = [\pi]_{\approx}$.

The states of the canonical model are saturated sets. The following main theorem shows that every state s of $\mathcal{M}(\phi_0)$ contains precisely the formulas that are true at s . Since $\phi_0 \in s_0$, it follows immediately that the interpretation $\mathcal{M}(\phi_0)$ is a model of ϕ_0 .

Theorem 5.1 (Canonical model) *Let \mathcal{M} be a canonical model (of any formula) with the initial state s_0 . Then $\phi \in s$ iff $\mathcal{M}[s_0 := s] \models \phi$ for every state s of \mathcal{M} and every formula ϕ .*

Proof of Theorem 5.1 We apply induction on the structure of ϕ . The atomic cases follow from Lemma 5.6. The propositional cases are consequences of the induction hypothesis and part (3) of Lemma 5.3.

If $\Box\phi \in s$ and $s \rightarrow_{\Box} t$, then $\phi \in t$ and, by the induction hypothesis, $\mathcal{M}[s_0 := t] \models \phi$. Now assume that $\Box\phi \notin s$ — that is, $\neg\Box\phi \in s$ — and show that $\mathcal{M}[s_0 := s] \not\models \Box\phi$. This follows from Lemma 5.8 (see below) by the induction hypothesis.

For the quantifier case, choose a term π_s such that $x.x = \pi_s \in s$. Consequently, we have that $\mathcal{M}[s_0 := s] \models x.\phi$ iff $\mathcal{M}[s_0 := s, [x] := |s_0|] \models \phi$ iff, by Lemma 5.7 (given below), $\mathcal{M}[s_0 := s] \models \phi[x := \pi_s]$ iff, by the induction hypothesis, $\phi[x := \pi_s] \in s$. From EQ4:

$$x.x = \pi_s \rightarrow (x.\phi \leftrightarrow \phi[x := \pi_s]),$$

we conclude that $\mathcal{M}[s_0 := s] \models x.\phi$ iff $x.\phi \in s$. ■

In the proof of Theorem 5.1, we invoked the following two lemmas. To show that substitution behaves as expected, can be done by straightforward structural induction. To show that there are enough states in the canonical model, we use the fact that every complete consistent complete set has a saturated extension.

Lemma 5.7 (Substitution) $\mathcal{M}[[x] := [\pi]] \models \phi$ iff $\mathcal{M} \models \phi[x := \pi]$.

Lemma 5.8 (Succession) *If $s \in \Sigma$ and $\neg\Box\phi \in s$, then there is a $t \in \Sigma$ such that $s \rightarrow_{\Box} t$ and $\neg\phi \in t$.*

Proof of Lemma 5.8 Suppose that $s \in \Sigma$ and $\neg\Box\phi \in s$. Let

$$\Phi = \{\psi \mid \Box\psi \in s\} \cup \{\neg\phi\}.$$

We show that Φ is (1) consistent and (2) complete. Thus, a saturated extension t of Φ exists by part (2) of Lemma 5.4; furthermore, $t \in \Sigma$ because $\approx_t = \approx_s$ by RIG1-2, and $s \rightarrow_{\Box} t$ by the definition of \rightarrow_{\Box} .

(1) Suppose that Φ is inconsistent, that is,

$$\vdash \psi_1 \rightarrow \dots \rightarrow \psi_n \rightarrow \neg\phi \rightarrow \text{false}$$

for some $\Box\psi_i \in s$, $1 \leq i \leq n$. By K2 and repeated application of K1,

$$\Box(\neg\phi \rightarrow \text{false}) \in s,$$

implying that $\Box\phi \in s$ (use K). Since also $\neg\Box\phi \in s$, it follows that s is inconsistent, a contradiction.

(2) Assume that $\Phi \vdash \varphi[x := \pi]$ for all terms π , let x be such that $x \notin \phi$ and $\varphi' \in \varphi - x$; by VAR2 it suffices to show that $\Phi \vdash \varphi'$. Similarly to part (1), by K it follows that

$$\Box(\neg\phi \rightarrow \varphi[x := \pi]) \in s$$

for all π . Hence $\Box(\neg\phi \rightarrow \varphi') \in s$ by the completeness of s (if φ' equals $x.\varphi$, use Q⁻ and K), and $\neg\phi \rightarrow \varphi' \in \Phi$ by the definition of Φ . Since also $\neg\phi \in \Phi$, we conclude that $\Phi \vdash \varphi'$.

□

Thus we have shown that every consistent formula $\neg\phi$ is satisfiable by the corresponding canonical model $\mathcal{M}(\neg\phi)$. By part (1) of Lemma 5.3, it follows that $\vdash \phi$ for every valid formula ϕ . Recall that we have assumed an arbitrary normal logic Λ ; thus K contains precisely all valid formulas:

Corollary 5.1 (Completeness) *A formula ϕ of half-order modal logic is valid iff $\phi \in K$; that is, $\models \phi$ iff $\vdash_K \phi$.*

We remark that it is an interesting open question under which conditions the general form of the quantifier introduction rule Q2 can be replaced by the simpler rule Q2⁻. We conjecture that this is case for all normal logics Λ_Φ , such as K, whose set Φ of nonlogical axioms satisfies certain closure properties.

5.1.4 Syntactic and semantic extensions

In order to treat the real-time temporal logic TPTL as a half-order modal logic, we admit multiple modal operators and semantic restrictions on the corresponding accessibility relations. By axiomatizing accessibility relations that are equivalence relations, we show that half-order modal logic subsumes first-order classical logic.

Multimodal logics

Half-order versions of *multimodal* logics are straightforwardly defined by admitting several modal operators \Box_i in the syntax. An interpretation contains, accordingly, a separate accessibility relation $\rightarrow_{\Box_i} \subseteq \Sigma^2$ for each operator \Box_i .

Multimodal normal logics are closed under the axiom schemata $K1_i$, $RIG1_i$, and $RIG2_i$, as well as the inference rules $K2_i$: one for each modal operator \Box_i . As for the inference rule $Q2$, consider every formula $\phi \rightarrow \Box_i \psi$, for any operator \Box_i , to be of the form $\phi \Rightarrow \psi$. For example, if $x \in \phi_1, \phi_2$, then from

$$\phi_1 \rightarrow \Box_1 (\phi_2 \rightarrow \Box_2 \psi)$$

we may infer

$$\phi_1 \rightarrow \Box_1 (\phi_2 \rightarrow \Box_2 x. \psi).$$

The completeness proof given above is easily generalized to show that multimodal K (the smallest multimodal normal logic) contains precisely all valid formulas.

Accessibility conditions

It is often useful to consider only a certain class C of interpretations for half-order model logic. We say that a formula ϕ is *C-valid*, and write $C \models \phi$, iff ϕ is true under all interpretations in C ; we also write $\Phi \vdash \phi$ for $\phi \in \Lambda_{\Phi}$. A set Φ of nonlogical axioms is *sound* for the class C of interpretations iff

$$\Phi \vdash \phi \text{ implies } C \models \phi$$

for all formulas ϕ . This is the case if

1. $C \models \phi$ for all nonlogical axioms $\phi \in \Phi$, and
2. all inference rules are sound for C ; that is, if the antecedent of $K2$ or $Q2$ is C -valid, then so is the consequent.

The set Φ of nonlogical axioms characterizes the class C of interpretations *completely* iff

$$C \models \phi \text{ iff } \Phi \vdash \phi$$

for all formulas ϕ .

Here we restrict ourselves to classes C of interpretations whose accessibility relations \rightarrow_{\square} satisfies a certain condition C , and add nonlogical axioms to our proof system so that all C -valid formulas can be derived. For example, consider the class $REFL$ of interpretations with a reflexive accessibility relation. The reflexive interpretations $REFL$ clearly satisfy all formulas of the form

$$REFL \quad \Box\phi \rightarrow \phi.$$

In order to show that adding these formulas as nonlogical axioms does not exclude any reflexive interpretations, we have to prove that all inference rules are sound for $REFL$. An inspection of the proof of Lemma 5.2 reveals that the inference rules are sound for any class C that is determined solely by a condition on the accessibility relation. To show that the chosen nonlogical axioms are sufficient to characterize reflexive interpretations completely, simply observe that, with the axiom schema $REFL$, the accessibility relation of the canonical model is reflexive. It follows that

$$REFL \models \phi \text{ iff } REFL \vdash \phi$$

for all formulas ϕ . The other parts of the following lemma are established similarly. (In fact, the proofs are identical to the corresponding arguments for propositional modal logic, and can, for example, be found in [46].)

Lemma 5.9 (Accessibility conditions) *The following nonlogical axiom schemata characterize the corresponding conditions on the accessibility relation $\rightarrow_{\square} \subseteq \Sigma^2$ completely.*

- *Reflexivity:* $\Box\phi \rightarrow \phi$.
- *Symmetry:* $\Box\phi \rightarrow \Box\Diamond\phi$.
- *Transitivity:* $\Box\phi \rightarrow \Box\Box\phi$.
- *Seriality and functionality* (for all $s \in \Sigma$ there is a [unique] $t \in \Sigma$ such that $s \rightarrow_{\square} t$):

$$\Box\phi \rightarrow \Diamond\phi \text{ and } \Box\phi \leftrightarrow \Diamond\phi.$$

- *Weak connectivity* ($s \rightarrow_{\square} t$ and $s \rightarrow_{\square} t'$ implies that $t \rightarrow_{\square} t'$ or $t = t'$ or $t' \rightarrow_{\square} t$):

$$\Box((\phi_1 \wedge \Box\phi_1) \rightarrow \phi_2) \vee \Box((\phi_2 \wedge \Box\phi_2) \rightarrow \phi_1).$$

Embedding classical logic

We have seen that half-order modal logic corresponds to a *fragment* of first-order modal logic, because the freeze quantifier can be expressed by conventional quantification in combination with a state variable. Alternatively, half-order modal logic can be viewed as a *generalization* of first-order classical logic. By showing how to embed classical first-order logic faithfully into a half-order normal logic, we prove the undecidability of the latter one.

Suppose that all predicates are rigid. Then we can read the combination " $\Box z$." of a modal operator and the freeze quantifier as a universal quantifier with restricted scope: it ranges only over the values of adjacent states. Similarly, " $\Diamond z$." can be viewed as a local existential quantifier. By pursuing this idea, we see that ordinary quantifiers are representable in a half-order modal logic with a universal accessibility relation, that is, in whose models every state is accessible from every other state.

Let RIGID-S5 be the smallest normal logic containing the following nonlogical axioms:

- SYMM** $\phi \rightarrow \Box \Diamond \phi$,
TRANS $\Box \phi \rightarrow \Box \Box \phi$,
RIG3 $p\bar{x} \rightarrow \Box p\bar{x}$ for all $p \in R$,
EX $\Diamond z.z = \pi$.

Note that in the presence of EX, which implies \Diamond true, the schema

$$\text{REFL } \Box \phi \rightarrow \phi$$

is derivable from SYMM and TRANS, while SYMM and RIG4 imply

$$\text{RIG4 } \neg p\bar{x} \rightarrow \Box \neg p\bar{x} \text{ for all } p \in R.$$

Together, the schemata REFL, SYMM, and TRANS characterize, as for the propositional modal logic S5, accessibility relations that are equivalence relations. The axiom schema EX assures that enough states are accessible (i.e., in the same equivalence class as the initial state). RIG3 and RIG4 assert that all relation symbols are rigid.

Let ϕ be a classical formula over the first-order language (F, R) , and let ϕ^\square the formula of half-order modal logic that results from replacing all quantifiers $\forall z$. and $\exists z$. by $\Box z$. and $\Diamond z$., respectively. The following theorem states that this translation preserves validity.

Proposition 5.1 (Embedding of classical logic) *A formula ϕ of first-order classical logic is valid iff the formula ϕ^\square of half-order modal logic is contained in RIGID-S5.*

Proof of Proposition 5.1 (1) First we show that

$$\phi^\square \rightarrow \square\phi^\square \in \text{RIGID-S5} \quad (\dagger)$$

for any classical first-order formula ϕ . We proceed by induction on the structure of ϕ , assuming that all negations in ϕ have been pushed inside in front of atomic formulas. If ϕ is an atomic formula or its negation, use one of RIG1-4. The propositional cases follow from the induction hypothesis by K. If ϕ is of the form $\forall x. \psi$, then

$$\square x. \psi^\square \rightarrow \square \square x. \psi^\square$$

holds by TRANS. Finally, suppose that the outermost symbol of ϕ is an existential quantifier. In this case the inductive step is an instance of $\diamond\phi \rightarrow \square\diamond\phi$, which follows from SYMM and TRANS by K.

Now assume that ϕ is provable by a complete Hilbert-style proof system for the first-order predicate calculus, say the one given in the textbook by Enderton [38]. Any classical deduction of ϕ (in the given proof system) can be transformed into a half-order modal derivation of ϕ^\square , thus implying that $\phi^\square \in \text{RIGID-S5}$. The only interesting case is the derivation of the translation of the classical quantifier axiom $\psi[x := \pi] \rightarrow \exists x. \psi$ in RIGID-S5: from EQ4, by K infer

$$\diamond x. x = \pi \rightarrow \square\psi^\square[x := \pi] \rightarrow \diamond x. \psi^\square,$$

which implies $\psi^\square[x := \pi] \rightarrow \diamond x. \psi^\square$ by EX and (\dagger).

(2) The second direction of the theorem is shown by semantic reasoning. Assume that $\phi^\square \in \text{RIGID-S5}$; we show that ϕ is true under an arbitrary classical first-order interpretation I . Define the class \mathcal{C}_I of interpretations for half-order modal logic to contain all interpretations

$$\mathcal{M}_I = (\Sigma, \rightarrow, \square, \top, \perp, [x]_I, [f]_I, [p]_I, \ast_0)$$

such that

- both Σ and \top are the universe of I ,

- $\rightarrow_{\square} = \Sigma^2$ is universal,
- $||$ is the identity (i.e., $|s| = s$), and
- $[\bar{x}] \in [p](s)$ iff $[\bar{x}] \in [p]_I$,

where $[x]_I$, $[f]_I$, and $[p]_I$ are the assignment functions of I for variables, function symbols, and relation symbols, respectively. Since any state of \mathcal{M}_I can be taken to be initial, we obtain a set C_I of interpretations. It is not hard to see all interpretations in C_I satisfy the nonlogical axioms of RIGID-S5, and that all inference rules are sound for the class C_I . Furthermore,

$$C_I \models \phi^{\square} \text{ iff } I \models \phi.$$

The theorem follows. \square

Since classical first-order logic is undecidable, so is RIGID-S5. This shows that axiomatizable half-order normal logics are not necessarily decidable. On the other hand, it is not hard to see that every satisfiable formula ϕ of half-order logic is satisfiable under an interpretation that contains only a bounded finite number of states and a finite number of values (at most one for every state and one for every term in ϕ). It follows that K itself is decidable in the half-order case.

5.2 Half-order Temporal Logic

In this subsection, we study half-order extensions of the linear propositional temporal logic PTL. The semantics of half-order *temporal* logic is restricted to interpretations with a state structure that is isomorphic to the natural numbers \mathbb{N} ; these “temporal” interpretations are essentially infinite sequences of states. The syntax of half-order temporal logic contains two modal operators: the *next* operator \bigcirc , which is interpreted as “at the immediate successor state,” and the *always* operator \square meaning “at all successor states.”

We show that, unlike in the propositional case, there cannot exist a complete half-order proof system for temporal structures in general. Yet we introduced a decidable half-order temporal logic, TPTL, for the specification and verification of real-time systems. The decidability of TPTL is due to a careful choice of both the set of values (monotonically increasing natural numbers) and the corresponding operations (the zero and successor functions, and the order relation on \mathbb{N}). Here we extend our proof system for half-order K to obtain a complete proof system for TPTL.

5.2.1 Syntax and semantics

The formulas of *timed temporal logic* (TPTL) are the formulas of half-order modal logic with the two modal operators \bigcirc and \square , where

- the set F of function symbols contains only the constant symbol 0 and the unary function symbol S (as in *Successor*), and
- the set R of relation symbols contains only propositions P (i.e., relation symbols that take no arguments) and the binary relation symbol $<$ (recall that equality is included in all half-order modal logics).

Abbreviations such as \leq and $+5$ are defined as usual. We omit congruence relations to simplify our discussion. The *until* operator \mathcal{U} will be added later. While all formulas of TPTL, as defined in Chapter 3, used to be closed, we admit now free variables and interpret them universally over the time domain \mathbb{N} (as “parameters”).

TPTL is interpreted over *digitally timed state sequences*, which are temporal structures whose values are monotonically increasing natural numbers. More precisely, a (digitally) *timed state sequence* is an interpretation

$$\mathcal{M} = (\Sigma, \rightarrow_{\bigcirc}, \rightarrow_{\square}, \mathbb{N}, ||, [x]_{x \in V}, [f]_{f \in F}, [p]_{p \in P}, \sigma_0)$$

for half-order modal logic such that

- \rightarrow_{\bigcirc} imposes a linear order

$$\sigma_0 \rightarrow_{\bigcirc} \sigma_1 \rightarrow_{\bigcirc} \sigma_2 \rightarrow_{\bigcirc} \dots$$

on the set $\Sigma = \{\sigma_i \mid i \geq 0\}$ of states,

- \rightarrow_{\square} is the reflexive transitive closure of \rightarrow_{\bigcirc} ,
- $|\sigma_i| \leq |\sigma_{i+1}|$ for all $i \geq 0$,
- for all $n \in \mathbb{N}$, there is some $i \geq 0$ such that $|\sigma_i| \geq n$,
- 0 denotes zero (i.e., $[0] = 0$),
- S denotes the successor function on \mathbb{N} (i.e., $[S](n) = n + 1$), and
- $<$ rigidly denotes the order relation on \mathbb{N} (i.e., $[<](\sigma, m, n)$ iff $m < n$).

We denote the set of timed state sequences by $TSS_{\mathbb{N}}^w$ or simply TSS^w .

The correspondence with the previous definition of timed state sequences is obvious: any timed state sequence can be viewed as an infinite sequence of states $\sigma_i \subseteq P$, for $i \geq 0$, together with an assignment function for the variables. Each of the states σ_i specifies the propositions that are true in that state (let $p \in \sigma_i$ iff $\sigma_i \in \llbracket p \rrbracket$), and has a value $|\sigma_i|$ associated with it. The values satisfy the *monotonicity* condition that, for all $i \geq 0$,

$$|\sigma_i| \leq |\sigma_{i+1}|,$$

and the *progress* condition that, for all $n \in \mathbb{N}$, there is some $i \geq 0$ such that

$$|\sigma_i| \geq n.$$

Both conditions are motivated by the original design of TPTL as a real-time logic: the values that are associated with the states can be interpreted as time-stamps; think of state σ as representing the state of a system at time $|\sigma|$. From this point of view, the freeze quantifier “ x .” binds the associated variable x to the “current” time.

Recall that, even though time is discrete, the notion of “next time” is entirely independent of “next state”; successive states may have the same or vastly different times associated with them, as long as the time does not decrease. If desired, the requirement that the time increases always by 1 between successive states, and thus acts as a state counter, can be expressed within TPTL, by the formula

$$\Box x. \bigcirc y. y = x + 1. \quad (\phi_{+1})$$

5.2.2 Proof system

We extend the proof system for half-order K by axioms for timed state sequences. The following three axiom schemata completely characterize temporal structures in propositional temporal logic [40]:

$$\text{LIN1} \quad \bigcirc \neg \phi \leftrightarrow \neg \bigcirc \phi,$$

$$\text{LIN2} \quad \Box \phi \rightarrow \phi \wedge \bigcirc \Box \phi,$$

$$\text{LIN3} \quad \phi \rightarrow \Box(\phi \rightarrow \bigcirc \phi) \rightarrow \Box \phi.$$

Note that LIN1 asserts the functionality of $\rightarrow \circ$, and that LIN2 immediately implies REFL: $\Box \phi \rightarrow \phi$. LIN3 gives an induction principle.

In addition, we need a set of axioms that allows us to derive all universal sentences of the decidable classical first-order theory of $(N, 0, S, \leq)$. For instance, the following group of NAT axioms from the textbook by Enderton has this completeness property, as can be shown by a quantifier elimination procedure [38]:

- NAT1 $x < Sy \leftrightarrow x \leq y$,
 NAT2 $x < y \vee x = y \vee y < x$,
 NAT3 $x < y \rightarrow y \not< x$,
 NAT4 $x < y \wedge y < z \rightarrow x < z$,
 NAT5 $x \not< 0$.

The axiom MON states that the time is monotonically increasing from state to state; the axiom schema PRO asserts that time diverges and assures that all free variables are interpreted as finite natural numbers:

- MON $x. \circ y. x \leq y$,
 PRO $\Diamond x. x \geq y$,
 RIG5 $\pi_1 < \pi_2 \rightarrow \circ \pi_1 < \pi_2$.

RIG5 is sufficient to guarantee the rigidity of $<$ (see the upcoming lemma on sample theorems).

Let TPTL be the set of formulas consisting of LIN1-3, NAT1-5, MON, PRO, and RIG5. We show that TPTL is the smallest normal logic that contains the nonlogical axioms TPTL; that is,

$$TSS^w \models \phi \text{ iff } TPTL \vdash \phi$$

for every formula ϕ of TPTL. It is straightforward to convince yourself that every nonlogical axiom is TSS^w -valid (i.e., true in all timed state sequences). Moreover, as timed state sequences are closed under suffixes, all inference rules are sound for TSS^w . Before proving completeness, we derive some additional formulas that will be useful later.

