Discrete Analysis of Continuous Behaviour in Real-Time Concurrent Systems

Joël Ouaknine
St Cross College and University College

Thesis submitted for the degree of Doctor of Philosophy
at the University of Oxford, Michaelmas 2000

Discrete Analysis of Continuous Behaviour in Real-Time Concurrent Systems

Joël Ouaknine

Thesis submitted for the degree of Doctor of Philosophy
at the University of Oxford, Michaelmas 2000

Abstract

This thesis concerns the relationship between *continuous* and *discrete* modelling paradigms for timed concurrent systems, and the exploitation of this relationship towards applications, in particular model checking. The framework we have chosen is Reed and Roscoe's process algebra Timed CSP, in which semantic issues can be examined from both a denotational and an operational perspective. The continuous-time model we use is the timed failures model; on the discrete-time side, we build a suitable model in a CSP-like setting by incorporating a distinguished *tock* event to model the passage of time. We study the connections between these two models and show that our framework can be used to verify certain specifications on continuous-time processes, by building upon and extending results of Henzinger, Manna, and Pnueli's. Moreover, this verification can in many cases be carried out directly on the model checker FDR[1]. Results are illustrated with a small railway level crossing case study. We also construct a second, more sophisticated discrete-time model which reflects continuous behaviour in a manner more consistent with one's intuition, and show that our results carry over this second model as well.

[1]FDR is a commercial product of Formal Systems (Europe) Ltd.

Acknowledgements

I would like to thank my supervisor, Mike Reed, for his encouragement, advice, and guidance. I am also grateful for his kindness and generosity, as well as for sharpening my understanding and appreciation of life in academia. I am indebted to Bill Roscoe, whose impeccable work in CSP (building on Hoare's seminal legacy) and Timed CSP (joint with Mike Reed) has been a rich and constant source of inspiration. Steve Schneider, Michael Goldsmith, James Worrell, and Mike Mislove deserve my thanks for many stimulating discussions.

This thesis is dedicated to my parents, for their love and support. Grateful thanks are also due to the rest of my family as well as to my friends, in Oxford and elsewhere, for making my life so full and exhilarating.

Finally, I wish to thank the Fonds FCAR (Québec) and the US Office for Naval Research for financial support during the course of this thesis.

There are no things,
only processes.

David Bohm

Contents

Chapter 1

Introduction

The motivations for undertaking the work presented in this thesis originate from two distinct sources. The more abstract was a desire to embark on a project with some relevance, however vague, to the ‘real world’. *Real-time concurrent systems*, composed of several components interacting with each other subject to timing constraints, certainly seemed a good candidate to fulfill this ambition: after all, such systems appear in an increasingly large number of applications, from kitchen appliances to nuclear power, telecommunications, aeronautics, and so on. Moreover, in many instances it is in fact crucial that these systems behave exactly as they were intended to, lest catastrophic consequences ensue. Unfortunately, the complexities involved often mean that it is very difficult—if not impossible—to satisfy oneself that a system will indeed always behave as intended.

The field of *formal methods*, which seeks to apply rigorous mathematical techniques to the understanding and analysis of computerised systems, was therefore an exciting area in which to undertake research. A prominent modelling paradigm within formal methods is that of *process algebra*, which in the case of timed systems splits into two branches, according to whether time is modelled in a *discrete* or *continuous/dense* fashion.

Although much work has been carried out in both the discrete and continuous instances, much less is known about the relationship between them. This fact provided a second incentive for the author to immerse himself into the subject.

The question remained of which framework to choose. Reed and Roscoe’s timed failures model for Timed CSP [RR86, Ree88, RR99, Sch00] seemed an excellent candidate to act as the continuous-time process algebra representative. Not only was it already quite well understood, having been the focus of a considerable amount of prior work, but it also encompassed most of the salient features found in other timed process algebras, and often more; it boasts, for instance, congruent denotational and operational models, whereas many process algebras are predicated solely upon an operational semantics.

An additional advantage of Timed CSP was that it is a natural extension of Hoare's CSP [Hoa85, Ros97], which Roscoe had used to describe discrete-time systems with the help of a CSP-coded fictitious clock. The project quickly evolved into one aimed at elucidating the relationship between these two distinct methods for modelling time.

One of the foremost applications of this research lies in *model checking*. Model checking consists in a (usually automated) graph-theoretic analysis of the state space of a system, with the goal of establishing whether or not a given specification on the system actually holds. Its main overall aim is to ensure reliability and correctness, properties which we argued earlier can be of paramount importance.

Model checking has a rich history, with one of the first reported instances of it dating back almost twenty years [QS81]. It has achieved a number of spectacular successes, yet formidable obstacles still remain in its path. The situation is yet worse when time is considered, as the following example demonstrates. Consider a trivial process such as $a \longrightarrow STOP$: it can communicate the event a at any time, and then deadlock. Under a continuous-time interpretation, this process has an uncountable number of behaviours, and hence an uncountably large state space. Mechanical exploration of the latter will therefore not be possible until some drastic reductions are effected. These sorts of difficulties explain why the first model checking algorithm for continuous-time systems only arose approximately a decade ago, fruit of the pioneering work of Alur, Courcoubetis, and Dill [ACD90, ACD93].

Discrete-time modelling, being a much more straightforward extension of untimed modelling, poses considerably fewer problems: model checking techniques developed for the untimed world are reasonably easy to extend to discrete-time frameworks. In particular, our proposed enterprise potentially entailed that one could employ the CSP model checker FDR to verify specifications on continuous-time systems described in Timed CSP. Although Jackson [Jac92] gave an algorithm, based on that of Alur *et al.*'s, to model check continuous-time processes written in a significantly restricted subset of Timed CSP, no actual implementation of his or any other approach exists to this day.[1]

The object of this thesis is therefore twofold: the first goal is to study the relationship between continuous-time and discrete-time models for timed process algebras, focussing on (Timed and untimed) CSP-based formalisms; the second goal is to apply the results gathered thus towards applications, in particular model checking.

The thesis is structured as follows:

In Chapter 2 we introduce the basic notation and concepts of Timed CSP and semantic modelling, and list a number of definitions and conventions which apply throughout the thesis.

Chapter 3 is devoted to the timed failures model and its congruent operational counterpart. We establish a crucial result, the *digitisation lemma*, which says that the continuous-time operational behaviour of any Timed CSP process is so-called *closed under digitisation*; this fact is the main ingredient allowing us later on to relate the discrete-time and continuous-time denotational behaviours of processes to each other. We conclude Chapter 3 by discussing