Sample proofs

The following theorems can be proved purely propositionally (see, for example, [46] for the derivations):

- TRANS $\Box\phi \rightarrow \Box\Box\phi,$
 LIN4 $\Box\phi \leftrightarrow \phi \wedge \bigcirc\Box\phi,$
 LIN5 $\Box(\Box\phi_1 \rightarrow \phi_2) \vee \Box(\Box\phi_2 \rightarrow \phi_1),$
 LIN6 $\Box(\Box(\phi \rightarrow \Box\phi) \rightarrow \phi) \rightarrow \bigcirc\Box\phi \rightarrow \Box\phi.$

Lemma 5.10 (Sample theorems of TPTL) *Any normal logic that is closed under the axioms TPTL contains the following formulas of TPTL:*

- MON' $x.\Box y. x \leq y.$
 RIG6 $\pi_1 \not\prec \pi_2 \rightarrow \bigcirc\pi_1 \not\prec \pi_2.$
 RIG7 $\pi_1 < \pi_2 \rightarrow \Box\pi_1 < \pi_2.$
 RIG8 $\pi_1 \not\prec \pi_2 \rightarrow \Box\pi_1 \not\prec \pi_2.$
 TSS1 $\Box x. \bigcirc y. y = x \rightarrow x.\Box y. y = x.$
 TSS2 $x.\Box y. y = x \rightarrow (x.\Box\phi \leftrightarrow \Box x.\phi).$

Proof of Lemma 5.10 MON' generalizes the monotonicity axiom MON. Its proof demonstrates the application of the induction schema LIN3. By applying Q^{*} to LIN3, it suffices to derive the "base case"

$$x.y.x \leq y$$

and the "inductive step"

$$x.\Box(y.x \leq y \rightarrow \bigcirc y.x \leq y).$$

The base case follows from QEQ by Q^{*} (let us begin to suppress to mention applications of the equality axioms EQ). To show the inductive step, by Q2^{*} and K2_□ it suffices to derive

$$y.x \leq y \rightarrow \bigcirc y.x \leq y.$$

From NAT:

$$x \leq y \rightarrow y \leq z \rightarrow x \leq z.$$

By Q^{*} and K_○:

$$\bigcirc x \leq y \rightarrow \bigcirc x.y \leq z \rightarrow \bigcirc x.z \leq z.$$

By RIG1-2_○, RIG5, and Q⁻:

$$y.x \leq y \rightarrow y. \bigcirc z.y \leq z \rightarrow \bigcirc z.x \leq z.$$

The desired conclusion follows by MCN, REFL, and VAR2.

RIG6 can be inferred from NAT:

$$\pi_1 \prec \pi_2 \leftrightarrow (\pi_1 = \pi_2 \vee \pi_2 < \pi_1)$$

by RIG1_○, RIG5, and K_○. RIG7 (and RIG8) follow from LIN3 by RIG5 (RIG6, respectively) and K2_○. Note that the rigidity axioms RIG1-2_○ are similarly derivable from RIG1-2_□, and thus can be omitted.

TSS1 is again derived by induction. By applying Q⁻ to LIN3, it suffices to derive the base case $x.y.y = x$, which holds by QEQ, and the inductive step

$$\Box x. \bigcirc y.y = x \rightarrow x. \Box (y.y = x \rightarrow \bigcirc y.y = x).$$

Using Q⁻, K_□, and VAR2, it suffices to show that

$$y. \bigcirc z.z = y \rightarrow y.y = x \rightarrow \bigcirc z.z = x,$$

which follows from EQ4 by Q⁻.

TSS2 follows from EQ4 by K_□ and Q⁻. ■

Updating time references

Let $Next_\delta$ and $Next_{>\Delta}$ be abbreviations for the two formulas

$$x. \bigcirc y.y = x + \delta,$$

$$x. \bigcirc y.y > x + \Delta,$$

respectively, which assert that the time difference between the current state and its successor state is exactly $\delta \in \mathbb{N}$ (greater than $\Delta \in \mathbb{N}$, respectively). These time-difference formulas will be used to update, in TPTL-formulas, references to the times of previous states. For example, the formula $x. \Box \psi$ holds in a state that contains $Next_\delta$ iff $x. \psi$ is true in that state and, intuitively speaking, " $x. \Box \psi[x := x - \delta]$ " is true in its successor state. If δ is

greater than the number of successor symbols occurring in ψ and $x.\psi$ is closed, then the monotonicity of time can be exploited to simplify, to *true* or *false*, all timing constraints in " $x.\Box\psi[x := x - \delta]$ " that refer to the current time x .

Let $x.\phi$ be a closed formula of TPTL. As in Chapter 4, we write $x.\phi^\delta$ for the TPTL-formula that expresses the condition " $x.\phi[x := x - \delta]$ "; that is, $x.\phi^\delta$ results from $x.\phi$ by updating all references of ϕ to the current time x by the time difference δ . From now on, whenever we write $x.\phi^\delta$, there is an implicit condition that the formula $x.\phi$ is closed. According to Lemma 4.1, we have that

$$\mathcal{M}[x := |\sigma_0| - \delta] \models \phi \quad \text{iff} \quad \mathcal{M} \models x.\phi^\delta$$

for every time difference $\delta \in \mathbb{N}$ and timed state sequence \mathcal{M} with $|\sigma_0| \geq \delta$. On the syntactic side, we have to make sure that our proof system is strong enough to derive the following facts about the updating of time references.

Lemma 5.11 (More theorems of TPTL) *Let Δ be the number of successor symbols in the formula ϕ of TPTL. Any normal logic that is closed under the axioms TPTL contains the following formulas (let x be different from y):*

- UPD1 $y.y = x + \delta \rightarrow (\phi[y := x] \leftrightarrow y.\phi^\delta).$
- UPD2 $y.y > x + \Delta \rightarrow (\phi[y := x] \leftrightarrow y.\phi^\Delta).$
- UPD3 $Next_\delta \rightarrow (x.\Box\phi \leftrightarrow \Box x.\phi^\delta).$
- UPD4 $Next_{>\Delta} \rightarrow (x.\Box\phi \leftrightarrow \Box x.\phi^\Delta).$
- CLOCK $(\bigvee_{0 \leq \delta \leq \Delta} Next_\delta) \vee Next_{>\Delta}.$

Proof of Lemma 5.11 UPD1 and UPD2 constitute the syntactic counterpart to the *time-step* lemma, Lemma 4.1; they capture the essence of the definition of $x.\phi^\delta$. The proofs of UPD1 and UPD2 proceed by induction on the structure of ϕ . Consider UPD1. By Q⁻ it suffices to derive

$$y = x + \delta \rightarrow (\phi[y := x] \leftrightarrow \phi^\delta).$$

The base cases follow from NAT. We present only the inductive step that introduces a new quantifier. By the condition of safe substitutivity, in

$$y = x + \delta \rightarrow ((z.\phi)[y := x] \leftrightarrow (z.\phi)^\delta)$$

the variable z has to be different from both x and y ; hence derive

$$y = x + \delta \rightarrow (z. \phi[y := x] \leftrightarrow z. \phi^\delta),$$

which follows from the induction hypothesis by Q^* .

UPD3 follows from UPD1 by K_\circ , Q^* , and VAR2 (rename y so that it does not occur in ϕ). UPD4 follows similarly from UPD2.

CLOCK is, in fact, derivable for any constant $\Delta \geq 0$ and *exclusive-or* connectives, implying that every state contains a unique time-difference formula. From NAT:

$$x \leq y \rightarrow ((\bigvee_{0 \leq \delta \leq \Delta} y = x + \delta) \vee y > x + \Delta),$$

infer

$$x. \circ y. x \leq y \rightarrow \text{CLOCK}$$

by Q^* , K_\circ , and LIN1. CLOCK follows by MON and REFL. ■

5.2.3 Completeness

We show that the proof system for half-order K together with the nonlogical axioms TPTL is complete for TPTL. The organization of the proof follows largely the completeness proof of Gabbay, Pnueli, Shelah, and Stavi for the propositional case, PTL, as presented by Goldblatt [46]. We proceed in two main steps:

Filtration Given a consistent TPTL-formula ϕ_0 , the canonical-model construction does not directly provide a timed state sequence, because \rightarrow_\square is not the transitive closure of \rightarrow_\circ . In order to have the induction axiom LIN3 force \rightarrow_\square to be the transitive closure of \rightarrow_\circ , we have to be able to characterize, by a TPTL-formula, all states that are reachable from the starting state by repeated traversal of \rightarrow_\circ . This can be achieved by collapsing the states of the canonical model into a *finite* number of *finitely* representable states (i.e., finite, consistent sets of formulas). Our filtration process is derived from the tableau-decision procedure for TPTL.

Unrolling While the structure $\mathcal{M}^F(\phi_0)$ that is obtained by filtration of the canonical model $\mathcal{M}(\phi_0)$ satisfies the desired transitive-closure property, it is still not a time! state sequence. The problem is that a state may have multiple successor states. Thus,

in a second step, we unroll the states of $\mathcal{M}^F(\phi_0)$ into a timed state sequence $\mathcal{M}^T(\phi_0)$. If the unrolling is done carefully, in a way that preserves the truth of all eventualities, then the resulting “canonical” timed state sequence $\mathcal{M}^T(\phi_0)$ is, at last, a model for ϕ_0 .

Let us assume that ϕ_0 does not any contain timing assertions that compare a variable with a natural number, and that ϕ_0 is closed. Absolute time references (such as $x = 5$ or $x + 1 > 3$) and free variables (“parameters”) have to be treated with some care; we delay their discussion until later. We also assume that all bound variables in ϕ_0 are distinct; this can always be achieved by renaming. Let

$$\mathcal{M}(\phi_0) = (\Sigma, \rightarrow, \circ, \square, \Upsilon, \parallel, [\mathbf{x}]_{\mathbf{x} \in V}, [f]_{f \in F}, [p]_{p \in R}, s_0)$$

be the canonical model for ϕ_0 , as constructed in the previous section. First, we remark that we may forget about the actual values (times) that are associated with the states in $\mathcal{M}(\phi_0)$, because the formulas $Next_\delta$ keep track of the time differences between adjacent states. In any timed state sequence, we can reconstruct the times from these time-difference formulas, modulo the initial time.

Filtration

Let Δ be the number of successor symbols S occurring in ϕ_0 . The key observation underlying the filtration is that we can restrict our attention to a *finite* number of time-difference formulas, namely $Next_\delta$ for all $0 \leq \delta \leq \Delta$ and $Next_{>\Delta}$. This is because if the time difference between two states is larger than Δ , its actual value has no bearing on the truth of ϕ_0 ; thus every model of ϕ_0 can be compressed into a model all of whose time steps are at most $\Delta + 1$ (simply reduce larger time steps to $\Delta + 1$). It follows that the truth of any TPTL-formula is determined by the truth of *finitely* many “subformulas.” The closure $Closure(\phi_0)$ of ϕ_0 under subformulas is defined essentially as in Chapter 4, as the smallest (i.e., finite) set containing ϕ_0 for some $z \notin \phi_0$ that is closed under the following operation *Sub*:

$$\begin{aligned} Sub(z.(\phi_1 \rightarrow \phi_2)) &= \{z.\phi_1, z.\phi_2\}, \\ Sub(z.\circ\phi) &= \{z.\phi^\delta \mid 0 \leq \delta \leq \Delta + 1\}, \\ Sub(z.\square\phi) &= \{z.\phi, z.\circ\square\phi\}, \\ Sub(z.x.\phi) &= \{z.\phi[x := z]\}. \end{aligned}$$

Let the finite filtration set Γ contain all formulas in $\text{Closure}(\phi_0)$ as well as the time-difference formulas Next_δ , for $0 \leq \delta \leq \Delta$, and $\text{Next}_{>\Delta}$. Note that the outermost symbol of every formula in Γ is a quantifier.

For $s, s' \in \Sigma$, let $s \sim_F s'$ iff

$$s \cap \Gamma = s' \cap \Gamma,$$

and $s^F = \{s' \mid s' \sim_F s\}$. We overload the symbol \in by writing $\phi \in s^F$ iff $\phi \in s'$ for all $s' \sim_F s$ (observe that \sim_F is an equivalence relation). Let

$$\mathcal{M}^F(\phi_0) = (\Sigma^F, \rightarrow_{\bigcirc}^F, \rightarrow_{\square}^F, \llbracket p \rrbracket_{p \in R}^F, s_0^F)$$

be the state structure that results from the canonical model $\mathcal{M}(\phi_0)$ by ignoring the values and identifying all states that agree on Γ :

- $\Sigma^F = \{s^F \mid s \in \Sigma \text{ and } s_0 \rightarrow_{\square} s\}$ (the reachable consistent subsets of Γ),
- $s^F \rightarrow_{\bigcirc}^F t^F$ iff $s' \rightarrow_{\bigcirc} t'$ for some $s' \sim_F s$ and some $t' \sim_F t$,
- $s^F \rightarrow_{\square}^F t^F$ iff $s \rightarrow_{\square}^F t$ for some $n \geq 0$ (the reflexive transitive closure of \rightarrow_{\bigcirc}^F),
- $\llbracket p \rrbracket^F(s^F)$ iff $p \in s'$ for all $s' \sim_F s$.

Note that there are only finitely many states s^F , each of which can be uniquely identified by a characteristic formula $\phi(s) \in s$, which is a finite conjunction of formulas and negated formulas from Γ :

$$\bigwedge_{\phi \in \Gamma \cap s} \phi \wedge \bigwedge_{\phi \in \Gamma - s} \neg \phi.$$

Let $s^F \rightarrow_{\bigcirc}^F t^F$ iff for all $s' \sim_F s$, there is some $t' \sim_F t$ such that $s' \rightarrow_{\square} t'$. The following lemma is proved similarly to the propositional case [46].

Lemma 5.12 (Filtration)

- (1) \rightarrow_{\bigcirc}^F is serial.
- (2) $s \rightarrow_{\square} t$ implies $s^F \rightarrow_{\square}^F t^F$.
- (3) \rightarrow_{\square}^F is reflexive, transitive, and connected (i.e., $s^F \rightarrow_{\square}^F t^F$ or $t^F \rightarrow_{\square}^F s^F$).
- (4) \rightarrow_{\bigcirc}^F is reflexive, transitive, connected, and $\rightarrow_{\bigcirc}^F \subseteq \rightarrow_{\square}^F$.

Proof of Lemma 5.12 (1) LIN1 ensures the functionality of \rightarrow_{\circ} by Lemma 5.9; this implies the seriality of \rightarrow_{\circ}^F . We remark that \rightarrow_{\circ}^F is, however, not functional.

(2) Suppose that $s \rightarrow_{\square} t$. Let ϕ be the finite disjunction of all characteristic formulas $\phi(r)$ with $s^F \rightarrow_{\square}^F r^F$; clearly $\phi \in r$ iff $s^F \rightarrow_{\square}^F r^F$. Use the induction schema LIN3 to show that $\square\phi \in s$, which implies that $\phi \in t$.

(3) Reflexivity and transitivity hold by definition. Lemma 5.9 implies that the relation \rightarrow_{\square} is reflexive because of REFL, transitive because of TRANS, and weakly connected because of REFL and LIN5; hence it is connected on all states reachable from s_0 by \rightarrow_{\square} . The connectivity of \rightarrow_{\square}^F follows by part (2).

(4) The reflexivity, transitivity, and connectivity of \rightarrow_{\diamond}^F follow from the corresponding properties of \rightarrow_{\square} (see part (3)). Part (2) implies that $\rightarrow_{\diamond}^F \subseteq \rightarrow_{\square}^F$. ■

Unrolling

Lemma 5.12 implies that $\mathcal{M}^F(\phi_0)$ consists of a finite sequence of strongly connected \rightarrow_{\square}^F -components, each one of which consists of a finite sequence of strongly connected \rightarrow_{\diamond}^F -components. We will construct a temporal model for ϕ_0 by unrolling $\mathcal{M}^F(\phi_0)$ into an infinite sequence of states. This has to be in a way such that, whenever some state contains an eventuality $z \diamond\phi$, then it is satisfied in a state that is unrolled "later." The following lemma guarantees this property for the unrolling that maintains the order of strongly connected \rightarrow_{\diamond}^F -components, and repeats all states in the final \rightarrow_{\square}^F -component infinitely often.

Lemma 5.13 (Unrolling) *Assume that $\neg\phi$ is equivalent to a formula in Γ . Let $s \in \Sigma$ be such that $t^F \not\rightarrow_{\diamond}^F s^F$ for some t . If $\square\phi \notin s$, then either $\phi \notin s$, or $\phi \notin t$ for some t such that $s^F \rightarrow_{\square}^F t^F$ and $t^F \not\rightarrow_{\diamond}^F s^F$.*

Proof of Lemma 5.13 Let $\neg\phi$ be equivalent to a formula in Γ ; then, for all s, t with $s \sim_F t$, $\phi \in s$ iff $\phi \in t$ and, by K_{\square} , $\square\phi \in s$ iff $\square\phi \in t$. The proof proceeds as in the propositional case, using Lemma 5.12 and LIN6 [46]. ■

Lemma 5.13 allows us to unroll $\mathcal{M}^F(\phi_0)$ in the described fashion, into an infinite sequence of states σ_i , for $i \geq 0$, such that

$$\bullet \sigma_0 = s_0^F,$$

- $\sigma_i \xrightarrow{F} \sigma_{i+1}$, and
- whenever $\sigma_i = s^F$, $\diamond\phi \in s$, and ϕ is equivalent to some formula in Γ , then $\phi \in t$ for some t such that $\sigma_j = t^F$ for some $j \geq i$.

CLOCK implies that every state σ_i contains one of the time-difference formulas. We reassign, in accordance with the time-difference formulas, times to all the states, thus obtaining the *canonical timed state sequence*

$$\mathcal{M}^T(\phi_0) = (\Sigma^T, \rightarrow_{\bigcirc}^T, \rightarrow_{\square}^T, N, \|\|^T, \llbracket x \rrbracket_{x \in V}^T, \llbracket p \rrbracket_{p \in P}^T, \sigma_0),$$

where

- $\Sigma^T = \{\sigma_i \mid i \geq 0\}$,
- $\rightarrow_{\bigcirc}^T = \{(\sigma_i, \sigma_{i+1}) \mid i \geq 0\}$,
- \rightarrow_{\square}^T is the reflexive transitive closure of \rightarrow_{\bigcirc}^T ,
- $|\sigma_{i+1}|^T = |\sigma_i|^T + \delta$ if $Next_{\delta} \in \sigma_i$, and
 $|\sigma_{i+1}|^T = |\sigma_i|^T + \Delta + 1$ if $Next_{>\Delta} \in \sigma_i$,
- $\llbracket p \rrbracket^T(s^F)$ iff $\llbracket p \rrbracket^F(s^F)$.

Note that $|\sigma_0|^T$ and $\llbracket x \rrbracket^T$ are left arbitrary; they need to be specified only in case ϕ_0 contains any absolute time references or free variables, respectively. The following main theorem asserts that we have indeed constructed a model of ϕ_0 . The proof depends crucially on Lemma 4.1 as well as UPD3 and UPD4, which ensure the consistency of all timing constraints.

Theorem 5.2 (Canonical timed state sequence) *Let $\mathcal{M}^T(\phi_0)$ be the canonical timed state sequence for the formula ϕ_0 of TPTL. Then $\phi \in \sigma_i$ iff $\mathcal{M}^T(\phi_0)[\sigma_0 := \sigma_i] \models \phi$ for all $i \geq 0$ and $\phi \in \text{Closure}(\phi_0)$.*

Proof of Theorem 5.2 We apply induction on the structure of ϕ , for $\phi \in \text{Closure}(\phi_0)$.

If ϕ is of the form $z.p$ for some proposition p , use Q3. The case that ϕ is of the form $z.m = n$, $z.m < n$, $z.z + m = z + n$, or $z.z + m < z + n$ follows from NAT and Q⁻. The propositional cases are established by Q⁻. If ϕ is of the form $z.x.\psi$, use Lemma 5.7 and VARS.

Now suppose that ϕ is of the form $z. \bigcirc \psi$ and $|\sigma_{i+1}|^T = |\sigma_i|^T + \delta$; that is, $Next_\delta \in s$ if $\delta \leq \Delta$, and $Next_{>\Delta} \in s$ otherwise. Furthermore, by the definition of \rightarrow_{\bigcirc}^F there are s, t such that $\sigma_i = s^F$, $\sigma_{i+1} = t^F$, and $s \rightarrow_{\bigcirc} t$. Then,

$$\mathcal{M}^T(\phi_0)[\sigma_0 := \sigma_i] \models z. \bigcirc \psi$$

iff $z. \psi^\delta \in t$ by Lemma 4.1 and the induction hypothesis. LIN1 and the canonical-model construction imply that $z. \psi^\delta \in t$ iff $\bigcirc z. \psi^\delta \in s$; and $\bigcirc z. \psi^\delta \in s$ iff $z. \bigcirc \psi \in \sigma_i$ follows from UPD3 or UPD4.

Finally, suppose that ϕ is of the form $z. \square \psi$. Let $\delta_j = |\sigma_j|^T - |\sigma_i|^T$ for all $j \geq i$. In this case,

$$\mathcal{M}^T(\phi_0)[\sigma_0 := \sigma_i] \models z. \square \psi$$

iff $z. \psi^{\delta_j} \in \sigma_j$ for all $j \geq i$, by Lemma 4.1 and the induction hypothesis.

We show that $z. \square \psi \in \sigma_i$ implies $z. \square \psi^{\delta_j} \in \sigma_j$ for all $j \geq i$, by induction on j ; then $z. \psi^{\delta_j} \in \sigma_j$ by LIN4 and Q*. Assume that $z. \square \psi^{\delta_j} \in \sigma_j$ and $Next_{\delta'} \in \sigma_j$, and show that $z. \square \psi^{\delta_j + \delta'} \in \sigma_{j+1}$. By the definition of \rightarrow_{\bigcirc}^F there are s, t such that $\sigma_j = s^F$, $\sigma_{j+1} = t^F$, and $s \rightarrow_{\bigcirc} t$. From $z. \square \psi^{\delta_j} \in s$, by LIN4 and Q* $z. \bigcirc \square \psi^{\delta_j} \in s$. Since $Next_{\delta'} \in s$, by UPD3 or UPD4 $\bigcirc z. (\square \psi^{\delta_j})^{\delta'} \in s$; that is, $\bigcirc z. \square \psi^{\delta_j + \delta'} \in s$. Hence $z. \square \psi^{\delta_j + \delta'} \in t$ by the canonical-model construction.

Conversely, assume that $z. \diamond \psi \in \sigma_i$ and $z. \psi^{\delta_j} \notin \sigma_j$ for all $j \geq i$, and show a contradiction. First observe that, by induction on j , it follows that $z. \diamond \psi^{\delta_j} \in \sigma_j$ for all $j \geq i$.

We distinguish two cases. If $\delta_j > \Delta$ for some $j \geq i$, then $z. \diamond \psi^\Delta \in \sigma_j$ and $z. \psi^\Delta \notin \sigma_{j'}$ for all $j' \geq j$. Let $\sigma_j = s^F$. Since $z \notin \diamond \psi^\Delta$, by Q* $z. \diamond \psi^\Delta$ is equivalent to $\diamond z. \psi^\Delta$; hence, by Lemma 5.13 there is some $j' \geq j$ with $\sigma_{j'} = t^F$ and $z. \psi^\Delta \in t$, contradicting $z. \psi^\Delta \notin \sigma_{j'}$.

On the other hand, suppose that there is some $j \geq i$ such that $\delta_{j'}$ is constant for all $j' \geq j$; then there is some such $j \geq i$ such that $\sigma_j = s^F$ and s^F is in the final \rightarrow_{\bigcirc}^F -component of $\mathcal{M}^F(\phi_0)$. Therefore $Next_0 \in t^F$ for all t^F with $s^F \rightarrow_{\bigcirc}^F t^F$. By part (2) of Lemma 5.12, $Next_{=0} \in t$ for all t such that $s \rightarrow_{\bigcirc} t$; thus $\square Next_0 \in s$ by Theorem 5.1. By TSS1, $z. \square z. x = z \in s$, and by TSS2, $z. \diamond \psi^{\delta_j}$ is equivalent to $\diamond z. \psi^{\delta_j}$. Hence, by Lemma 5.13 there is some $j' \geq j$ with $z. \psi^{\delta_j} \in \sigma_{j'}$, again a contradiction. ■

This finishes the completeness proof for TPTL. We conclude by indicating how absolute time references, free variables, and *until* operators can be incorporated into our argument.

A detailed formal treatment of these cases is straightforward and left to the ambitious reader.

Absolute time references

To handle absolute time references, we include in the filtration set Γ all formulas of the form $z.z = \delta$ for $0 \leq \delta \leq \Delta$, as well as $z.z > \Delta$. In the definition of the canonical timed state sequence $\mathcal{M}^T(\phi_0)$, let $|\sigma_0| = \delta$ if $z.z = \delta \in \sigma_0$, and $|\sigma_0| = \Delta + 1$ if $z.z > \Delta \in \sigma_0$. The additional base cases in the proof of the main theorem can be shown by induction on the canonical state sequence.

Free variables

If ϕ_0 contains free variables, then it is no longer the case that every model can be compressed into a model all of whose time steps are at most $\Delta + 1$. However, it is not hard to see that, if ϕ_0 contains N free variables, then ϕ_0 is satisfiable iff it is satisfiable by a timed state sequence all of whose time steps are at most $\Delta' = (N + 1)(\Delta + 1)$. This is because, in any interpretation, the difference between any two times that are either associated with a state or a free variable, can be reduced to $\Delta + 1$ without changing the truth of ϕ_0 .

Thus the completeness proof goes through if we take the filtration set Γ large enough (replace Δ by Δ'), and include all formulas of the form $z.z = z + \delta$ and $z = y + \delta$, for $0 \leq \delta \leq \Delta'$, as well as $z.z > z + \Delta'$ and $z > y + \Delta'$, for all free variables $z, y \in \phi_0$ (recall that $z \notin \phi_0$). For this purpose, it is necessary to redefine the the time-update function $z.\phi^\delta$ for TPTL-formulas ϕ with free variables; for instance, let $z.(z < x)^1$ be the formula

$$z.(z < x \vee z = x).$$

The axiom PRO ensures that the variable assignment function of the canonical timed state sequence can be defined properly. If the *progress* condition on timed state sequences is dropped, we can obtain a complete proof system by replacing the nonlogical axiom PRO with the weaker version

$$\text{PRO}' \quad \Box z.\Diamond y.y > z \rightarrow \Diamond z.z \geq z.$$

We point out that the canonical-model construction for TPTL-formulas with free variables leads to a tableau-based decision procedure for TPTL with free variables, whose

validity problem is, therefore, no harder than determining the validity of closed TPTL-formulas.

Until operators

Also the addition of the temporal *until* operator \mathcal{U} yields no surprises. The syntax of TPTL can be extended to admit formulas of the form $\phi_1 \mathcal{U} \phi_2$ such that for any timed state sequence \mathcal{M} :

$\mathcal{M} \models \phi_1 \mathcal{U} \phi_2$ iff $\mathcal{M}[\sigma_0 := \sigma_i] \models \phi_2$ for some $i \geq 0$, and $\mathcal{M}[\sigma_0 := \sigma_j] \models \phi_1$ for all $0 \leq j < i$.

By adding, to TPTL, the two axiom schemata

$$\begin{aligned} \text{UNTIL1} \quad & \phi_1 \mathcal{U} \phi_2 \rightarrow \Diamond \phi_2, \\ \text{UNTIL2} \quad & \phi_1 \mathcal{U} \phi_2 \leftrightarrow (\phi_2 \vee (\phi_1 \wedge \bigcirc(\phi_1 \mathcal{U} \phi_2))), \end{aligned}$$

which characterize the *until* operator completely in PTL [40], we obtain a complete proof system for TPTL. From an generalization of Theorem 5.2 to absolute time references and *until* operators it follows that

Corollary 5.2 (Completeness of TPTL) *A formula ϕ of TPTL is valid if and only if $\text{TPTL} \vdash \phi$.*

5.2.4 Nonaxiomatizable extensions

We show that half-order temporal logic in general is, unlike TPTL, not (recursively) axiomatizable and, therefore, highly undecidable. From TPTL we obtain TPTL^2 by restraining time to provide a state counter that starts, in the initial state, at 0, and by adding the unary function symbol $2 \cdot$ that is interpreted, on \mathbb{N} , as multiplication by 2. With Theorem 3.8, we have proved the validity problem for TPTL^2 -formulas to be Π_1^1 -hard, which implies that there is no complete proof system for this logic, as well as for TPTL with addition. Here we show that if half-order temporal logic were axiomatizable, for any choice of function and relation symbols, then so would be TPTL^2 . It follows that temporal interpretations cannot be completely characterized in half-order modal logic.

Let ϕ^2 denote the conjunction of the following formulas of TPTL^2 :

COUNT1	$x \cdot x = 0,$
COUNT2	$\Box x. \bigcirc y. y = Sx,$
SUCC1	$\Box x. Sx \neq 0,$
SUCC2	$\Box x. \Box y. (Sx = Sy \rightarrow x = y),$
LESS1	$\Box x. x \not< x,$
LESS2	$\Box x. \bigcirc \Box y. (x < y \wedge y \not< x),$
DOUB1	$2 \cdot 0 = 0,$
DOUB2	$\Box x. (2 \cdot Sx = SS(2 \cdot x)).$

Recall that a *temporal* interpretation for half-order modal logic is one in which the set of states together with the two accessibility relations \rightarrow_{\bigcirc} and \rightarrow_{\Box} is isomorphic to the structure $(\mathbb{N}, +1, \leq)$; the set of values as well as the interpretation of all function and relation symbols is left arbitrary. The following proposition states that ϕ^2 , viewed as a formula of half-order modal logic with uninterpreted function and relation symbols, completely characterizes multiplication by 2 in temporal interpretations.

Proposition 5.2 (Half-order temporal logic) *A closed formula ψ of TPTL² is valid iff the formula $\phi^2 \rightarrow \psi$ of half-order modal logic is true under all temporal interpretations.*

Proof of Proposition 5.2 It is not hard to see that the closed formula ϕ^2 of half-order modal logic is true under a temporal interpretation

$$\mathcal{M}: \sigma_0 \rightarrow_{\bigcirc} \sigma_1 \rightarrow_{\bigcirc} \sigma_2 \rightarrow_{\bigcirc} \dots$$

iff the set of values of \mathcal{M} contains \mathbb{N} (modulo isomorphism), $|\sigma_i| = i$ for all $i \geq 0$, and the function and relation symbols 0, S, <, and 2· are, on \mathbb{N} , interpreted as the zero and successor functions, the ordering relation, and multiplication by 2, respectively. The proposition follows. ■

Since TPTL² is Π_1^1 -hard, so is the restriction of half-order modal logic to temporal interpretations. We have, in fact, shown that any extension of TPTL with a single uninterpreted unary function symbol is Π_1^1 -hard, because this function symbol can be forced, by ϕ^2 , to be interpreted as multiplication by 2 on the natural numbers.

5.3 Proving Timed Transition Systems Correct

We present two methods for the deductive verification of timed transition systems (the terminology *exogenous* versus *endogenous* is taken from Pnueli [106]):

Exogenous Just as transition systems can be explicitly encoded by temporal formulas, we have seen that timed transition systems can be encoded in TPTL or MTL. Any proof system for these logics can, therefore, be used to derive properties of a timed transition system, to prove the equivalence of two timed transition systems, or to prove that one timed transition system refines another timed transition system. In the untimed case, the possibility of encoding systems by temporal formulas and proving temporal implications was seen by Pnueli [107] and has more recently been strongly advocated by Lamport [79].

Endogenous If we wish to prove properties of a given timed transition system S only, we may, instead of presenting the system as a temporal formula, add proof rules to our proof system that need not be unconditionally sound, but only sound for reasoning about the runs of S . In the untimed temporal framework, this approach of reasoning about a single hidden program was first advocated by Pnueli [106] and has been greatly refined by many researchers (see, for example, [91, 105], and compare the related framework of UNITY [26]).

Indeed, as Pnueli points out, both methods have had a long tradition in program verification before the advent of temporal logic [106]. While the exogenous approach is, by staying fully within logic, both more uniform and more general, Pnueli has argued that the endogenous approach, when applicable, is preferable for any specific verification task, because it equips the verifier with the strongest possible tools for a particular class of verification problems. In the case of real-time reasoning, we have already provided the foundation for an exogenous verification method; it will be discussed in the following subsection. Thereafter, an endogenous method for real-time verification will be motivated and fully developed in the next chapter.

5.3.1 Reasoning about explicit programs

Suppose we are given a finite-state timed transition system S . We can use the proof system for TPTL to show that all runs of S meet the TPTL-specification ψ in two steps:

1. Write the logical representation ϕ_S of S , as defined in Subsection 4.3.3. Then

$$\Pi(\phi_S) = \text{Det}(\Pi(S));$$

that is, the runs of the system S are exactly the models of the TPTL-formula ϕ_S .

2. Prove that the TPTL-formula $\phi_S \rightarrow \psi$ is valid.

Even if the timed transition system S has an infinite number of states, this approach is, in principle, possible, provided we complement the proof system for TPTL with capabilities to reason about the infinitary data domains of S . Thus, a proof system for exogenous real-time verification can be partitioned into two parts:

General part Axioms and proof rules to establish the validity of TPTL-formulas. This part provides the tools for domain-independent reasoning.

Domain part Axioms and proof rules to reason about the underlying data domains of a system. This part generally consists of first-order theories of data types.

By giving a complete proof system for TPTL, we have presented a *general* part. However, while our proof system is of theoretical interest, its derivations are on a level that is much too low to be practical. This point will be illustrated in the following segment. Thus, to make real-time verification feasible for systems that lie outside the scope of decision procedures, the proof system for TPTL has to be extended by useful derivable metarules.

Deriving new proof rules

A typical step in the deduction of a bounded-response property requires the chaining of more local bounded-response properties. Hence the following inference rule turns out to be very practical:

$$\begin{array}{l} \text{from } \Box x. (\phi_1 \rightarrow \Diamond y. (\phi_2 \wedge y \leq x + m)) \\ \text{and } \Box x. (\phi_2 \rightarrow \Diamond y. (\phi_3 \wedge y \leq x + n)) \\ \hline \text{infer } \Box x. (\phi_1 \rightarrow \Diamond y. (\phi_3 \wedge y \leq x + m + n)) \end{array}$$

\Diamond -TRANS

for all constants $m, n \geq 0$, provided that neither x nor y occurs free in any of ϕ_1 , ϕ_2 , and ϕ_3 . Since \Diamond -TRANS is sound over all timed state sequences, by our completeness result it must be derivable within the given proof system for TPTL. However, this derivation, which we are going to sketch briefly, is extremely tedious; it vividly demonstrates the need of practical high-level inference rules for real-time verification.

So let us derive \Diamond -TRANS. In fact, we show the stronger assertion that the two antecedents of \Diamond -TRANS imply the consequent:

$$\begin{aligned} & \Box x. (\phi_1 \rightarrow \Diamond y. (\phi_2 \wedge y \leq x + m)) \rightarrow \\ & \Box x. (\phi_2 \rightarrow \Diamond y. (\phi_3 \wedge y \leq x + n)) \rightarrow \\ & \Box x. (\phi_1 \rightarrow \Diamond y. (\phi_3 \wedge y \leq x + m + n)) \end{aligned}$$

is valid. By K_{\Box} , TRANS, Q^* , and VAR2, it suffices to derive

$$\begin{aligned} & x. \Diamond (\phi_2 \wedge y. y \leq x + m) \rightarrow \\ & \Box (\phi_2 \rightarrow y. \Diamond (\phi_3 \wedge z. z \leq y + n)) \rightarrow \\ & x. \Diamond (\phi_3 \wedge z. z \leq x + m + n), \end{aligned}$$

which can be rewritten (use K_{\Box}) as

$$\begin{aligned} & \Box (y. \Box (z. z \leq y + n \rightarrow \neg \phi_3) \rightarrow \neg \phi_2) \rightarrow \\ & x. \Box (z. z \leq x + m + n \rightarrow \neg \phi_3) \rightarrow \\ & x. \Box (y. y \leq x + m \rightarrow \neg \phi_2). \end{aligned}$$

By Q^* , K_{\Box} , and TRANS show

$$\begin{aligned} & \Box (y. \Box (z. z \leq y + n \rightarrow \neg \phi_3) \rightarrow \neg \phi_2) \rightarrow \\ & \Box y. \Box (z. z \leq x + m + n \rightarrow \neg \phi_3) \rightarrow \\ & \Box (y. y \leq x + m \rightarrow \neg \phi_2). \end{aligned}$$

Applying K_{\Box} and Q^* again, it suffices to derive

$$\Box (z. z \leq x + m + n \rightarrow \neg \phi_3) \rightarrow y \leq x + m \rightarrow \Box (z. z \leq y + n \rightarrow \neg \phi_3).$$

By RIG1 $_{\Box}$, RIG7, K^* , and Q^* , show

$$y \leq x + m \rightarrow z \leq y + n \rightarrow z \leq x + m + n,$$

which follows from NAT.

5.3.2 Reasoning about one implicit program

Suppose we wish to verify a particular timed transition system S . The endogenous approach to verification allows us not only to extend the repertoire of available proof rules by useful derivable rules such as \diamond -TRANS, but to complement a *general* proof system for TPTL with tools that are sound for reasoning about the given system S only. Thus, an endogenous proof system contains three components [88]:

General part As before, this part consists of a complete proof system for TPTL and derived theorems and metarules of TPTL; it provides the tools for system and domain-independent reasoning.

Program part This part consists of axioms and proof rules that restrict the models under consideration to the execution sequences (runs) of a particular system (program); it exploits the structure of the given timed transition system S to provide powerful tools for reasoning about S .

Domain part As before.

To offer strong and practical proof rules, the *program* part is, ideally, designed in accordance with proof methodologies for different classes of real-time properties (see, for example, [91] for the untimed case). In the following chapter, we will present two alternative *program* parts, which correspond to two different proof methodologies, for the derivation of bounded-invariance and bounded-response properties of timed transition systems.



Chapter 6

Deductive Verification: Program Part

We present high-level proof rules for verifying that all runs of a timed transition system satisfy a bounded-invariance property or a bounded-response property. In fact, we discuss *two* alternative proof methodologies that are quite different in flavor:

Explicit-clock reasoning The explicit-clock style of establishing real-time properties accesses absolute time explicitly as a state parameter. The proofs in this style rely on *global* invariants and have the flavor of untimed *safety* arguments.

Bounded-operator reasoning The hidden-clock style of establishing real-time properties refers to time implicitly only, through the relative offsets of time-bounded temporal operators. The proofs in this style proceed by incrementally combining *local* timing properties and resemble untimed *liveness* arguments.

Consider, for example, the bounded-response property Π_3^\diamond that is defined by the MTL-formula

$$\Box(p \rightarrow \Diamond_{\leq 3} q), \quad (\phi_3^\diamond)$$

which states that “every p -state is followed by a q -state within 3 time units.” We have shown, in Subsection 1.2.3, that this property is a (real-time) safety property. Indeed, even though it has been expressed by a “liveness-like” formula (employing a bounded version of the liveness operator \Diamond), the bounded-response property Π_3^\diamond can, alternatively, be defined by

a temporal formula that uses the untimed safety operator U (*unless*) and a clock variable t that refers, in any state, to the current time:

$$\Box((p \wedge t = z) \rightarrow (t \leq z + 3) U (q \wedge t \leq z + 3)).$$

This formula states that if p happens at time z , then from this point on, the time will not exceed $z + 3$ either forever (which is ruled out by the requirement on timed state sequences that time progresses eventually) or until q happens. It follows that q must occur within 3 time units from p . Consequently, no new proof rules are necessary for the explicit-clock style of real-time verification: if the system states are augmented by a clock variable, real-time properties can, in principle, be verified using a standard, uniform set of untimed rules. This approach of proving timing properties has been advocated, among others, by Lamport [79].

On the other hand, when using the time-bounded temporal operators of MTL for the specification of real-time properties, one discerns a clear dichotomy between the definition of *upper-bound* properties, such as the bounded-response formula

$$\Box(p \rightarrow \Diamond_{\leq 3} q) \quad (\phi_3^\diamond)$$

considered above, and *lower-bound* properties, such as the bounded-invariance formula

$$\Box(p \rightarrow \Box_{< 3} \neg q), \quad (\phi_3^\square)$$

which states that “no p -state is followed by a q -state within less than 3 time units.” While upper-bound properties assert that something good will happen within a specified amount of time, lower-bound properties assert that nothing bad will happen for a certain amount of time. Clearly, the MTL-specification of upper-bound properties bears a close resemblance to the temporal definition of the *liveness* properties of untimed response, and the MTL-specification of lower-bound properties closely resembles the temporal definition of the *safety* properties of untimed invariance. The second proof system we present cultivates this similarity by introducing separate proof principles for the classes of lower-bound (i.e., bounded-invariance) and upper-bound (i.e., bounded-response) properties. These proof principles, which often follow intuitive correctness arguments for timing properties more closely than the use of an explicit clock variable, can easily be seen to be natural extensions of the proof rules for the untimed invariance and response classes, respectively [90].

6.1 Properties of Timed Transition Systems

Throughout this chapter, we prove properties of a given fixed timed transition system $S = \langle \Sigma, \Theta, \mathcal{T}, l, u \rangle$. We assume that S is associated with a concrete real-time system P that belongs to one of the classes we have discussed in Chapter 2; the modeled system P may, for example, be a multiprocessing or a multiprogramming system, a shared-variables or a message-passing system. It follows that S is defined by a set of timed transition diagrams, one for every component process of P , and that the set Σ of states consists of all interpretations of a finite set V of data and control variables that range over finite or infinite domains. Throughout this chapter, we use the following assumptions about V :

1. We assume that, in addition to data and control variables, V contains sufficiently many *auxiliary* variables that range over the natural numbers \mathbb{N} and are not changed by any of the transitions of S . We will on occasion need a “new, rigid” variable, and for this purpose we employ one of the auxiliary variable that have not been used previously.
2. We assume that, for every variable $x \in V$, there is a corresponding unique primed variable $x' \notin V$ that ranges over the same domain as x .

6.1.1 State properties

We are given an assertion language — a first-order language with equality that contains interpreted function and predicate symbols to express operations and relations on the domains of the variables in V . A *state formula* is a first-order formula p of the assertion language such that only variables from V occur freely in p . Thus, every state in Σ provides an interpretation for the state formulas. If the state formula p is true in state σ , we say that σ is a p -state. We use the following abbreviations for state formulas:

- For any transition $\tau \in \mathcal{T}$, the *enabling* condition $enabled(\tau)$ asserts that τ is enabled. In particular, $enabled(\tau_I)$ abbreviates *true* for the idle transition τ_I .
- For any transition $\tau \in \mathcal{T}$ and state formulas p and q , the *verification* condition $\{p\}\tau\{q\}$ asserts that if p is true of a state $\sigma \in \Sigma$, then q is true of all τ -successors

of σ . In particular, $\{p\} \tau_I \{q\}$ stands for the universal closure of the formula $p \rightarrow q$. For any set $T \subseteq \mathcal{T}$ of transitions, we write $\{p\} T \{q\}$ for the conjunction

$$\bigwedge_{\tau \in T} \{p\} \tau \{q\}$$

of all individual verification conditions.

- For any transition $\tau \in \mathcal{T}$ and state formulas p and q , the *inverse verification condition* $\{p\} \tau^- \{q\}$ asserts that if p is true of a state $\sigma \in \Sigma$, then q is true of all τ -predecessors of σ . Observe that all inverse verification conditions are definable by ordinary verification conditions:

$$\{p\} \tau^- \{q\} \text{ is equivalent to } \{\neg q\} \tau \{\neg p\}.$$

In particular, $\{p\} \tau_I^- \{q\}$ is equivalent to $\{p\} \tau_I \{q\}$ for the idle transition τ_I . For any set $T \subseteq \mathcal{T}$ of transitions, we write $\{p\} T^- \{q\}$ for the conjunction of the inverse verification conditions for all transitions in T .

Note that while the truth value of an enabling condition depends on the state in which it is interpreted, the verification conditions are state-independent and, thus, equivalent to closed formulas.

In the case that the timed transition system S is associated with a shared-variables multiprocessing system P , it is not hard to see that the enabling and verification conditions of all transitions can indeed be expressed by state formulas. Suppose that P consists of the m processes P_i , for $1 \leq i \leq m$, and the data precondition θ , which is a state formula:

$$\{\theta\} [P_1 \parallel \dots \parallel P_m].$$

Let us assume that each process P_i , $1 \leq i \leq m$, is given by a timed transition diagram with the locations $\{\mathcal{L}_0^i, \dots, \mathcal{L}_{n_i}^i\}$ and the entry location \mathcal{L}_0^i . We write $at \mathcal{L}_\perp^i$ for $\pi_i = \perp$, and $at \mathcal{L}_j^i$ for $\pi_i = \mathcal{L}_j^i$; that is, the control of the process P_i is at the location \mathcal{L}_j^i . We abbreviate any disjunction $at \mathcal{L}_j^i \vee at \mathcal{L}_k^i$ further, to $at \mathcal{L}_{j,k}^i$.

1. For each entry transition $\tau_0^i \in \mathcal{T}$ of S_P , the enabling condition $enabled(\tau_0^i)$ is equivalent to the state formula

$$at \mathcal{L}_\perp^i,$$

and the verification condition $\{p\} \tau_0^i \{q\}$ is equivalent to the universal closure of the formula

$$(p \wedge \text{enabled}(\tau_0^i) \wedge (\text{at-}\mathcal{L}_0^i)' \wedge \bigwedge_{y \in V - \{x_i\}} (y' = y)) \rightarrow q',$$

where the formula q' is obtained from q by replacing every variable with its primed version; for example, $(\text{at-}\mathcal{L}_0^i)'$ stands for $\pi_i^i = \mathcal{L}_0^i$. The inverse verification condition $\{p\} (\tau_0^i)^- \{q\}$ is equivalent to the universal closure of

$$(p' \wedge \text{enabled}(\tau_0^i) \wedge (\text{at-}\mathcal{L}_0^i)' \wedge \bigwedge_{y \in V - \{x_i\}} (y' = y)) \rightarrow q.$$

2. All other nonidle transitions of S_P correspond to edges in the timed transition diagrams for the processes P_i . Let $\tau_{j \rightarrow k}^i \in \mathcal{T}$ be such a transition and assume that the corresponding edge that connects the location \mathcal{L}_j^i to the location \mathcal{L}_k^i and is labeled by the instruction $c \rightarrow \bar{x} := \bar{e}$. Then, the enabling condition $\text{enabled}(\tau_{j \rightarrow k}^i)$ is equivalent to

$$\text{at-}\mathcal{L}_j^i \wedge c,$$

and the verification condition $\{p\} \tau_{j \rightarrow k}^i \{q\}$ is equivalent to the universal closure of the formula

$$(p \wedge \text{enabled}(\tau_{j \rightarrow k}^i) \wedge (\text{at-}\mathcal{L}_k^i)' \wedge (\bar{x}' = \bar{e}) \wedge \bigwedge_{y \in V - \{x_i, x\}} (y' = y)) \rightarrow q'.$$

The inverse verification condition $\{p\} (\tau_{j \rightarrow k}^i)^- \{q\}$ is equivalent to the universal closure of

$$(p' \wedge \text{enabled}(\tau_{j \rightarrow k}^i) \wedge (\text{at-}\mathcal{L}_k^i)' \wedge (\bar{x}' = \bar{e}) \wedge \bigwedge_{y \in V - \{x_i, x\}} (y' = y)) \rightarrow q.$$

It is also straightforward to express the enabling and verification conditions as state formulas, if the timed transition system S is associated with any of the other concrete real-time systems that we have introduced in Chapter 2, such as message-passing, multiprogramming, dynamic, and priority systems.

Synchronous multiprocessing systems

Our examples will be drawn from timed transition systems S that are associated with multiprocessing systems P of the form

$$\{\theta\} \{P_1 \parallel_s \dots \parallel_s P_m\}$$

all of whose component processes start synchronously (i.e., at the exact same time). We call such a system *synchronous* and model it by a single entry transition that sets all control variables, simultaneously, to the entry locations of the individual processes. For multiprocessing systems P , it is convenient to define the following two additional abbreviations for state formulas:

- The *ready* condition *ready* holds precisely in the initial states Θ of S_P ; it indicates that none of the processes of P has started yet. Consequently, the ready condition *ready* of S_P stands for the state formula

$$\theta \wedge \left(\bigwedge_{1 \leq i \leq m} at_{L_1^i} \right).$$

- The *synchronous starting* condition *start* indicates that all processes of P have entered their entry locations, but none has proceeded any further; that is, *start* abbreviates the state formula

$$\theta \wedge \left(\bigwedge_{1 \leq i \leq m} at_{L_0^i} \right).$$

Note that if P is synchronous, then the two verification conditions

$$\{ready\} T - \tau_I \{start\},$$

$$\{start\} (T - \tau_I)^- \{ready\}$$

are valid (by $T - \tau$ we denote the set difference $T - \{\tau\}$).

6.1.2 Temporal properties

Temporal formulas are constructed from state formulas by boolean connectives and time-bounded temporal operators. They are interpreted over timed state sequences whose states are drawn from Σ and whose times are natural numbers. In this chapter, we are interested in proving two classes of real-time properties of timed transition systems — bounded-invariance properties and bounded-response properties. Thus we restrict ourselves, semantically, to the digital-clock model, which was justified in Subsection 3.1.2, and, syntactically, to the following fragment of MTL:

- Every state formula is a temporal formula.

- Every boolean combination of temporal formulas is a temporal formula.
- If p is a state formula, ϕ a temporal formula, and $l \in \mathbb{N}$, then $pU_{\geq l}\phi$ is a temporal formula; recall that it is true over the timed state sequence $\rho = (\sigma, T)$ iff either all σ_i , for $i \geq 0$, are p -states, or there is some position $i \geq 0$ such that $T_i \geq T_0 + l$, ϕ is true over the i -th suffix ρ^i of ρ , and all σ_j , for $0 \leq j < i$, are p -states. We use the abbreviations $pU\phi$, $\Box_{<l}p$, and $pU_{\geq l}^+\phi$ for the temporal formulas $pU_{\geq 0}\phi$, $pU_{\geq l}\text{ true}$, and $p \wedge (pU_{\geq l}\phi)$, respectively.
- If ϕ is a temporal formula and $u \in \mathbb{N}$, then $\Diamond_{\leq u}\phi$ is a temporal formula; recall that it is true over the timed state sequence $\rho = (\sigma, T)$ iff there is some position $i \geq 0$ such that $T_i \leq T_0 + u$ and ϕ is true over the i -th suffix ρ^i of ρ .

From now on, we use the convention that the letters p, q, r as well as φ (and primed versions) denote state formulas, while the letters ϕ, ψ , and χ stand for arbitrary temporal formulas.

S-validity and *S*-soundness

The following definitions and comments apply equally to timed and untimed transition systems as well as to all formulas that are interpreted over timed state sequences and state sequences, respectively. Recall that the run fragments of a (timed) transition system are obtained by closing its runs under suffixes. We say that a formula ϕ is *S*-valid iff it is true over all run fragments of the (timed) transition system *S*. While (general) validity — truth over all (timed) state sequences — implies *S*-validity for every system *S*, the converse does not necessarily hold. In fact, even a state formula p that is *S*-valid may not be true in some states of *S* that do not occur along any run of *S* and, hence, p may not be generally valid.

If a formula ϕ is *S*-valid, then it is, by definition, satisfied by all runs of *S*. Thus, to show that the given system *S* meets the specification ϕ , it suffices to show that ϕ is *S*-valid. This observation has two important ramifications:

1. Since the set of run fragments of *S* is closed under suffixes, a formula ϕ is *S*-valid iff the invariance $\Box\phi$ is *S*-valid. Therefore, as we are concerned with *S*-validity only in this chapter, we may omit all outermost unbounded *always* operators from specifications.

2. A proof rule is called *S-sound* iff the *S*-validity of all premises implies the *S*-validity of the conclusion. Clearly, a generally sound rule may not be *S*-sound, and vice versa. Therefore, for verifying properties of the given system *S* by proving *S*-validity, we restrict ourselves to *S*-sound proof rules.

We shall build extensively on both remarks throughout this chapter, and begin by discussing some immediate applications of *S*-validity and *S*-soundness in the following two segments.

Bounded invariance and bounded response

First, we point out that the *bounded-unless* and *bounded-eventually* operators of the fragment of MTL that we have chosen for this chapter suffice to define both bounded-invariance and bounded-response properties of the given timed transition system *S*.

Bounded invariance The *bounded-invariance formula*

$$p \rightarrow \Box_{<l} q$$

is *S*-valid iff for every run (σ, T) of *S* and all $i \geq 0$ and $j \geq i$,

if σ_i is a *p*-state and $T_j < T_i + l$,
then σ_j is a *q*-state;

that is, no *p*-state is followed by a $\neg q$ -state within time less than *l*. A typical application of bounded invariance is to state a lower bound *l* on the termination of a multiprocessing system *S* with the termination condition *r*: the temporal formula

$$\text{ready} \rightarrow \Box_{<l} \neg r$$

asserts that, if not started before time *t*, then *S* will not reach a final state before time $t + l$.

Bounded response The *bounded-response formula*

$$p \rightarrow \Diamond_{\leq u} q$$

is *S*-valid iff for every run (σ, T) of *S* and all $i \geq 0$,

if σ_i is a p -state,

then there is some q -state σ_j , with $j \geq i$, such that $T_j \leq T_i + u$;

that is, every p -state is followed by a q -state within time u . A typical application of bounded response is to state an upper bound u on the termination of a multiprocessing system S with the termination condition τ : the temporal formula

$$\text{start} \rightarrow \Diamond_{\leq u} \tau$$

asserts that if all component processes of S are started synchronously at time t , then S is guaranteed to reach a final state no later than at time $t + u$. As the runs of timed transition systems are closed under shifting the origin of time, we shall, without loss of generality, henceforth assume that $t = 0$.

Monotonicity rules

Secondly, we introduce two important proof rules that are S -sound for every timed transition system S . The monotonicity rule U-MON allows us to weaken any of the three arguments of the *bounded-unless* operator:

$$\text{U-MON} \frac{p \rightarrow p' \quad \phi \rightarrow \phi' \quad l' \leq l}{(p U_{\geq l} \phi) \rightarrow (p' U_{\geq l'} \phi')}$$

A similar monotonicity rule holds for the *bounded-eventually* operator:

$$\Diamond\text{-MON} \frac{\phi \rightarrow \phi' \quad u' \geq u}{(\Diamond_{\leq u} \phi) \rightarrow (\Diamond_{\leq u'} \phi')}$$

It is not hard to see that both monotonicity rules are generally sound as well as S -sound for every timed transition system S . Since propositional reasoning, too, is S -sound for every system S , we will refer to applications of the two weakening rules and propositional reasoning in derivations through the simple annotation "by monotonicity." For example, from the *bounded-unless formula*

$$p \rightarrow q U_{\geq l} \tau,$$

we can establish, by monotonicity, both the bounded-invariance formula

$$p \rightarrow \Box_{<1} q$$

and the unbounded *unless* formula

$$p \rightarrow q U \tau.$$

Note that every unless formula can be read as an untimed formula of PTL and interpreted over state sequences; that is, it defines an untimed safety property.

6.2 Explicit-clock Reasoning

We point out that none of our state formulas is able to refer to the value of the time, and the only references to time that are admitted in temporal formulas are bounds on temporal operators. In this section, we investigate the consequences of extending the notion of state, by adding a state variable t that represents, in every state, the current time. This extension is interesting, because once we are given explicit access to the absolute time through the clock variable t , both bounded-invariance and bounded-response formulas can, equivalently, be written as unbounded unless formulas and, consequently, be verified by conventional untimed techniques for establishing safety properties of transition systems.

6.2.1 Explicit-clock temporal logic

For the given timed transition system S and a bounded-invariance or bounded-response formula ϕ , we formalize the explicit-clock approach by

1. Translating the timed transition system S into the untimed *explicit-clock transition system* S' , which has been defined in Subsection 2.1.2. Note that if the states of S are all interpretations of the set V of variables, then the states of S' assign values to all variables in the set

$$V' = V \cup \{t\} \cup \{d_\tau \mid \tau \in \mathcal{T}\},$$

which is augmented by the clock variable and a delay counter for each transition of S .

2. Translating the specification ϕ over V into an untimed unless formula ϕ' over V' such that the *explicit-clock formula* ϕ' is S' -valid iff ϕ is S -valid:

- The explicit-clock translation of the bounded-invariance formula $p \rightarrow \Box_{<l} q$ is

$$(p \wedge t = z) \rightarrow q \cup (t \geq z + l),$$

for a new, rigid variable $z \in V$ that ranges over \mathbb{N} (recall that V supplies suitable variables that occur neither in the description of S nor in ϕ).

- The explicit-clock translation of the bounded-response formula $p \rightarrow \Diamond_{\leq u} q$ is

$$(p \wedge t = z) \rightarrow (t \leq z + u) \cup q$$

for a new, rigid variable $z \in V$ that ranges over \mathbb{N} .

Both unless formulas use the rigid variable z to record the time of the p -state. In the case of bounded-response properties the explicit-clock translation exploits the facts that all run fragments of S are deterministic timed state sequences and that the time is guaranteed to reach and surpass $z + u$, for any value of z . We emphasize that neither of the state formulas p and q may contain free occurrences of the clock variable or any of the delay counters.

From the properties of explicit-clock transition systems that we have inferred in Chapter 2, it is not hard to conclude that the explicit-clock formula ϕ^s is indeed S^s -valid iff ϕ is S -valid. We remark that, since both bounded-invariance and bounded-response formulas define safety properties, there has been no need to add fairness assumptions to the explicit-clock transition system.

Untimed temporal reasoning about real time

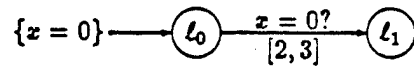
Our observations suggests a method for verifying bounded-invariance and bounded-response properties of timed transition systems: to prove the S -validity of the temporal formula ϕ , we establish instead the S^s -validity of the untimed safety formula ϕ^s . To show the unbounded unless formula ϕ^s , a single untimed *unless* rule suffices [89]:

$\text{UNLESS } \frac{p \rightarrow (\varphi \vee r) \quad \{\varphi\} T^s \{\varphi \vee r\} \quad \varphi \rightarrow q}{p \rightarrow q \cup r}$

We point out that all three premises of the *unless* rule are state formulas over the augmented set V' of variables; their S' -validity is usually shown by proving them generally valid. The state formula φ is called the *invariant* of the rule, because the main (i.e., second) premise asserts that φ is preserved by all transitions of the system S' (unless the desired state condition r is established).

To see that the rule UNLESS is S' -sound for the untimed transition system S' with the set T' of transitions, suppose that the three premises of the rule are true in all states that occur along any run fragment of S' , and consider an arbitrary run fragment σ of S' that contains a p -state σ_i . By the first premise, σ_i is either an r -state, thus satisfying the consequent of the rule, or the invariant φ holds at σ_i . The second premise guarantees that φ holds at all subsequent states either forever or until an r -state is encountered. Since φ implies q by the third premise, the consequent follows in either case.

To demonstrate the explicit-clock style of reasoning about timing properties, we look first at a trivial, yet already insightful, example; the verification of a more elaborate system will be carried out at the end of this section. Consider the single-process system P with the data precondition $z = 0$ and the following timed transition diagram:



Both the lower bound on the termination of P ,

$$\text{ready} \rightarrow \square_{<2} \neg \text{at}_{l_1},$$

is translated into the explicit-clock formula

$$(\text{ready} \wedge t = z) \rightarrow (\neg \text{at}_{l_1}) \cup (t \geq z + 2),$$

which can be derived by the *unless* rule from the invariant

$$(\text{at}_{l_1} \wedge t \geq z) \vee (\text{at}_{l_0} \wedge t \geq z + d_{c_{0-1}})$$

(we write d_{0-1} for the delay counter of the transition τ_{0-1} ; recall that it ranges over the set $\{0, 1, 2, 3\}$ only). The upper bound on the termination of P ,

$$\text{start} \rightarrow \diamond_{\leq 3} \text{at}_{l_1},$$

is translated into the untimed unless formula

$$(start \wedge t = z) \rightarrow (t \leq z + 3) \cup at_{\mathcal{L}_1},$$

which can be concluded by the *unless* rule from the invariant

$$at_{\mathcal{L}_0} \wedge z = 0 \wedge z \leq t \leq z + 3 \wedge t \leq z + d_{0 \rightarrow 1}.$$

The asymmetry of the invariants for establishing the lower and the upper bound reflects a subtle difference in the corresponding arguments, which will be fully exploited in the next section, where we will devise separate proof methodologies for deriving lower bounds and for deriving upper bounds.

Relative completeness

Let us conclude with two remarks about the explicit-clock style of proving real-time properties by untimed techniques:

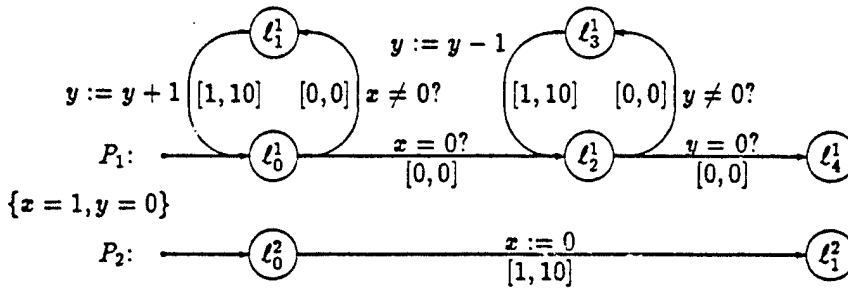
1. The method applies to every formula ϕ of TPTL or MTL that can be faithfully translated into an explicit-clock formula ϕ^s ; that is, ϕ^s is S^s -valid iff ϕ is S -valid.
2. The method is complete relative to reasoning about the data domains. This is because the *unless* rule has been shown complete, relative to state reasoning, for establishing unless formulas, provided the underlying data types and the assertion language are sufficiently powerful to encode runs of transition systems [89]. It follows that every bounded-invariance and bounded-response property of S can be shown by untimed reasoning:

Theorem 6.1 (Relative completeness of explicit-clock reasoning) *Let S be a timed transition system and let ϕ be a bounded-invariance or a bounded-response formula. If ϕ is S -valid, then the unless formula ϕ^s can be derived by the unless rule relative to state reasoning.*

6.2.2 Example: Race condition

As our main example, we present a multiprocessing system that looks innocent at first glance but turns out to be rather intricate, because its execution time depends on an

interesting interplay of the minimal delays and maximal delays of transitions that belong to different processes. Consider the following timed transition diagram definition of the *increment-decrement* system:



We wish to analyze the worst-case (maximal) running time of the synchronous two-process shared-variables multiprocessing system

$$\{x = 1, y = 0\}[P_1 ||_s P_2].$$

Note that the first process, P_1 , consumes the maximal amount of time if its first loop, in which the value of y is incremented, is executed as often (fast) as possible — 11 times: the control of P_1 may enter the first loop 11 times before and at time 10, the latest time at which the second process closes the loop, and it may spend another 10 time units in the first loop after the guard has been reversed. In this worst (slowest) case, the first loop is left at time 20 with $y = 11$ and, thus, the second loop may use up no more than 110 time units. It follows that P_1 terminates by time 130.

Assuming that assignments cost at least 2 time units (instead of 1), tests still being free, the maximal value of y would be only 6, implying termination by time 80: the increase of individual *lower* bounds decreases the composite *upper* bound! This phenomenon vividly demonstrates that real-time reasoning amounts to more than simply adding up minimal delays or maximal delays of individual transitions; it shows that lower-bound and upper-bound requirements are not independent, but may jointly affect the global time bounds of a system.

Let us now formally prove the upper bound 130 on the termination of P_1 by explicit-clock reasoning. To simplify the derivation, we may assume that both processes start simultaneously at time 0. Then we can infer the explicit-clock formula

$$(\text{start} \wedge t = d_{0 \rightarrow 1}^1 = d_{0 \rightarrow 1}^2 = 0) \rightarrow (t \leq 130) \cup \text{at } \ell_4^1$$

by the *unless* rule from the following global invariant:

$$\begin{aligned}
 & (at.L_0^1 \wedge at.L_0^2 \wedge (y = t = d_{0 \rightarrow 1}^2 = 0 \vee 1 \leq y \leq t = d_{0 \rightarrow 1}^2)) \vee \\
 & (at.L_1^1 \wedge at.L_0^2 \wedge y + d_{1 \rightarrow 0}^1 \leq t = d_{0 \rightarrow 1}^2) \vee \\
 & (at.L_0^1 \wedge at.L_1^2 \wedge 1 \leq y \leq 11 \wedge t \leq 20) \vee \\
 & (at.L_1^1 \wedge at.L_1^2 \wedge y \leq 10 \wedge t \leq 10 + d_{1 \rightarrow 0}^1) \vee \\
 & (at.L_2^1 \wedge at.L_1^2 \wedge y \leq 11 \wedge t + 10y \leq 130) \vee \\
 & (at.L_3^1 \wedge at.L_1^2 \wedge 1 \leq y \leq 11 \wedge t + 10y \leq 130 + d_{3 \rightarrow 2}^1).
 \end{aligned}$$

This proof of timely termination resembles a mechanical, exhaustive case analysis of all possible state-time combinations that can occur during an execution of the two processes of the *increment-decrement* system. In the following section, we will introduce an alternative style of deriving the desired bound on termination that follows much more closely the intuitive argument we have outlined above.

6.3 Bounded-operator Reasoning

In this section, we present an alternative, and quite different, proof method for establishing bounded-response and bounded-invariance properties of the given timed transition system S — a method that does not employ the clock variable t or any other reference to the absolute time. For this purpose, we formulate the property we wish to prove using the time-bounded temporal operators of MTL and employ, without detours, a deductive system for deriving the S -validity of bounded-invariance and bounded-response formulas. The proof rules fall into four categories:

1. The *single-step* rules derive real-time properties that follow from the lower-bound or upper-bound requirement for a single transition.
2. The *transitivity* rules combine two local real-time properties of the same type — that is, either two bounded-invariance properties or two bounded-response properties — into a composite timing property.
3. The *induction* rules combine arbitrarily many local real-time properties of the same type into a global timing property.
4. The *crossover* rules combine local real-time properties of opposite types into a composite timing property.

At the end of this section, we will show this proof system to be relatively complete for a restricted class of verification problems and discuss some of the limitations of bounded-operator reasoning.

6.3.1 Deterministic rules

First we present the bounded-operator methodology for verifying deterministic systems: a timed transition system S is called *deterministic* if any two guards that are associated with outgoing edges of the same vertex in the timed transition diagram representation of S are disjoint. Nondeterministic systems require more complex (conditional) single-step reasoning and will be treated in the next subsection.

Single-step rules

The *single-step lower-bound* rule uses the minimal delay $l_\tau \in \mathbb{N}$ of a transition $\tau \in T$ to infer a bounded-unless formula:

$$\begin{array}{l}
 \text{U-SS} \quad p \rightarrow \neg \text{enabled}(\tau) \\
 \quad p \rightarrow \varphi \\
 \quad \{\varphi\} T - \tau \{\varphi\} \\
 \quad \varphi \rightarrow q \\
 \quad \frac{(\varphi \wedge \text{enabled}(\tau)) \rightarrow \tau}{p \rightarrow q U_{\geq l_\tau} \tau}
 \end{array}$$

The rule U-SS derives a *temporal* (bounded-unless) formula from premises all of which are *state* formulas, whose S -validity is usually shown by proving them generally valid. The state formula φ is called the *invariant* of the rule. Choosing τ to be *true*, the rule infers a bounded-invariance property,

$$p \rightarrow \square_{< l_\tau} q$$

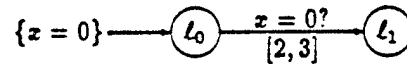
(note that the last premise holds trivially in this case). To see why the rule U-SS is S -sound, observe that whenever the transition τ is not enabled, it cannot be taken for at least l_τ time units.

The *single-step upper-bound* rule uses the maximal delay $u_\tau \in \mathbb{N}$ of a transition $\tau \in T$ to infer a bounded-response formula:

$$\begin{array}{l}
 \diamond\text{-SS} \quad p \rightarrow (\varphi \vee q) \\
 \varphi \rightarrow \text{enabled}(\tau) \\
 \{\varphi\} \tau - \tau \{\varphi \vee q\} \\
 \hline
 \{\varphi\} \tau \{q\} \\
 \hline
 p \rightarrow \diamond_{\leq u_\tau} q
 \end{array}$$

This rule derives a *temporal* bounded-response formula from premises all of which are *state* formulas. The state formula φ is again called the *invariant* of the rule. To see why the rule $\diamond\text{-SS}$ is *S*-sound, recall that the transition τ has to be taken before it would be continuously enabled for more than u_τ time units.

To demonstrate a typical application of the single-step rules, we consider again the single-process system P with the data precondition $x = 0$ and the following timed transition diagram:



The process P confirms that $x = 0$ and proceeds to the location ℓ_1 . Because of the delay interval $[2, 3]$ of the transition $\tau_{0 \rightarrow 1}$, the final location ℓ_1 cannot be reached before time 2 and must be reached by time 3. Using single-step reasoning, we can carry out a formal proof of this analysis. The bounded-invariance property that P does not terminate before time 2,

$$\text{ready} \rightarrow \square_{< 2} \neg \text{at } \ell_1,$$

is established by an application of the single-step lower-bound rule U-SS with respect to the transition $\tau_{0 \rightarrow 1}$ (let the invariant φ be $\text{at } \ell_{1,0}$). The bounded-response property that P terminates by time 3,

$$\text{start} \rightarrow \diamond_{\leq 3} \text{at } \ell_1,$$

follows from the single-step upper-bound rule $\diamond\text{-SS}$ with respect to the transition $\tau_{0 \rightarrow 1}$ (use the invariant $\text{at } \ell_0 \wedge x = 0$).

Transitivity rules

To join a finite number of successive real-time constraints into a more complicated real-time property, we introduce transitivity rules. The *transitive lower-bound* rule combines two bounded-unless formulas:

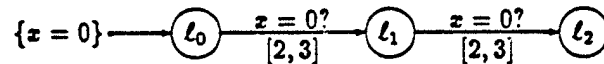
$$\text{U-TRANS} \quad \frac{\begin{array}{l} \phi \rightarrow p U_{\geq l_1} \chi \\ \chi \rightarrow q U_{\geq l_2} \psi \end{array}}{\phi \rightarrow (p \vee q) U_{\geq l_1 + l_2} \psi}$$

We refer to the formula χ as the *link* of the rule. The *transitive upper-bound* rule combines two bounded-response formulas:

$$\text{◇-TRANS} \quad \frac{\begin{array}{l} \phi \rightarrow \text{◇}_{\leq u_1} \chi \\ \chi \rightarrow \text{◇}_{\leq u_2} \psi \end{array}}{\phi \rightarrow \text{◇}_{\leq u_1 + u_2} \psi}$$

The formula χ is again called the *link* of the rule. Both transitivity rules are easily seen to be generally sound as well as *S*-sound for every timed transition system *S*.

We demonstrate the application of the transitivity rules by examining the single-process system *P* with the following timed transition diagram:



We want to show that *P* terminates not before time 4 and not after time 6. First, we prove the lower bound on the termination of *P*:

$$\text{ready} \rightarrow \square_{<4} \neg \text{at } l_2.$$

By the transitive lower-bound rule U-TRANS, it suffices to show the two premises

$$\text{ready} \rightarrow (\neg \text{at } l_2) U_{\geq 2} \text{at } l_0, \quad (1)$$

$$\text{at } l_0 \rightarrow (\neg \text{at } l_2) U_{\geq 2} \text{true}. \quad (2)$$

Both premises can be established by single-step lower-bound reasoning. To show the premise (1), we apply the rule U-SS with respect to the transition $\tau_{0 \rightarrow 1}$, using the invariant $at.L_{1,0}$; the premise (2) follows from the rule U-SS with respect to the transition $\tau_{1 \rightarrow 2}$ and the invariant $at.L_{0,1}$.

The upper bound on the termination of P ,

$$start \rightarrow \Diamond_{\leq 6} at.L_2,$$

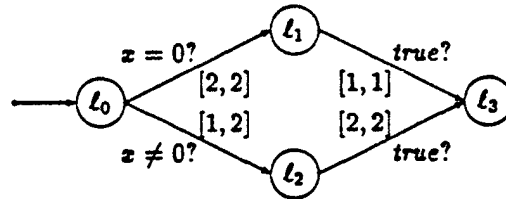
is concluded by the transitive upper-bound rule \Diamond -TRANS. It suffices to show the premises

$$start \rightarrow \Diamond_{\leq 3} (at.L_1 \wedge x = 0),$$

$$(at.L_1 \wedge x = 0) \rightarrow \Diamond_{\leq 3} at.L_2,$$

both of which can be established by single-step upper-bound reasoning (use the invariants $at.L_0 \wedge x = 0$ and $at.L_1 \wedge x = 0$, respectively). Note that for lower-bound reasoning the link $at.L_0$ identifies the last state *before* the transition $\tau_{0 \rightarrow 1}$ is taken, while for upper-bound reasoning the link $at.L_1 \wedge x = 0$ refers to the first state *after* $\tau_{0 \rightarrow 1}$ is taken.

For an example with a (deterministic) branching structure, consider the process P' with the following timed transition diagram:



We show that P' terminates either at time 3 or at time 4. The proof requires a case analysis on the initial value of x , which determines which path of the transition diagram is taken. The lower bound

$$ready \rightarrow \Box_{< 3} \neg at.L_3$$

is implied by the two bounded-invariance formulas

$$(ready \wedge x = 0) \rightarrow \Box_{< 3} \neg at.L_3,$$

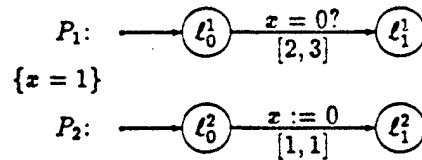
$$(ready \wedge x \neq 0) \rightarrow \Box_{< 3} \neg at.L_3,$$

both of which can be derived by transitive lower-bound reasoning (as links use the two state formulas $at_{\perp,0} \wedge x = 0$ and $at_{\perp,0} \wedge x \neq 0$, respectively). The upper bound

$$start \rightarrow \Diamond_{\leq 4} at_{\mathcal{L}_3}$$

follows by a similar case analysis and transitive upper-bound reasoning.

So far we have examined only single-process examples. In general, several processes that communicate through shared variables interfere with each other. Consider the synchronous two-process shared-variables multiprocessing system with the data precondition $x = 1$ and the following timed transition diagrams:



The first process, P_1 , is identical to a previous example; with a minimal delay of 2 time units and a maximal delay of 3 time units, it confirms that $x = 0$ and proceeds to the location \mathcal{L}_1^1 . However, this time the value of x is not 0 from the very beginning, but set to 0 by the second process, P_2 , only at time 1. Thus, P_1 can reach its final location \mathcal{L}_1^1 no earlier than at time 3 and no later than at time 4.

For a formal proof we need the transitivity rules. The bounded-invariance property

$$ready \rightarrow \Box_{<3} \neg at_{\mathcal{L}_1^1}$$

is established by an application of the transitive lower-bound rule U-TRANS. It suffices to show the premises

$$ready \rightarrow (\neg at_{\mathcal{L}_1^1}) U_{\geq 1} (at_{\perp,0} \wedge x = 1),$$

$$(at_{\perp,0} \wedge x = 1) \rightarrow (\neg at_{\mathcal{L}_1^1}) U_{\geq 2} true,$$

both of which follow from single-step lower-bound reasoning. Similarly, the transitive upper-bound rule \Diamond -TRANS is used to show the bounded-response property

$$start \rightarrow \Diamond_{\leq 4} at_{\mathcal{L}_1^1}$$

from the link $at_{\mathcal{L}_0^1} \wedge x = 0$.

Induction rules

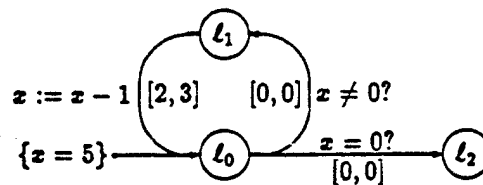
To prove lower and upper bounds on the execution time of program loops, we need to combine a state-dependent number of bounded-invariance or bounded-response properties. For this purpose it is economical to have induction schemes.

The *inductive lower-bound* rule U-IND generalizes the transitive lower-bound rule U-TRANS; it combines a potentially large number of similar bounded-unless formulas in a single proof step. Assume that the new, rigid variable $i \in V$ ranges over the natural numbers N ; for any $n \in N$:

$$\text{U-IND} \quad \frac{(\varphi(i) \wedge i > 0) \rightarrow p U_{\geq 1} \varphi(i-1)}{\varphi(n) \rightarrow p U_{\geq n-1} \varphi(0)}$$

By $\varphi(i-1)$ we denote the state formula that results from the inductive invariant $\varphi(i)$ by replacing all occurrences of the variable i with the expression $i-1$; the formulas $\varphi(n)$ and $\varphi(0)$ are obtained analogously. Note that every instance of the rule U-IND, for any constant $n \in N$, is derivable from the transitive lower-bound rule U-TRANS.

For a demonstration of inductive lower-bound reasoning, we consider the following single-process system P :



The process P decrements the value of x until it is 0, at which point P proceeds to the location L_2 . Since x starts out with the value 5, and each decrement operation takes at least 2 time units, while the tests are instantaneous, the final location L_2 cannot be reached before time 10. This lower bound,

$$\text{ready} \rightarrow \square_{<10} \neg \text{at } L_2,$$

follows by transitivity and monotonicity from the two bounded-unless properties

$$\text{ready} \rightarrow (\neg \text{at } L_2) U_{\geq 2} (\text{at } L_1 \wedge x = 5), \quad (1)$$

$$(at_{L_1} \wedge z = 5) \rightarrow (\neg at_{L_2}) U_{\geq 8} (at_{L_1} \wedge z = 1). \quad (2)$$

The first property, (1), is enforced by two single-step lower bounds; the second property, (2), can be derived by the inductive lower-bound rule U-IND from the premise

$$(at_{L_1} \wedge z = i + 1 \wedge i > 0) \rightarrow (\neg at_{L_2}) U_{\geq 2} (at_{L_1} \wedge z = i),$$

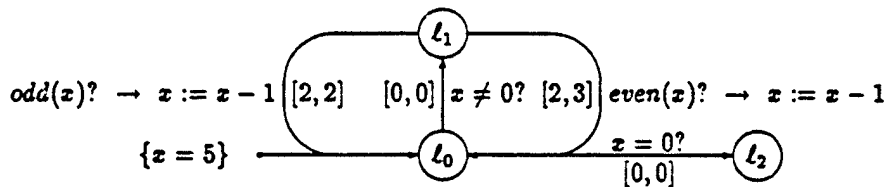
which is concluded by transitive reasoning.

The inductive lower-bound rule has a twin that combines several similar bounded-response formulas by adding up their upper bounds u . In fact, both induction rules can be generalized, by letting the bounds l and u vary as functions of i . In its more general form, we state only the *inductive upper-bound* rule. It uses again a new, rigid variable $i \in V$ that ranges over the natural numbers N ; for any $n \in N$:

$$\boxed{\begin{array}{c} \diamond\text{-IND} \quad \frac{(\varphi(i) \wedge i > 0) \rightarrow \diamond_{\leq u_i} \varphi(i-1)}{\varphi(n) \rightarrow \diamond_{\leq \sum_{0 < i \leq n} u_i} \varphi(0)} \end{array}}$$

Every instance of this rule is derivable from the transitive upper-bound rule $\diamond\text{-TRANS}$.

The general form of the inductive upper-bound rule is useful to prove upper bounds for programs with loops whose execution time is not uniform. An example for such a system is the following *odd-even* variant of the process P :



The upper bound

$$start \rightarrow \diamond_{\leq 12} at_{L_2}$$

follows by single-step reasoning and transitivity from the bounded-response property

$$start \rightarrow \diamond_{\leq 12} (at_{L_0} \wedge z = 0),$$

which can be concluded by the inductive upper-bound rule $\diamond\text{-IND}$ from the premise

$$(at_{L_0} \wedge z = i \wedge i > 0) \rightarrow \diamond_{\leq 2 + even(i)} (at_{L_0} \wedge z = i - 1)$$

(the expression $even(i)$ evaluates to either 1 or 0 depending on whether the value of i is even). This bounded-response formula follows from transitive reasoning.

Crossover rules

So far we have presented only proof rules that combine properties of the same type. However, as the *increment-decrement* example of Subsection 6.2.2 has demonstrated, the addition of delays of the same type is insufficient for deriving all bounded-invariance and bounded-response properties of a timed transition system: both local lower bounds and upper bounds may jointly affect a global bounded-invariance (or bounded-response) property.

Specifically, to mimic the informal argument for the timely termination of the *increment-decrement* system by a bounded-operator proof, we need the *crossover upper-bound* rule:

$$\begin{array}{l}
 \diamond\text{-MIX} \quad u < l \\
 p \rightarrow \diamond_{\leq u} \hat{p} \\
 q \rightarrow \square_{< l} \hat{q} \\
 \{p\}(\mathcal{T} - \tau_I)^- \{q\} \\
 \hline
 p \rightarrow \diamond_{\leq u} (\hat{p} \wedge \hat{q})
 \end{array}$$

This rule is a modification of the temporal formula

$$(\diamond_{\leq u} \hat{p} \wedge \square_{< l} \hat{q}) \rightarrow \diamond_{\leq u} (\hat{p} \wedge \hat{q}),$$

which is valid if $u < l$. We need the more complicated rule because the reasoning about about lower and upper bounds is asymmetric: while bounded-invariance formulas refer, intuitively, to the last state before a transition is taken, bounded-response formulas refer to the first state after a transition was taken. This phenomenon is captured by the inverse verification condition

$$\{p\}(\mathcal{T} - \tau_I)^- \{q\},$$

which asserts that, in any run fragment (σ, T) of S , if σ_{i+1} is a p -state and $\sigma_{i+1} \neq \sigma_i$, then σ_i is a q -state; note that, in this case, $T_{i+1} = T_i$, because all run fragments are deterministic timed state sequences. Also observe that for any state σ_i in a run fragment such that σ_i falsifies the ready condition, there is a run that contains a predecessor state that is different from σ_i . The S -soundness of the rule $\diamond\text{-MIX}$ follows.

We give here only a brief sketch of the bounded-operator proof for the bounded-response property

$$start \rightarrow \Diamond_{\leq 130} at_{\mathcal{L}_4^1}$$

of the *increment-decrement* system. The derivation relies, as expected, on an interplay of lower-bound and upper-bound rules. First we show that within 10 time units P_1 can increase the value of y at most to 10:

$$ready \rightarrow \Box_{< 11} (y \leq 10);$$

this is done by inductive lower-bound reasoning. Then we apply the crossover upper-bound rule \Diamond -MIX to the single-step upper bound

$$start \rightarrow \Diamond_{\leq 10} (at_{\mathcal{L}_1^2} \wedge x = 0),$$

thus obtaining the bounded-response property

$$start \rightarrow \Diamond_{\leq 10} (y \leq 10 \wedge at_{\mathcal{L}_1^2} \wedge x = 0).$$

From here we proceed by pure upper-bound reasoning, performing a case analysis on the locations of P_1 .

The *increment-decrement* example illustrates the trade-off between bounded-operator reasoning and explicit-clock reasoning beautifully. Compare the two proofs of the upper bound on termination: while the bounded-operator (or “hidden-clock”) style of real-time verification of this section refers to time only through the relative offsets of time-constrained temporal operators, the explicit-clock style of the previous section uses ordinary untimed temporal operators and refers to the absolute time in state formulas. Both styles trade off the complexity of the temporal proof structure against the complexity of the state invariants:

- The *hidden-clock* approach relies on complex proof structures similar to the proof lattices for establishing ordinary (untimed) *liveness* properties [105, 90] and uses relatively simple *local* invariants.
- The *explicit-clock* method employs only the plain *unless* rule — an (untimed) *safety* rule — but requires a powerful *global* invariant.

While the crossover upper-bound rule combines a bounded-invariance property and a bounded-response property into a bounded-response property, its counterpart, the *crossover lower-bound* rule, yields a bounded-unless property:

U-MIX $u < l$ $p \rightarrow \Box_{<l} \hat{p}$ $q \rightarrow \Diamond_{\leq u} \hat{q}$ $\{p\}T - \tau_l \{q\}$ $\frac{\hat{q} \rightarrow \hat{q}U(\hat{q} \wedge \neg \hat{p})}{p \rightarrow \hat{p}U_{\geq l} \hat{q}}$

This rule is S -sound, because it originates with the valid temporal formula (for $u < l$)

$$(\Box_{<l} \hat{p} \wedge \Diamond_{\leq u} (\hat{q}U(\hat{q} \wedge \neg \hat{p}))) \rightarrow \hat{p}U_{\geq l} \hat{q}.$$

Note that the last premise, which contains only an unbounded *unless* operator, can be established by untimed reasoning. In the following subsection, we will move further into this direction by delegating as much of the derivation of timing properties as possible to conventional untimed proof systems.

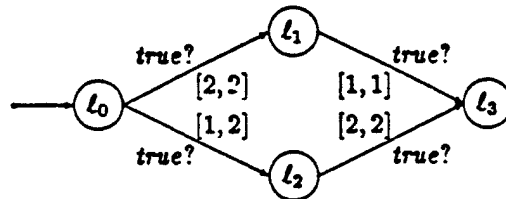
The crossover lower-bound rule U-MIX can be used to derive the lower bound

$$\text{ready} \rightarrow \Box_{<2} \neg \text{at}_4^1$$

of the *increment-decrement* system.

6.3.2 Conditional rules

Unfortunately, the proof rules we have designed are not strong enough to show tight bounds on *nondeterministic* systems. To see this, consider the following nondeterministic variant P of a process encountered previously:



As before, P terminates either at time 3 or at time 4. However, during an execution of P , one of the two transitions $\tau_{0 \rightarrow 1}$ and $\tau_{0 \rightarrow 2}$ is chosen nondeterministically. Thus we cannot

carry out a case analysis with respect to a state formula that selects a unique guard. Instead, we proceed in two steps. First we establish an untimed safety formula that enumerates all possible nondeterministic choices. Then we decorate the unbounded temporal formula with time bounds.

Step 1 To establish the S -validity of a temporal formula ϕ that contains only unbounded *unless* operators (i.e., $U_{\geq 0}$), it suffices to show that ϕ is true over all run fragments of the untimed transition system S^- that underlies S . This can be achieved with the help of any conventional untimed proof system (for instance, the proof system of Manna and Pnueli [91]).

For example, to derive the lower bound 3 on the termination of our example P , we show the untimed formula

$$\text{ready} \rightarrow ((\text{at } \mathcal{L}_1 U^- \text{ at } \mathcal{L}_0 U^+ \text{ at } \mathcal{L}_1) \vee (\text{at } \mathcal{L}_1 U^+ \text{ at } \mathcal{L}_0 U^+ \text{ at } \mathcal{L}_2)) \quad (\dagger)$$

(nested *unless* operators associate to the right).

Step 2 To add time bounds to this disjunction of nested *unless* formulas, we need *conditional* single-step rules. They establish single-step real-time bounds under the assumption that a particular disjunct has been chosen. The time bounds can, then, be combined by the transitivity rules and *conditional* crossover rules.

Nondeterministic lower bounds

The *conditional single-step lower-bound* rule uses the minimal delay $L_\tau \in \mathbb{N}$ of a transition $\tau \in \mathcal{T}$:

$\text{U-CSS} \quad \frac{\begin{array}{l} p \rightarrow \neg \text{enabled}(\tau) \\ \{q\} \mathcal{T} - \tau \{q \vee \neg r\} \end{array}}{(p U_{\geq L_\tau}^+ q U^+ (r \wedge \phi)) \rightarrow (p U_{\geq L_\tau}^+ q U_{\geq L_\tau}^+ (r \wedge \phi))}$

The rule U-CSS is S -sound for any temporal formula ϕ .

In our example, we use the conditional single-step lower-bound rule U-CSS with respect to the transitions $\tau_{0 \rightarrow 1}$ and $\tau_{0 \rightarrow 2}$ to derive the conditional single-step bounds

$$(\text{at } \mathcal{L}_1 U^+ \text{ at } \mathcal{L}_0 U^+ \text{ at } \mathcal{L}_1) \rightarrow (\text{at } \mathcal{L}_1 U^+ \text{ at } \mathcal{L}_0 U_{\geq 2}^+ \text{ at } \mathcal{L}_1).$$

$$(at_{L_1} U^+ at_{L_0} U^+ at_{L_2}) \rightarrow (at_{L_1} U^+ at_{L_0} U_{\geq 1}^+ at_{L_2}).$$

They allow us to conclude, from (†),

$$ready \rightarrow ((at_{L_1} U^+ at_{L_0} U_{\geq 2}^+ at_{L_1}) \vee (at_{L_1} U^+ at_{L_0} U_{\geq 1}^+ at_{L_2})). \quad (\ddagger)$$

To collapse nested *bounded-unless* operators, we use the temporal formula U-COLL:

$$\boxed{\text{U-COLL} \quad (p U_{\geq l_1} q U_{\geq l_2} \phi) \rightarrow ((p \vee q) U_{\geq l_1 + l_2} \phi)}$$

Note that this temporal formula, which is generally valid, can be derived by from transitive lower-bound rule U-TRANS by using the two tautologies

$$(p U_{\geq l_1} q U_{\geq l_2} \phi) \rightarrow (p U_{\geq l_1} q U_{\geq l_2} \phi),$$

$$(q U_{\geq l_2} \phi) \rightarrow (q U_{\geq l_2} \phi).$$

From (‡) we obtain by collapsing and monotonicity

$$ready \rightarrow ((at_{L_0} U_{\geq 2}^+ at_{L_1}) \vee (at_{L_0} U_{\geq 1}^+ at_{L_2}));$$

that is, using the (untimed) validity $p \rightarrow p U p$ and monotonicity,

$$ready \rightarrow ((at_{L_0} U_{\geq 2}^+ at_{L_1} U^+ at_{L_1}) \vee (at_{L_0} U_{\geq 1}^+ at_{L_2} U^+ at_{L_2})).$$

Adding conditional single-step lower bounds for the transitions $\tau_{1 \rightarrow 3}$ and $\tau_{2 \rightarrow 3}$ gives

$$ready \rightarrow ((at_{L_0} U_{\geq 2}^+ at_{L_1} U_{\geq 1}^+ at_{L_1}) \vee (at_{L_0} U_{\geq 1}^+ at_{L_2} U_{\geq 2}^+ at_{L_2})),$$

and by collapsing and monotonicity we finally arrive at the desired bounded-invariance property

$$ready \rightarrow \Box_{<3} \neg at_{L_3}.$$

Nondeterministic upper bounds

Conditional upper-bound reasoning does not require the nesting of *unless* operators. The *conditional single-step upper-bound* rule uses the maximal delay $\nu_\tau \in \mathbb{N}$ of a transition $\tau \in \mathcal{T}$:

$$\boxed{\begin{array}{l} \diamond\text{-CSS } p \rightarrow \text{enabled}(\tau) \\ \frac{\{p\} \tau \{-p\}}{(p \cup \phi) \rightarrow \diamond_{\leq u, \tau} \phi} \end{array}}$$

Clearly, the rule $\diamond\text{-CSS}$ is S -sound for any temporal formula ϕ . Note that the second premise of this rule is trivially valid if τ becomes disabled by being taken, as is the case for all transitions of a timed transition system that is given by timed transition diagrams (recall that we have ruled out self-loops in transition diagrams). It is also worth pointing out that both conditional lower-bound and conditional upper-bound reasoning rely only on assumptions that are built only from state formulas by positive boolean connectives and unbounded *unless* operators and, therefore, define untimed safety properties. Accordingly, the first step of conditional reasoning can be carried out by any untimed method for deriving safety properties.

To derive the upper bound 4 on the termination of our example P , we show first the untimed formula

$$\text{start} \rightarrow ((\text{at}_{L_0} \cup \text{at}_{L_1}) \vee (\text{at}_{L_0} \cup \text{at}_{L_2})).$$

By the conditional single-step upper-bound rule $\diamond\text{-CSS}$ with respect to the transitions $\tau_{0 \rightarrow 1}$ and $\tau_{0 \rightarrow 2}$, we derive the conditional single-step bounds

$$(\text{at}_{L_0} \cup \text{at}_{L_1}) \rightarrow \diamond_{\leq 2} \text{at}_{L_1},$$

$$(\text{at}_{L_0} \cup \text{at}_{L_2}) \rightarrow \diamond_{\leq 2} \text{at}_{L_2}.$$

They allow us to conclude

$$\text{start} \rightarrow (\diamond_{\leq 2} \text{at}_{L_1} \vee \diamond_{\leq 2} \text{at}_{L_2}).$$

Now we can proceed by *unconditional* upper-bound reasoning to arrive at the desired bounded-response property

$$\text{start} \rightarrow \diamond_{\leq 4} \text{at}_{L_3}.$$

Conditional crossover reasoning

For combining assumptions that contain both lower bounds and upper bounds, we need the *conditional crossover lower-bound* rule U-CMIX and the *conditional crossover upper-bound* rule $\diamond\text{-CMIX}$:

$$\text{U-CMIX} \frac{u \leq l \quad \phi \rightarrow (\neg p \wedge \neg q)}{(((p \vee q) U_{\geq l} \phi) \wedge (p U ((q U \phi) \wedge \Diamond_{\leq u} \phi))) \rightarrow (p U_{\geq l-u} q U \phi)}$$

$$\Diamond\text{-CMIX} \frac{u \leq l \quad \phi \rightarrow (\neg p \wedge \neg q)}{(\Diamond_{\leq u} \phi \wedge (p U q U_{\geq l} \phi)) \rightarrow \Diamond_{\leq u-l} (q U \phi)}$$

Both crossover rules, which really are schemas of valid temporal formulas, will be essential for the proof of relative completeness of conditional bounded-operator reasoning.

6.3.3 Relative completeness

Suppose we are given an untimed proof system that is complete for nested unless formulas. Although such a proof system cannot exist for most data domains, there are temporal proof systems that are complete relative to state reasoning [92]. Assuming that all untimed safety properties of the given timed transition system S can be derived, we prove two results about the power of bounded-operator reasoning:

1. First, we use the simplifying assumption that the nontrivial timing constraints of S are either all minimal delays or all maximal delays. This case does not require crossover reasoning, and our proof system (without crossover rules) can indeed derive every bounded-invariance and bounded-response property of S . We prove relative completeness by constructing a "constraint pattern" of transition delays for any given property. This technique can be generalized to the following case.
2. Secondly, we show that our proof system (with crossover rules) can derive every bounded-invariance and bounded-response property that satisfies, relative to the given timed transition system S , the sufficient condition of *stability*. Roughly speaking, a temporal formula ϕ will be called stable for S iff the truth of ϕ at any state along a run of S can be determined without any information about the history of the run.

In addition, we will argue that bounded-operator reasoning, as we have presented it, is strictly less powerful than explicit-clock reasoning and cannot be used to derive every bounded-invariance and bounded-response property of S .

Theorem 6.2 (Relative completeness of bounded-operator reasoning) (The unidirectional case) *Let $S = \langle \Sigma, \Theta, \mathcal{T}, l, u \rangle$ be an operational timed transition system such that either $l_\tau = 0$ for all $\tau \in \mathcal{T}$ or $u_\tau = \infty$ for all $\tau \in \mathcal{T}$. Let ϕ be a bounded-invariance or a bounded-response formula. If ϕ is S -valid, then it can be derived by the monotonicity, transitivity, and conditional single-step rules relative to untimed safety reasoning.*

Proof of Theorem 6.2 (1) Suppose that all maximal delays of S are ∞ . First we observe that, under the given restrictions, untimed reasoning is complete for untimed properties of S . This is because in the absence of finite maximal delays there is a time sequence T for every run fragment σ of the untimed weakly-fair transition system $S_{\bar{w}}$ that underlies S such that (σ, T) is a run fragment of S (choose all time steps large enough). It follows that any untimed temporal formula that is S -valid is also $S_{\bar{w}}$ -valid and, thus, can be established by untimed reasoning.

Any bounded-response property is either trivially not S -valid or can be established by untimed reasoning. Now suppose that the bounded-invariance property

$$p \rightarrow \Box_{<l} \neg q \quad (1)$$

is S -valid; we show that it can be derived within our proof system. The main idea is to see that in order for (1) to be valid, for any p -state in a run of S there has to be a sequence of nonoverlapping single-step lower bounds that add up to at least l before a q -state can be reached. We show that there are only finitely many such ways in which a q -state can be delayed for l time units; hence they can be enumerated by a single untimed formula.

Consider an arbitrary run fragment $\rho = (\sigma, T)$ of S such that σ_i , $i \geq 0$, is a p -state. Let σ_j be the first q -state with $j \geq i$; if no such state exists, let $j = \infty$. We write τ_k for the transition that is completed at position $k \geq 0$ of ρ . A lower-bound l -constraint pattern for $\sigma_{i..j}$ is a finite sequence of nonoverlapping single-step lower bounds between i and j that add up to at least l . Formally, a constraint pattern C is a sequence of transitions $\tau_{i_1}, \dots, \tau_{i_n}$. The pattern C is a lower-bound l -constraint pattern iff

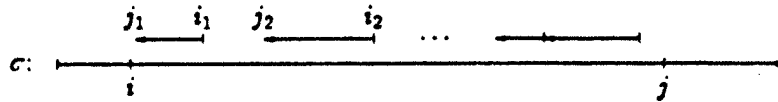
$$\sum_{1 \leq k \leq n} l_{\tau_{i_k}} \geq l;$$

it is a lower-bound constraint pattern for $\sigma_{i..j}$ iff

$$(a) \ i = i_0 < i_1 < \dots < i_n \leq j \text{ and}$$

- (b) for all $1 \leq k \leq n$, the transition τ_{i_k} is not enabled on some state σ_{j_k} such that $i_{k-1} \leq j_k < i_k$.

A lower-bound constraint pattern for $\sigma_{i..j}$ can be visualized by annotating the run fragment ρ with backward arrows that represent single-step lower bounds:



Two constraint patterns are equivalent iff one is a subpattern of the other (i.e., can be obtained by omitting transitions). It is not hard to show the following two properties of lower-bound constraint patterns:

Property A There is a lower-bound l -constraint pattern for $\sigma_{i..j}$ (use the truth of (1) over the i -th suffix of ρ).

Property B There are only finitely many different equivalence classes of lower-bound l -constraint patterns.

We add, for every transition $\tau \in \mathcal{T}$, the boolean variable $completed_\tau$ to our language; it is intended to be true in a state σ_i , $i \geq 0$, of a run $\rho = (\sigma, \tau)$ iff the transition τ is completed at position i of ρ . For our purpose, it turns out to be sufficient that $completed_\tau$ satisfies the two axioms

$$\begin{aligned} & \{true\} \tau \{completed_\tau\}, \\ & \{true\} \mathcal{T} - \tau \{\neg completed_\tau\}. \end{aligned} \quad (†)$$

By Property A, there is an untimed formula of the form

$$\begin{aligned} & (\neg q) \cup (\neg q \wedge \neg enabled(\tau_{i_1})) \cup^+ (\neg q) \cup^+ (\neg q \wedge completed_{\tau_{i_1}}) \cup^+ \dots \\ & \cup^+ (\neg q \wedge completed_{\tau_{i_n}}) \end{aligned}$$

that is true over the i -th suffix of ρ . Since there are, by Property B, only finitely many formulas of this form, $p \rightarrow \psi$ for some finite disjunction ψ of nested unless formulas is S -valid and, thus, given by untimed reasoning. From (†) we infer by the conditional single-step lower-bound rule U-CSS with respect to any transition $\tau \in \mathcal{T}$ that

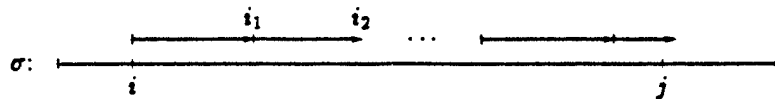
$$\begin{aligned} & (\neg enabled(\tau)) \cup_{\geq l}^+ \varphi \cup^+ (completed_\tau \wedge \phi) \rightarrow \\ & (\neg enabled(\tau)) \cup_{\geq l}^+ \varphi \cup_{\geq l}^+ (completed_\tau \wedge \phi) \end{aligned}$$

for any state formula φ and temporal formula ϕ . Hence we can decorate the untimed nested unless formula with time bounds. By repeated collapsing and monotonicity similar to the sample lower-bound derivation above, we arrive at the desired bounded-invariance property (1).

(2) Now suppose that all minimal delays of S are 0; the proof proceeds similarly to the previous case. Untimed reasoning is complete for untimed properties of S , because S is operational. Any bounded-invariance property is either trivially not S -valid or can be established by untimed reasoning. So let us assume that the bounded-response property

$$p \rightarrow \Diamond_{\leq u} q \quad (2)$$

is S -valid. Consequently, every p -state in a run of S has to be followed by a q -state that can be reached by a sequence of overlapping single-step upper bounds that add up to at most u . We visualize single-step upper bounds by forward arrows:



Formally, let $\rho = (\sigma, T)$ be a run fragment of S such that $\sigma_i, i \geq 0$, is a p -state, and let σ_j be the first q -state with $j \geq i$. For the sake of simplicity, we assume that the transition τ_k , which is completed at the position $k \geq 0$ of ρ , is not enabled on σ_k (otherwise split τ_k into two identical transitions with different names). A constraint pattern τ_1, \dots, τ_n is an upper-bound u -constraint pattern iff

$$\sum_{1 \leq k \leq n} u_{\tau_k} \leq u;$$

it is an upper-bound constraint pattern for $\sigma_{i..j}$ iff

- (a) $i = i_0 < i_1 < \dots < i_{n-1} < j \leq i_n$ and
- (b) for all $1 \leq k \leq n$, the transition τ_k is enabled but not completed at all states σ_{j_k} such that $i_{k-1} \leq j_k < i_k$.

It is not hard to see that upper-bound constraint patterns also satisfy the two crucial properties:

Property A There is an upper-bound u -constraint pattern for $\sigma_{i..j}$ (use the truth of (2) over the i -th suffix of ρ).

Property B There are only finitely many different equivalence classes of upper-bound u -constraint patterns (use the operationality of S).

By Property A, there is an untimed formula of the form

$$(enabled(\tau_{i_1}) \wedge \neg completed_{\tau_{i_1}}) \cup (enabled(\tau_{i_2}) \wedge \neg completed_{\tau_{i_2}}) \cup \dots \\ \cup (enabled(\tau_{i_n}) \wedge \neg completed_{\tau_{i_n}}) \cup q$$

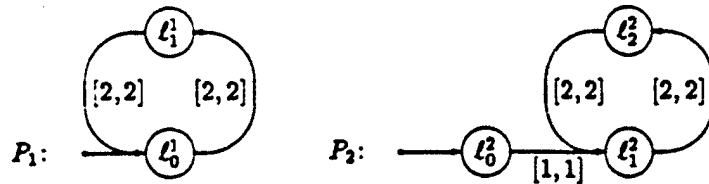
that is true over the i -th suffix of ρ . By Property B, there is again a finite disjunction ψ of nested unless formulas such that the implication $p \rightarrow \psi$ is S -valid and, therefore, given by untimed reasoning. By repeated application of the conditional single-step upper-bound rule \diamond -CSS, transitivity, and monotonicity, we arrive at the desired bounded-response property (2). More specifically, to collapse nested *bounded-eventually* operators, we can use the valid temporal formula \diamond -COLL, which is derivable from the transitive upper-bound rule \diamond -TRANS:

$$\boxed{\diamond\text{-COLL} \quad (\diamond_{\leq u_1} \diamond_{\leq u_2} \phi) \rightarrow \diamond_{\leq u_1+u_2} \phi}$$

■

The need for history information

Now let us, as promised, generalize the relative completeness result to timed transition systems that contain both nontrivial minimal delays and nontrivial maximal delays. To illustrate some of the complications that arise in the general case, we consider the synchronous two-process multiprocessing system with the following timed transition diagram definition:



Both processes of this system operate perfectly synchronous, at the exact same speed, but they are "out of phase," because the second process, P_2 , starts to loop only 1 time unit after the first process, P_1 . For instance, the bounded-response formula

$$at.L_0^1 \rightarrow \diamond_{\leq 1} at.L_1^2 \quad (\dagger)$$

is S -valid for the timed transition system S that is associated with the concrete system $P_1 \parallel P_2$. A proof of the bounded-response property (\dagger) in the explicit-clock style of reasoning uses the delay counters of transitions in an essential way. Since we have, in the version of bounded-operator reasoning that we have presented, no way of referring to the history of a state in a run, we cannot derive the property (\dagger) . This deficiency can, of course, be remedied by introducing a mechanism whose expressiveness is equivalent to explicit delay counters. One option that preserves the flavor of bounded-operator reasoning is the addition of time-constrained *past* temporal operators for referring to the immediate history of a state in a run of a system. Here we choose not to pursue this extension, but rather prove the relative completeness of bounded-operator reasoning for a restricted set of verification problems only. For this purpose, we define the notions of lower-bound and upper-bound stability.

Stable properties

A state formula p is called *lower-bound-stable* for the timed transition system S iff for every run (σ, T) of S and all $i \geq 0$,

if σ_i is a p -state and σ_{i+1} is not a p -state and the transition $\tau \in T$ is enabled on both σ_i and σ_{i+1} , then $L_\tau = 0$.

The formula p is *upper-bound-stable* for S iff for every run (σ, T) of S and all $i \geq 0$,

σ_0 is not a p -state, and if σ_{i+1} is a p -state and σ_i is not a p -state and the transition $\tau \in T$ is enabled on both σ_i and σ_{i+1} , then $u_\tau = \infty$.

For instance, the ready condition *ready* is lower-bound-stable for every timed transition system S and the synchronous starting condition *start* is upper-bound-stable for every system S . If the timed transition system S is associated with a single-process system, then every state formula of the form $at.L_i$ is both lower-bound stable and upper-bound stable

(if $i \neq \perp$) for S . Clearly, if p is lower-bound-stable (or upper-bound-stable) for any timed transition system, then so is the stronger assertion $p \wedge q$.

Given a timed transition system S , we say that a bounded-invariance property

$$p \rightarrow \Box_{<l} q$$

is *stable* for S iff the antecedent p is lower-bound stable for S . Similarly, a bounded-response property

$$p \rightarrow \Diamond_{\leq u} q$$

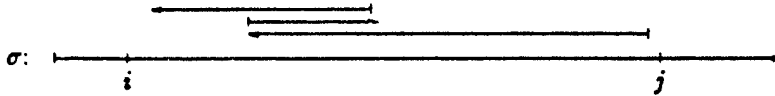
is *stable* for S iff its antecedent p is upper-bound stable for S . We will show that every stable bounded-invariance and bounded-response property of S can be derived by bounded-operator reasoning. As we have just seen, this includes all bounded-invariance and bounded-response properties with the antecedents *ready* and *start*, respectively, as well as all properties of *single-process* systems whose antecedents contain a conjunct of the form $at.L_i$, for any location L_i . The trouble with the bounded-response property (†) of the *two-process* system given above is that its antecedent $at.L_0^1$ is not upper-bound stable: a transition of the process P_1 may enter or leave the location L_0^1 while a transition of the competing process P_2 is enabled and counting towards its maximal delay of 2.

We show that stability is a sufficient condition for relative completeness. In fact, as the following argument will reveal, our definition of stability has been motivated by the attempt to generalize the constraint pattern technique that we used to prove Theorem 6.2.

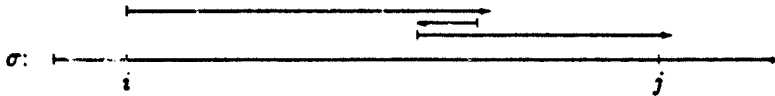
Theorem 6.3 (Relative completeness of bounded-operator reasoning) (The bidirectional case) *Let S be an operational timed transition system and let ϕ be a bounded-invariance or a bounded-response formula that is stable for S . If ϕ is S -valid, then it can be derived by the monotonicity, transitivity, conditional single-step, and conditional crossover rules relative to untimed safety reasoning.*

Proof of Theorem 6.3 We give only a brief sketch of the proof, which parallels the proof of Theorem 6.2. Suppose we wish to derive the S -valid stable property ϕ . In the presence of both minimal and maximal delays, constraint patterns are not linear, but resemble, in general, zigzag sequences of single-step upper bounds and single-step lower bounds. Let us draw single-step lower bounds by backward arrows and single-step upper bound by forward

arrows. Then, a lower-bound constraint pattern for the run segment $\sigma_{i..j}$ may be of the form:



Dually, an upper-bound constraint pattern for $\sigma_{i..j}$ may be of the form:



We hope that these pictures give enough intuition to justify the omission of the straightforward, but tedious, formal definition of general constraint patterns. For the relative completeness proof to go through, we have to establish the familiar two properties of constraint patterns for both the lower-bound case and the upper-bound case:

Property A Existence of a constraint pattern for $\sigma_{i..j}$ (use the S -validity of ϕ as before).

Property B Existence of at most finitely many different equivalence classes of constraint patterns. This is where the stability of ϕ comes into play: while the operability of S ensures the existence of a constraint pattern for $\sigma_{i..j}$ that does not cross beyond the position j , the stability of ϕ guarantees, by definition, that no constraint pattern for $\sigma_{i..j}$ crosses below the position i . Since S has only finitely many transitions, it follows that there can be only finitely many distinct constraint patterns within the finite interval from i to j .

Both properties ensure the existence of an S -valid untimed formula that characterizes all constraint patterns. It is not hard to see that the conditional crossover rules U-CMIX and \diamond -CMIX, which have been designed for exactly this purpose, suffice to collapse any reversal of direction in a zigzag constraint pattern. ■

Chapter 7

Discussion

The time has come to look back and see what has been accomplished, and what remains to be done. Once we agreed that real-time verification is a genuine object for theoretical investigation, we were immediately confronted with several decisions. To avoid distraction from the issues that concern time, we chose a simple model of computation and represented reactive systems by sets of infinite sequences of state changes. With regard to time, we faced two questions:

Semantics *How should we model time?* While physicists seem fairly unanimous in their opinion that, above quantum level, time is best approximated by the real line, philosophers and logicians have proposed a curious variety of different models of time. These models range from linear to branching to partial-order time and from discrete to dense to continuous time, to name just a few (consult, for example, [25, 124]). We believe that for computer scientists, the choice should be directed by two issues:

1. Given a certain mathematical model of physical time, *what can we prove in the model and how difficult is it to do so?* This question can, for any given model, be formulated and answered in precise complexity-theoretic terms. In the analog-clock model we encountered tremendous hurdles in the form of undecidability results. Hence we opted for the more abstract digital-clock model and an indirect approach to real-time verification.
2. If we have proved something in a certain mathematical model of physical time, *what, if anything, have we proved about physical time?* Needless to point out,

“verification” of a real-time system in, say, the digital-clock model is only sensible if the result gives us some insight into the physical reality. After all, that flight control system we just proved “correct” has to operate in dense time.

The question we have posed, however, must inevitably remain somewhat vague and philosophical (what is the “physical reality”?) and thus — unfortunately — is often neglected. We believe that the classical physicists’ model of time as the real line captures all aspects of reality that we are interested in, and we suggest that it serves as the point of reference for rating the adequacy of any real-time verification method that deems it convenient to assume, for computational or other reasons, a more abstract representation of time. As a case in point, we justified our use of the digital-clock model by showing that a large and important class of real-time systems and real-time properties is digitizable.

Syntax *How should we define real-time properties?* Given our model of time and computation, real-time properties are sets of timed state sequences. We chose to build on the established verification framework that has been developed for temporal logic. Thus we essentially looked at two ways to specify real-time properties — transition systems and temporal logics.

One commonly raised objection to temporal logic as a real-time specification language is that the very purpose of the temporal operators is the abstraction of time. Since, for the definition of timing properties, it is necessary to reintroduce time as a first-order domain, one should dispense with the temporal operators altogether. We showed that this argument leads to unnecessarily unwieldy and expensive specification languages. Instead, we pursued a more careful, and often more natural, introduction of time than by first-order time variables, which gave us the crucial benefit of elementary decision procedures.

To summarize, we have succeeded in incorporating time conservatively into transition systems and temporal logic. We have demonstrated that qualitative temporal reasoning about state sequences, be it model checking or theorem proving, can be naturally and conservatively extended to quantitative temporal reasoning about timed state sequences. Along the way, we identified the restrictions on syntax and semantics necessary for obtaining elementarily decidable instances of the real-time verification problem. We showed that only a very weak arithmetic over a discrete domain of time can be combined with reasoning

about state sequences to obtain decidable real-time logics. Then we presented two ways of constraining the syntax further to find elementary real-time extensions of linear temporal logic with the full expressive power of the maximal decidable theory of timed state sequences. Thus, the two temporal logics TPTL and MTL occupy a position among real-time logics that is as theoretically appealing as the standing of the untimed logic PTL is for qualitative reasoning.

In addition, we have provided evidence that timed transition systems naturally model many classes of real-time systems of practical importance, just as we have demonstrated that both TPTL and MTL are practical real-time specification languages. This is why we believe that the model-checking algorithms and the proof methodologies that we presented are important milestones on the long and winding road to the formal verification of "real" real-time systems.

7.1 Some Connections with Related Research

Even though the field is, by any standard, extremely young, there has been a surge of literature on the formal analysis of real-time systems in recent years. As the number of researchers has proliferated, so has the number of models and languages that have been studied. The proceedings of a recent workshop on the topic "Real Time: Theory in Practice" will provide an excellent starting point for anybody who wishes to explore the range of endeavors that are under way [31]. We have already pursued, throughout the thesis, concrete comparisons with formalisms that are directly related to the issues being discussed. So instead of trying (and necessarily failing) to give an exhaustive list of all proposals for formal reasoning about the combination of time and computation, we take this opportunity to attempt, first, a classification of the semantical assumptions of typical approaches. This will allow us to focus, thereafter, on the formalisms that are interpreted over models closely related to timed state sequences.

7.1.1 Semantic alternatives: Real-time models

In interpreting systems and specifications over timed state sequences, we have assumed a state-based, discrete, interleaved, linear, asynchronous model of computation:

State-based because states are our primitive components of system behaviors. Alternatives include *action-based* models, whose semantic primitives are state changes. It may be argued that state-based approaches are more general, because states can encode not only individual actions but entire histories of actions [78].

Although the choice of semantics is, in principle, often independent of the syntax of a language, there are certain classes of languages that traditionally have been interpreted over certain domains. While temporal logics are usually (but by no means necessarily) given a state-based semantics, process algebras generally refer to actions. Consequently, unlike temporal logic, process algebra cannot treat time explicitly as a state variable, and the need arose early on to extend process algebras with operators that refer to time. By now there are more proposals of how this may be done than we can enumerate and compare here; we only point to the real-time extensions of CSP by Reed and Roscoe [112], the real-time extensions of CCS by Moller and Tofts [101] and by Wang [128], the real-time extensions of ACP by Nicollin, Richier, Sifakis, and Voiron [103] and by Baeten and Bergstra [14], and the formalism *Communicating Shared Resources* of Gerber and Lee [44].

Discrete because we allow only countably many state changes. Alternatives include models for *continuous* processes and models for *hybrid* systems, which combine both discrete and continuous components [87]. A continuous process may change its state at every real point in time according to, say, a set of differential equations.

Interleaved because we model concurrent activity by nondeterministic interleaving. Alternatives include *partial-order* models and other “truly concurrent” models such as Petri nets (for issues about interleaving versus true concurrency, consult, for instance, the tutorials in [30]). For real-time extensions of Petri nets, we refer the reader to the proposals by Merlin and Farber [98] (see also [20]) and by Walter [127], and to the related work by Gabrielian and Franklin [41].

Linear because we identify systems that agree on the sets of their possible behaviors. Models that adhere to the combined assumption of interleaving and linearity are often called *trace* models, because they represent reactive systems as sets of traces. Alternatives include *branching* models with various stronger notions of system equivalence such as bisimulation (for the entire spectrum of possible equivalences, see, for

example, [125]). These models are traditionally studied in process algebra and, to some extent, in branching-time temporal logic.

Branching-time temporal logics clearly can be extended in the same ways in which we have augmented linear-time temporal logic, by freeze quantifiers or by time-bounded temporal operators. The latter approach has been pursued by Alur, Courcoubetis, and Dill [7], who obtained the surprising result that branching-time MTL, while still undecidable, permits model checking in the analog-clock model. The same conclusions apply to branching-time TPTL [6]. A similar result was independently shown by Lewis; he, too, gave a model-checking procedure for an MTL-like branching-time logic [82].

Asynchronous because we allow state changes at any real point in time. The combined assumption of discreteness and asynchronicity is sometimes referred to as *finite variability* [17], because only finitely many state changes can occur between any two points in time. Alternatives include *synchronous* models, in which all concurrent activity happens in lock-step; that is, the term “synchronicity” is used in the sense of Milner [100]: all component processes are clocked by a global clock.

The combined assumption of true concurrency and synchronicity has been called *maximal parallelism* by Pnueli and Harel [110] (although the term has also been used in connection with a weaker notion of synchronicity [73]). Pnueli and Harel contrast a maximally parallel semantics with our asynchronous interleaving semantics. Maximally parallel models identify “next-state” with “next-time” and, therefore, the *next* operator of traditional temporal logics can be used to reason about real-time properties. This approach has been taken by Pnueli and Harel [110] and by Gabrielian and Iyer [42] for linear-time logics, and by Emerson, Mok, Sistla, and Srinivasan [37] for a branching-time logic. Clearly, any such “calculus of the *next* operator” is strictly subsumed by our methods, because synchronous systems can be modeled by timed transition systems and time can be forced to act as a state counter in both TPTL and MTL. Moreover, we showed in Chapter 4 that the simplifying assumption of synchronicity does not make real-time verification any more tractable.

7.1.2 Syntactic alternatives: Real-time languages

Now let us point to some of the work that builds on the model of computation that we have used. There are two main categories of languages that have been proposed to define sets of timed state sequences — temporal-logic-based formalisms and automata-based formalisms. We include in this discussion all languages that can be interpreted over timed state sequences, even if their original semantics are somewhat more restrictive than ours, for example, “truly concurrent” by requiring that time increases *strictly* monotonically. Also, while almost every concrete proposal assumes a particular model of time — usually either a discrete or a continuous time domain — most of the languages in the following two groups can be interpreted in both the digital-clock model and the analog-clock model.

Temporal logics The overwhelming majority of extensions of temporal logic for real-time reasoning fall into one of two classes:

1. Logics with *time-bounded temporal operators* similar to MTL. This approach to the specification of timing properties has been advocated by Koymans, Vytöpil, and de Roever [71, 72, 74], although an early proposal by Bernstein and Harter can be viewed as a precursor [19]. Shasha, Pnueli, and Ewald [117] and Pnueli and Harel [110] have also used this method of expressing timing constraints. All of these proponents have been interested in deductive verification only and have employed ad hoc proof techniques that make use of a few valid formulas of MTL. While Koymans’ syntactic and semantic assumptions are far too permissive to obtain model-checking algorithms, Pnueli has applied the logic primarily in the overly restrictive synchronous case. Under the assumption of being given a complete axiomatization for MTL, Hooman and Widom presented a relatively complete proof system for verifying MTL-specifications of CSP-like programs [61].
2. Logics with an explicit *time variable*; that is, a state variable that refers, in any state, to the current time. In Section 3.1, we called a generic version of this type of language “real-time temporal logic.” Scattered examples of this method of expressing timing constraints have been used by Pnueli and de Roever [109] and by Ron [114]. More systematic expositions of the logic can be found in the work of Harel, Lichtenstein, and Pnueli [53, 54, 110] and of Ostroff, who

has presented a wide variety of interesting applications [104]. More recently, the use of a time variable has been advocated by Lamport and Abadi in the *Temporal Logic of Actions* [80]. All of these proponents have employed, for real-time verification, untimed safety proof methods in the spirit of Section 6.2. Ostroff has given model-checking procedures for a limited set of properties. Only Harel, Lichtenstein, and Pnueli have been interested in questions of decidability and, after we showed their logic to be undecidable, they identified a nonlogical decidable fragment that has a model-checking algorithm. We already critiqued the introduction of a time variable into temporal logic in Chapter 3 and found it to be both unnecessarily unwieldy for specification and prohibitively expensive for finite-state verification. It was these drawbacks that led us to design the logic TPTL.

The kind of assertional reasoning about real-time safety properties that refers, in state assertions, to an explicit time variable can also be carried out in non-temporal Hoare-style proof systems. This approach has been advocated for various kinds of programming languages; for instance, by Haase [47], Shankar and Lam [116], Schneider [115], and Hooman [60].

Real-time extensions of interval temporal logics (for example, [27, 97, 102]) do not fit properly into either of these two categories. Also the real-time logic RTL of Jahanian and Mok [63], essentially an extension of Presburger arithmetic with unary predicates, is quite different in flavor from conventional temporal logics. In Section 3.5 we showed it to be undecidable. Jahanian and Stuart have identified some decidable classes of RTL-formulas and presented, similar to Ostroff, specific model-checking procedures for individual classes of timing properties [65].

Automata As an alternative to logical languages, both specifications and implementations can be described in automata-based formalisms. We distinguish between operational and nonoperational approaches, in the sense that only the former restrict us to congruous definitions of real-time properties and, therefore, only they can serve directly as machine models or programming languages:

1. Timed transition systems fall, as we showed in Section 2.1, into the executable class. Similar state-transition systems with minimal delays and maximal delays on transitions have been defined by Pnueli and Harel [110] and by Ostroff [104].

Burch has used unit delays only [23]. Jahanian and Mok have modified *Statecharts* to *Modecharts* by incorporating minimal and maximal time delays [64]. All of these references use the state-transition systems as implementation languages only and specify real-time properties in one of the logical languages we discussed above.

A second group of researchers has used automata for both specification and implementation. Merritt, Modugno, and Tuttle have augmented I/O automata, which distinguish between input and output events, with minimal and maximal transition delays [99]. A variant of this model has been studied by Lynch and Attiya, who verify real-time properties by mappings between automata [85]. Aggarwal and Kurshan have extended Büchi automata in a similar way and use finite-state techniques for the verification of real-time properties [3].

2. Alur and Dill have added timing constraints to finite automata over infinite sequences in a much more flexible way than by putting only minimal and maximal delays on transitions [8, 9, 33, 34]. A comprehensive account of finite-state verification techniques that are based on this powerful notion of *timed automata* has been compiled by Alur [6]. Lewis has proposed a similar nonoperational extension of finite automata with time [82].

As far as we know, nobody has formally addressed the issues of real-time stuttering, safety, liveness, and operability, nor has anybody attempted to use the digital-clock model for the verification of continuous properties. Although we presented model-checking algorithms for the verification of timed transition systems only, it is not difficult to use our digital methods to check if, say, a timed automaton meets a temporal-logic specification.

We remark that while the extension of automata with minimal and maximal delays on transitions resembles the time-bounded temporal operators of MTL, the timed automata of Alur and Dill can be thought of as an augmentation of finite automata with the freeze quantifier of TPPL (although, in the automata context, time variables that are bound to the current time are, instead, called "clocks" or "timers" that are "set" to the current time). Thus there seem to be, independent of any particular language, two principal styles of adding timing constraints to untimed formalisms:

Local Lower and upper bounds on the time differences of *adjacent* states or actions or temporal contexts only. The time-bounded temporal operators of MTL and the minimal and maximal transition delays of timed transition systems fall into this category.

Global Lower and upper bounds on the time differences of *any* two states or actions or temporal contexts. In TPTL, for example, a freeze quantifier can be bound in one temporal context and checked for its value, later, in an arbitrarily distant temporal context (provided certain scoping rules are obeyed). Similarly, a timed automaton can set a clock with a transition and check its value, later, at an arbitrarily distant state. Also, while local timing constraints seem to dominate in process algebras and Petri nets, it is not hard to come up with such formalisms that permit global timing constraints.

We used the discreteness of time to show that MTL and TPTL are equally expressive in the digital-clock model. Yet it is doubtful that the global style of defining timing relations is in general no more expressive than the local style. In particular, we suspect that there are real-time properties in the analog-clock model that can be defined in TPTL but not in MTL. This remark anticipates our final comments, which briefly discuss some open problems in real-time verification.

7.2 Some Directions for Future Research

It is perhaps a sign of the vitality of a field that the answer to every question opens several new questions. Indeed, little has been solved in formal reasoning about real-time systems. Thus we find it appropriate to conclude by raising, in no particular order, a few problems that we would like to see addressed:

- *Making verification practical.* Clearly, the applicability of our verification techniques has not matured beyond the level of toy examples. We hope, however, that we have demonstrated the theoretical suitability of temporal logic for real-time verification, and that our languages will generate sufficient interest to warrant research on improving the practicality of our approach. The improvement of the finite-state verification methods is particularly pressing in the real-time case, which we showed to be exponentially more difficult than untimed verification. For this purpose, it will be essential to

1. Deal with the state explosion problem of model checking by, say, symbolic and partial techniques [24, 45].
2. Deal with timing information by symbolic constraints rather than constant delays.
3. Combine model checking and other decision procedures with theorem proving technology.
4. Decompose and compose systems, specifications, and correctness proofs [2, 15, 16].

Finite-state techniques will prove to be particularly useful if they can be practically applied not only to the verification of given systems but also to the automatic inference of time bounds and the automatic synthesis of system skeletons [34, 36, 94].

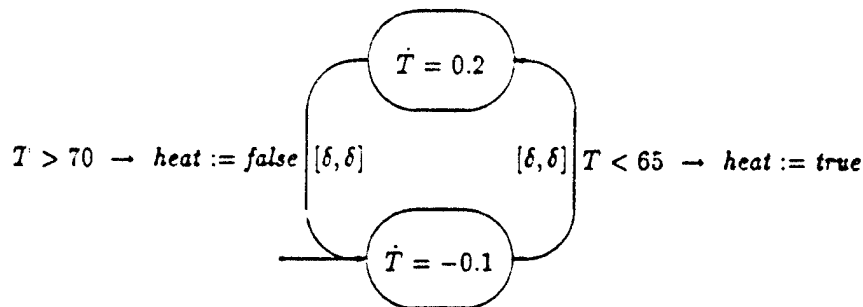
- *Increasing the scope of verification.* There are several problems about the deductive verification of real-time properties that have been left open in this thesis:
 1. Find a complete axiomatization for MTL (some axioms for a more general version of MTL than ours have been suggested by Koymans without claim of completeness [71]).
 2. Classify more complex real-time properties than bounded response and bounded invariance to obtain a hierarchy of real-time properties similar to the untimed hierarchy of temporal properties [93], and find relatively complete proof methods for all classes of properties in the real-time hierarchy.
 3. Determine which analog property is established if the digital-clock model is used to show that a timed transition system meets an arbitrary (i.e., not necessarily digitizable) specification that is given as a formula of TPTL or MTL.
- *The analog-clock model.* We feel strongly that in the analog-clock model, the interval semantics (see Section 3.4) is more appropriate than the timed state sequence semantics and recently we identified a real-time temporal logic that yields to finite-state verification techniques under an analog interval interpretation [9]. This result and the analog verification methods for timed automata [8] suggest that our indirect approach through the digital-clock model may be unnecessarily roundabout. However, from a theoretical perspective, the picture regarding the analog-clock model is far from complete.

We showed that the appealing untimed identification of *finite-state properties* with ω -regular languages leads also to a clean notion of "finite-state *real-time* property" in the digital-clock model: the class of *digital* finite-state real-time properties is exactly the class of properties that can be defined by sentences of the second-order theory of timed state sequences or, alternatively, by the temporal logic TETL or, alternatively, by timed automata or, alternatively, by several equivalent untimed formalisms with additional time-difference and time-congruence propositions. Moreover, this class is closed under all boolean operations and all problems of relevance to verification are elementarily decidable. It is, on the other hand, not obvious to us what constitutes "finite-state" information about a sequence of *real* numbers. Thus we have asked the question [12]:

Is there an agreeable notion of finite-state real-time property in the analog-clock model? The set of such properties ought to be closed under all boolean operations, have an elementarily decidable emptiness problem, and be, in a suitable sense, "maximal."

An acceptable characterization of analog finite-state properties would also lead to a theory of expressiveness of analog languages, including analog TPTL, analog MTL, and timed automata.

- *Local clocks and hybrid systems.* By adding time to transition systems, we introduced a single variable that changes continuously in the analog-clock model. This quality distinguishes time from all other system components. From here it is not hard to conceive of timed transition systems that permit all variables to change continuously as functions of time. We may, for example, add differential equations to all states of a timed transition system. The equations in a state govern the continuous change of all variables as long as the control of the system resides in the state. For instance, consider the following timed transition diagram, which models a thermostat:



The boolean variable *heat* indicates if the heater is turned on. The real variable T represents the room temperature and changes continuously: when the heater is turned on, it increases as a function of time; otherwise it decreases. The heater is turned off whenever the temperature rises above 70 degrees, but there is a delay of δ time units from the time that the temperature actually passes the threshold to the time that the heater is turned off; the heater is turned on again as soon as the temperature falls below 65 degrees after another sensory and mechanical delay of δ time units.

What we have just described is called a *hybrid* system, because it combines discrete and continuous elements. It can be given the semantics of Maler, Manna, and Pnueli, which is an extension of our interval semantics to several continuously changing variables [87]. As the results of Section 3.5 may prove to be insurmountable obstacles for the finite-state verification of hybrid systems, the interesting questions concern proof methods for hybrid real-time properties. We also remark that variables that change implicitly as functions of time, without the system taking any of its discrete transitions, can be used to model (analog or digital) *local clocks*, an important concept in many distributed algorithms.

- *Freeze quantification.* The freeze quantifier, which has turned out to be so useful in dealing with time, can, in principle, be added to any propositional modal logic, completely independent of the notion of time. Indeed, there seem to be some intriguing prospects for other applications of freeze quantification. Suppose, for example, that with every state of a program, we associate not a single-stamp but an entire vector containing the current values of all program variables, say, u_1, \dots, u_k . The freeze quantifier “ x .” binds this tuple to the variable x , and we have k functions, *value-of- u_i* , for $1 \leq i \leq k$ — one to access each component of x (i.e., the value of u_i). This allows us to assert program properties, such as the condition that the program variable u is

increased by 1 in the next execution step:

$$x. \bigcirc y. \text{value-of-}u(y) = \text{value-of-}u(x) + 1$$

or, in half-order *dynamic* logic:

$$x. \langle u := u + 1 \rangle y. \text{value-of-}u(y) = \text{value-of-}u(x) + 1$$

(for an introduction to dynamic logic, see, for example, [51]). This property is ordinarily stated in a much richer, first-order, modal logic:

$$\forall x. (u = x \rightarrow \langle u := u + 1 \rangle u = x + 1).$$

It can be argued that most transition axioms and input-output relations are more naturally written without universal quantifiers and auxiliary variables. In addition, similar to the real-time case there may be trade-offs between the expressiveness and complexity of programming logics with different forms of quantification.

Other applications of the freeze quantifier can be found in half-order logics of knowledge. For example, in the course of the knowledge-based analysis of a protocol, Halpern and Zuck introduced the notation " $p_{\mathbf{a}_i}$ " to denote the proposition p_i for the value i of the variable i in the current state [50]. This condition can be naturally expressed by freeze quantification. One may also attempt to interpret the freeze quantifier as a kind of personal pronoun that ranges over the domain of agents that are reasoning or the processors of a distributed system (for an overview of the use of epistemic logics in distributed computing, we refer to [48]).

Bibliography

- [1] ABADI, M., AND LAMPORT, L. The existence of refinement mappings. In *Proceedings of the Third Annual Symposium on Logic in Computer Science* (1988), IEEE Computer Society Press, pp. 165-175.
- [2] ABADI, M., AND LAMPORT, L. Composing specifications. In *Stepwise Refinement of Distributed Systems: Models, Formalism, Correctness*, J. de Bakker, W.-P. de Roever, and G. Rozenberg, Eds., Lecture Notes in Computer Science 430. Springer-Verlag, 1990.
- [3] AGGARWAL, S., AND KURSHAN, R. Modeling elapsed time in protocol specification. In *Proceedings of the Third IFIP WG6.1 International Workshop on Protocol Specification, Testing, and Verification* (1983), H. Rudin and C. West, Eds., Elsevier Science Publishers (North-Holland), pp. 51-62.
- [4] ALPERN, B., DEMERS, A., AND SCHNEIDER, F. Safety without stuttering. *Information Processing Letters* 23, 4 (1986), 177-180.
- [5] ALPERN, B., AND SCHNEIDER, F. Defining liveness. *Information Processing Letters* 21, 4 (1985), 181-185.
- [6] ALUR, R. *Techniques for Automatic Verification of Real-time Systems*. PhD thesis, Stanford University, 1991.
- [7] ALUR, R., COURCOUBETIS, C., AND DILL, D. Model checking for real-time systems. In *Proceedings of the Fifth Annual Symposium on Logic in Computer Science* (1990), IEEE Computer Society Press, pp. 414-425.

- [8] ALUR, R., AND DILL, D. Automata for modeling real-time systems. In *ICALP 90: Automata, Languages, and Programming*, M. Paterson, Ed., Lecture Notes in Computer Science 443. Springer-Verlag, 1990, pp. 322-335.
- [9] ALUR, R., FEDER, T., AND HENZINGER, T. The benefits of relaxing punctuality. In *Proceedings of the Tenth Annual Symposium on Principles of Distributed Computing* (1991), ACM Press, pp. 139-152.
- [10] ALUR, R., AND HENZINGER, T. A really temporal logic. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science* (1989), IEEE Computer Society Press, pp. 164-169.
- [11] ALUR, R., AND HENZINGER, T. Real-time logics: complexity and expressiveness. In *Proceedings of the Fifth Annual Symposium on Logic in Computer Science* (1990), IEEE Computer Society Press, pp. 390-401.
- [12] ALUR, R., AND HENZINGER, T. Time for logic. *SIGACT News* 22, 3 (1991).
- [13] APT, K., FRANCEZ, N., AND KATZ, S. Appraising fairness in languages for distributed programming. *Distributed Computing* 2, 4 (1988), 226-241.
- [14] BAETEN, J., AND BERGSTRA, J. Real-time process algebra. *Formal Aspects of Computing* 3, 2 (1991), 142-188.
- [15] BARRINGER, H. The use of temporal logic in the compositional specification of concurrent systems. In *Temporal Logics and their Applications*, A. Galton, Ed. Academic Press, 1987, pp. 53-90.
- [16] BARRINGER, H., KUIPER, R., AND PNUELI, A. Now you may compose temporal logic specifications. In *Proceedings of the 16th Annual Symposium on Theory of Computing* (1984), ACM Press, pp. 51-63.
- [17] BARRINGER, H., KUIPER, R., AND PNUELI, A. A really abstract concurrent model and its temporal logic. In *Proceedings of the 13th Annual Symposium on Principles of Programming Languages* (1986), ACM Press, pp. 173-183.
- [18] BEN-ARI, M., MANNA, Z., AND PNUELI, A. The temporal logic of branching time. In *Proceedings of the Eighth Annual Symposium on Principles of Programming Languages* (1981), ACM Press, pp. 164-176.

- [19] BERNSTEIN, A., AND HARTER, JR., P. Proving real-time properties of programs with temporal logic. In *Proceedings of the Eighth Annual Symposium on Operating System Principles* (1981), ACM Press, pp. 1-11.
- [20] BERTHOMIEU, B., AND DIAZ, M. Modeling and verification of time-dependent systems using time Petri nets. *IEEE Transactions on Software Engineering SE-17*, 3 (1991), 259-273.
- [21] BÜCHI, J. Weak second-order arithmetic and finite automata. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik* 6 (1960), 66-92.
- [22] BÜCHI, J. On a decision method in restricted second-order arithmetic. In *Proceedings of the First International Congress on Logic, Methodology, and Philosophy of Science 1960* (1962), E. Nagel, P. Suppes, and A. Tarski, Eds., Stanford University Press, pp. 1-11.
- [23] BURCH, J. Combining CTL, trace theory, and timing models. In *CAV 89: Automatic Verification Methods for Finite-state Systems*, J. Sifakis, Ed., Lecture Notes in Computer Science 407. Springer-Verlag, 1989, pp. 334-348.
- [24] BURCH, J., CLARKE, E., MCMILLAN, K., DILL, D., AND HWANG, L. Symbolic model checking: 10^{20} states and beyond. In *Proceedings of the Fifth Annual Symposium on Logic in Computer Science* (1990), IEEE Computer Society Press, pp. 428-439.
- [25] BURGESS, J. Basic tense logic. In *Handbook of Philosophical Logic*, D. Gabbay and F. Guenther, Eds., vol. II. D. Reidel Publishing Company, 1984, pp. 89-133.
- [26] CHANDY, K., AND MISRA, J. *Parallel Program Design: A Foundation*. Addison-Wesley Publishing Company, 1988.
- [27] CHAOCHEN, Z., HOARE, C., AND RAVN, A. A calculus of durations. *Information Processing Letters*. To appear.
- [28] CLARKE, E., EMERSON, E., AND SISTLA, A. Automatic verification of finite-state concurrent systems using temporal-logic specifications. *ACM Transactions on Programming Languages and Systems* 8, 2 (1986), 244-263.

- [29] CLARKE, E., AND GRÜMBERG, O. Research on automatic verification of finite-state concurrent systems. In *Annual Review of Computer Science*, J. Traub, B. Grosz, B. Lampson, and N. Nilsson, Eds., vol. II. Annual Reviews, 1987, pp. 269-290.
- [30] DE BAKKER, J., DE ROEVER, W.-P., AND ROZENBERG, G., Eds. *Linear Time, Branching Time, and Partial Order in Logics and Models for Concurrency*. Lecture Notes in Computer Science 354. Springer-Verlag, 1988.
- [31] DE ROEVER, W.-P., Ed. *Real Time: Theory in Practice*. Lecture Notes in Computer Science. Springer-Verlag. To appear.
- [32] DEDERICHES, F., AND WEBER, R. Safety and liveness from a methodological point of view. *Information Processing Letters* 36, 1 (1990), 25-30.
- [33] DILL, D. Timing assumptions and verification of finite-state concurrent systems. In *CAV 89: Automatic Verification Methods for Finite-state Systems*, J. Sifakis, Ed., Lecture Notes in Computer Science 407. Springer-Verlag, 1989, pp. 197-212.
- [34] DILL, D., AND WONG-TOI, H. Synthesizing processes and schedulers from temporal specifications. In *CAV 90: Automatic Verification Methods for Finite-state Systems*, R. Kurshan and E. Clarke, Eds., Lecture Notes in Computer Science. Springer-Verlag. To appear.
- [35] EMERSON, E. Temporal and modal logic. In *Handbook of Theoretical Computer Science*, J. van Leeuwen, Ed., vol. B. Elsevier Science Publishers (North-Holland), 1990, pp. 995-1072.
- [36] EMERSON, E., AND CLARKE, E. Using branching-time temporal logic to synthesize synchronization skeletons. *Science of Computer Programming* 2, 3 (1982), 241-266.
- [37] EMERSON, E., MOK, A., SISTLA, A., AND SRINIVASAN, J. Quantitative temporal reasoning. Presented at the First Annual Workshop on Computer-aided Verification, Grenoble, France, 1989.
- [38] ENDERTON, H. *A Mathematical Introduction to Logic*. Academic Press, 1972.
- [39] FITTING, M. *Proof Methods for Modal and Intuitionistic Logics*. D. Reidel Publishing Company, 1983.

- [40] GABBAY, D., PNUELI, A., SHELAH, S., AND STAVI, J. On the temporal analysis of fairness. In *Proceedings of the Seventh Annual Symposium on Principles of Programming Languages* (1980), ACM Press, pp. 163-173.
- [41] GABRIELIAN, A., AND FRANKLIN, M. Multi-level specification and verification of real-time software. *Communications of the ACM* 34, 5 (1991), 50-60.
- [42] GABRIELIAN, A., AND IYER, R. Integrating automata and temporal logic: a framework for specification of real-time systems and software. In *The Unified Computation Laboratory*, C. Rattray and R. Clark, Eds. Oxford University Press. To appear.
- [43] GARSON, J. Quantification in modal logic. In *Handbook of Philosophical Logic*, D. Gabbay and F. Guenther, Eds., vol. II. D. Reidel Publishing Company, 1984, pp. 249-307.
- [44] GERBER, R., AND LEE, I. Communicating Shared Resources: a model for distributed real-time systems. In *Proceedings of the Tenth Annual Real-time Systems Symposium* (1989), IEEE Computer Society Press, pp. 68-78.
- [45] GODEFROID, P., AND WOLPER, P. A partial approach to model checking. In *Proceedings of the Sixth Annual Symposium on Logic in Computer Science* (1991), IEEE Computer Society Press, pp. 406-415.
- [46] GOLDBLATT, R. *Logics of Time and Computation*. CSLI Lecture Notes 7. Center for the Study of Language and Information, Stanford, California, 1987.
- [47] HASE, V. Real-time behavior of programs. *IEEE Transactions on Software Engineering* SE-7, 5 (1981), 494-501.
- [48] HALPERN, J. Using reasoning about knowledge to analyze distributed systems. In *Annual Review of Computer Science*, J. Traub, B. Grosz, B. Lampson, and N. Nilsson, Eds., vol. II. Annual Reviews, 1987, pp. 37-68.
- [49] HALPERN, J. Presburger arithmetic with unary predicates is Π_1^1 -complete. *The Journal of Symbolic Logic* 56, 2 (1991), 637-642.
- [50] HALPERN, J., AND ZUCK, L. A little knowledge goes a long way: simple knowledge-based derivations and correctness proofs for a family of protocols. In *Proceedings of*

the Sixth Annual Symposium on Principles of Distributed Computing (1987), ACM Press, pp. 269-280.

- [51] HAREL, D. Dynamic logic. In *Handbook of Philosophical Logic*, D. Gabbay and F. Guenther, Eds., vol. II. D. Reidel Publishing Company, 1984, pp. 497-604.
- [52] HAREL, D., PNUELI, A., AND STAVI, J. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences* 26, 2 (1983), 222-243.
- [53] HAREL, E. Temporal analysis of real-time systems. Master's thesis, The Weizmann Institute of Science, Rehovot, Israel, 1988.
- [54] HAREL, E., LICHTENSTEIN, O., AND PNUELI, A. Explicit-clock temporal logic. In *Proceedings of the Fifth Annual Symposium on Logic in Computer Science* (1990), IEEE Computer Society Press, pp. 402-413.
- [55] HENKIN, L. The completeness of the first-order functional calculus. *The Journal of Symbolic Logic* 14, 3 (1949), 159-166.
- [56] HENZINGER, T. Half-order modal logic: how to prove real-time properties. In *Proceedings of the Ninth Annual Symposium on Principles of Distributed Computing* (1990), ACM Press, pp. 281-296.
- [57] HENZINGER, T., MANNA, Z., AND PNUELI, A. An interleaving model for real time. In *Proceedings of the Fifth Jerusalem Conference on Information Technology* (1990), IEEE Computer Society Press, pp. 717-730.
- [58] HENZINGER, T., MANNA, Z., AND PNUELI, A. Temporal proof methodologies for real-time systems. In *Proceedings of the 18th Annual Symposium on Principles of Programming Languages* (1991), ACM Press, pp. 353-366.
- [59] HOARE, C. *Communicating Sequential Processes*. Prentice-Hall, 1985.
- [60] HOOMAN, J. *Specification and Compositional Verification of Real-time Systems*. PhD thesis, Technische Universiteit Eindhoven, The Netherlands, 1991.
- [61] HOOMAN, J., AND WIDOM, J. A temporal-logic-based compositional proof system for real-time message passing. In *PARLE 89: Parallel Architectures and Languages*

- Europe, E. Odijk, M. Rem, and J.-C. Syre, Eds., vol. II, Lecture Notes in Computer Science 366. Springer-Verlag, 1989, pp. 424-441.
- [62] HOPCROFT, J., AND ULLMAN, J. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Publishing Company, 1979.
- [63] JAHANIAN, F., AND MOK, A. Safety analysis of timing properties in real-time systems. *IEEE Transactions on Software Engineering SE-12*, 9 (1986), 890-904.
- [64] JAHANIAN, F., AND MOK, A. A graph-theoretic approach for timing analysis and its implementation. *IEEE Transactions on Computers C-36*, 8 (1987), 961-975.
- [65] JAHANIAN, F., AND STUART, D. A method for verifying properties of Modechart specifications. In *Proceedings of the Ninth Annual Real-time Systems Symposium* (1988), IEEE Computer Society Press, pp. 12-21.
- [66] JAY, F., Ed. *IEEE Standard Dictionary of Electrical and Electronics Terms*, fourth ed. The Institute of Electrical and Electronics Engineers, Piscataway, New Jersey, 1988.
- [67] JAYASIMHA, D. *Communication and Synchronization in Parallel Computation*. PhD thesis, University of Illinois at Urbana-Champaign, 1988.
- [68] KAMP, J. *Tense Logic and the Theory of Linear Order*. PhD thesis, University of California at Los Angeles, 1968.
- [69] KELLER, R. Formal verification of parallel programs. *Communications of the ACM* 19, 7 (1976), 371-384.
- [70] KELLEY, J. *General Topology*. Springer-Verlag, 1955.
- [71] KOYMANS, R. Specifying real-time properties with metric temporal logic. *Real-time Systems* 2, 4 (1990), 255-299.
- [72] KOYMANS, R., AND DE ROEVER, W.-P. Examples of a real-time temporal specification. In *The Analysis of Concurrent Systems*, B. Denfir, W. Harwood, M. Jackson, and M. Wray, Eds., Lecture Notes in Computer Science 207. Springer-Verlag, 1985, pp. 231-252.

- [73] KOYMANS, R., SHYAMASUNDAR, R., DE ROEVER, W.-P., GERTH, R., AND ARUNKUMAR, S. Compositional semantics for real-time distributed computing. *Information and Computation* 79, 3 (1988), 210-256.
- [74] KOYMANS, R., VYTOPIL, J., AND DE ROEVER, W.-P. Real-time programming and asynchronous message passing. In *Proceedings of the Second Annual Symposium on Principles of Distributed Computing* (1983), ACM Press, pp. 187-197.
- [75] KOZEN, D. Results on the propositional μ -calculus. *Theoretical Computer Science* 27, 3 (1983), 333-354.
- [76] LAMPORT, L. Proving the correctness of multiprocess programs. *IEEE Transactions on Software Engineering SE-3*, 2 (1977), 125-143.
- [77] LAMPORT, L. Specifying concurrent program modules. *ACM Transactions on Programming Languages and Systems* 5, 2 (1983), 190-222.
- [78] LAMPORT, L. What good is temporal logic? In *Information Processing 83: Proceedings of the Ninth IFIP World Computer Congress* (1983), R. Mason, Ed., Elsevier Science Publishers (North-Holland), pp. 657-668.
- [79] LAMPORT, L. The temporal logic of actions. Tech. rep., DEC Systems Research Center, Palo Alto, California, 1991.
- [80] LAMPORT, L., AND ABADI, M. Refining and composing real-time specifications. In *Real Time: Theory in Practice*, W.-P. de Boever, Ed., Lecture Notes in Computer Science. Springer-Verlag. To appear.
- [81] LEMMON, E. Algebraic semantics for modal logics. *The Journal of Symbolic Logic* 31, 2 (1966), 191-218.
- [82] LEWIS, H. A logic of concrete time intervals. In *Proceedings of the Fifth Annual Symposium on Logic in Computer Science* (1990), IEEE Computer Society Press, pp. 380-389.
- [83] LICHTENSTEIN, O., AND PNUELI, A. Checking that finite-state concurrent programs satisfy their linear specification. In *Proceedings of the 11th Annual Symposium on Principles of Programming Languages* (1984), ACM Press, pp. 97-107.

- [84] LICHTENSTEIN, O., PNUELI, A., AND ZUCK, L. The glory of the past. In *Logics of Programs*, R. Parikh, Ed., Lecture Notes in Computer Science 193. Springer-Verlag, 1985, pp. 196-218.
- [85] LYNCH, N., AND ATTIYA, H. Using mappings to prove timing properties. In *Proceedings of the Ninth Annual Symposium on Principles of Distributed Computing* (1990), ACM Press, pp. 265-280.
- [86] LYNCH, N., AND TUTTLE, M. Hierarchical correctness proofs for distributed algorithms. In *Proceedings of the Sixth Annual Symposium on Principles of Distributed Computing* (1987), ACM Press, pp. 137-151.
- [87] MALER, O., MANNA, Z., AND PNUELI, A. A formal approach to hybrid systems. Unpublished manuscript, 1991.
- [88] MANNA, Z., AND PNUELI, A. How to cook a temporal proof system for your pet language. In *Proceedings of the Tenth Annual Symposium on Principles of Programming Languages* (1983), ACM Press, pp. 141-154.
- [89] MANNA, Z., AND PNUELI, A. Proving precedence properties: the temporal way. In *ICALP 83: Automata, Languages, and Programming*, J. Diaz, Ed., Lecture Notes in Computer Science 154. Springer-Verlag, 1983, pp. 491-512.
- [90] MANNA, Z., AND PNUELI, A. Adequate proof principles for invariance and liveness properties of concurrent programs. *Science of Computer Programming* 4, 3 (1984), 257-289.
- [91] MANNA, Z., AND PNUELI, A. The anchored version of the temporal framework. In *Linear Time, Branching Time, and Partial Order in Logics and Models for Concurrency*, J. de Bakker, W.-P. de Roever, and G. Rozenberg, Eds., Lecture Notes in Computer Science 354. Springer-Verlag, 1989, pp. 201-284.
- [92] MANNA, Z., AND PNUELI, A. Completing the temporal picture. In *ICALP 89: Automata, Languages, and Programming*, G. Ausiello, M. Dezani-Ciancaglini, and S. Ronchi Della Rocca, Eds., Lecture Notes in Computer Science 372. Springer-Verlag, 1989, pp. 534-558.

- [93] MANNA, Z., AND PNUELI, A. A hierarchy of temporal properties. In *Proceedings of the Ninth Annual Symposium on Principles of Distributed Computing* (1990), ACM Press, pp. 377-408.
- [94] MANNA, Z., AND WOLPER, P. Synthesis of communicating processes from temporal-logic specifications. *ACM Transactions on Programming Languages and Systems* 6, 1 (1984), 68-93.
- [95] MCNAUGHTON, R. Testing and generating infinite sequences by a finite automaton. *Information and Control* 9, 5 (1966), 521-530.
- [96] MCNAUGHTON, R., AND PAPERT, S. *Counter-free Automata*. The MIT Press, 1971.
- [97] MELLIAR-SMITH, P. Extending interval logic to real-time systems. In *Temporal Logic in Specification*, B. Banieqbal, H. Barringer, and A. Pnueli, Eds., Lecture Notes in Computer Science 398. Springer-Verlag, 1989, pp. 224-242.
- [98] MERLIN, P., AND FARBER, D. Recoverability of communication protocols: implications of a theoretical study. *IEEE Transactions on Communications COM-24*, 9 (1976), 1036-1043.
- [99] MERRITT, M., MODUGNO, F., AND TUTTLE, M. Time-constrained automata. In *CONCUR 91: Theories of Concurrency*, Lecture Notes in Computer Science. Springer-Verlag. To appear.
- [100] MILNER, R. Calculi for synchrony and asynchrony. *Theoretical Computer Science* 25, 3 (1983), 267-310.
- [101] MOLLER, F., AND TOFTS, C. A temporal calculus of communicating processes. In *CONCUR 90: Theories of Concurrency: Unification and Extension*, J. Baeten and J. Klop, Eds., Lecture Notes in Computer Science 458. Springer-Verlag, 1990, pp. 401-415.
- [102] NARAYANA, K., AND AABY, A. Specification of real-time systems in real-time temporal interval logic. In *Proceedings of the Ninth Annual Real-time Systems Symposium* (1988), IEEE Computer Society Press, pp. 86-95.
- [103] NICOLLIN, X., RICHIER, J.-L., SIFAKIS, J., AND VOIRON, J. ATP: an algebra for timed processes. In *Proceedings of the IFIP WG2.2/2.3 Working Conference on*

- Programming Concepts and Methods* (1990), M. Broy and C. Jones, Eds., Elsevier Science Publishers (North-Holland), pp. 415-442.
- [104] OSTROFF, J. *Temporal Logic of Real-time Systems*. Research Studies Press, 1990.
- [105] OWICKI, S., AND LAMPORT, L. Proving liveness properties of concurrent programs. *ACM Transactions on Programming Languages and Systems* 4, 3 (1982), 455-495.
- [106] PNUELI, A. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science* (1977), IEEE Computer Society Press, pp. 46-57.
- [107] PNUELI, A. The temporal semantics of concurrent programs. *Theoretical Computer Science* 13, 1 (1981), 45-60.
- [108] PNUELI, A. Applications of temporal logic to the specification and verification of reactive systems: a survey of current trends. In *Current Trends in Concurrency*, J. de Bakker, W.-P. de Roever, and G. Rozenberg, Eds., Lecture Notes in Computer Science 224. Springer-Verlag, 1986, pp. 510-584.
- [109] PNUELI, A., AND DE ROEVER, W.-P. Rendez-vous with Ada: a proof-theoretical view. In *Proceedings of the SIGPLAN AdaTEC Conference on Ada* (1982), ACM Press, pp. 129-137.
- [110] PNUELI, A., AND HAREL, E. Applications of temporal logic to the specification of real-time systems. In *Formal Techniques in Real-time and Fault-tolerant Systems*, M. Joseph, Ed., Lecture Notes in Computer Science 331. Springer-Verlag, 1988, pp. 84-98.
- [111] PRATT, V. A near-optimal method for reasoning about action. *Journal of Computer and System Sciences* 20, 2 (1980), 231-254.
- [112] REED, G., AND ROSCOE, A. A timed model for communicating sequential processes. *Theoretical Computer Science* 58, 1/2/3 (1988), 249-261.
- [113] ROGERS, JR., H. *Theory of Recursive Functions and Effective Computability*. McGraw-Hill Book Company, 1967.

- [114] RON, D. Temporal verification of communication protocols. Master's thesis, The Weizmann Institute of Science, Rehovot, Israel, 1984.
- [115] SCHNEIDER, F. Real-time, reliable systems project. In *Proceedings of the ONR Kickoff Workshop for the Foundations of Real-time Computing Research Initiative* (1988), Office of Naval Research, pp. 28-32.
- [116] SHANKAR, A., AND LAM, S. Time-dependent distributed systems: proving safety, liveness, and timing properties. *Distributed Computing* 2, 2 (1987), 61-79.
- [117] SHASHA, D., PNUELI, A., AND EWALD, W. Temporal verification of carrier-sense local area network protocols. In *Proceedings of the 11th Annual Symposium on Principles of Programming Languages* (1984), ACM Press, pp. 54-65.
- [118] SISTLA, A. *Theoretical Issues in the Design and Verification of Distributed Systems*. PhD thesis, Harvard University, 1983.
- [119] SISTLA, A., AND CLARKE, E. The complexity of propositional linear temporal logics. *Journal of the ACM* 32, 3 (1985), 733-749.
- [120] SMULLYAN, R. *First-order Logic*. Springer-Verlag, 1968.
- [121] STOCKMEYER, L. *The Complexity of Decision Problems in Automata Theory and Logic*. PhD thesis, Massachusetts Institute of Technology, 1974.
- [122] THOMAS, W. A combinatorial approach to the theory of ω -automata. *Information and Control* 48, 3 (1981), 261-283.
- [123] THOMAS, W. Automata on infinite objects. In *Handbook of Theoretical Computer Science*, J. van Leeuwen, Ed., vol. B. Elsevier Science Publishers (North-Holland), 1990, pp. 133-191.
- [124] VAN BENTHEM, J. *The Logic of Time*. D. Reidel Publishing Company, 1983.
- [125] VAN GLABBEK, R. *Comparative Concurrency Semantics and Refinement of Actions*. PhD thesis, Vrije Universiteit te Amsterdam, The Netherlands, 1990.
- [126] VARDI, M. A temporal fixpoint calculus. In *Proceedings of the 15th Annual Symposium on Principles of Programming Languages* (1988), ACM Press, pp. 250-259.

- [127] WALTER, B. Timed Petri nets for modeling and analyzing protocols with real-time characteristics. In *Proceedings of the Third IFIP WG6.1 International Workshop on Protocol Specification, Testing, and Verification* (1983), H. Rudin and C. West, Eds., Elsevier Science Publishers (North-Holland), pp. 149-159.
- [128] WANG, Y. Real-time behavior of asynchronous agents. In *CONCUR 90: Theories of Concurrency: Unification and Extension*, J. Baeten and J. Klop, Eds., Lecture Notes in Computer Science 458. Springer-Verlag, 1990, pp. 502-520.
- [129] WOLPER, P. *Synthesis of Communicating Processes from Temporal-Logic Specifications*. PhD thesis, Stanford University, 1982.
- [130] WOLPER, P. Temporal logic can be more expressive. *Information and Control* 56, 1/2 (1983), 72-99.
- [131] WOLPER, P., VARDI, M., AND SISTLA, A. Reasoning about infinite computation paths. In *Proceedings of the 24th Annual Symposium on Foundations of Computer Science* (1983), IEEE Computer Society Press, pp. 185-194.

END
FILMED

DATE: 4-9-96

NTIS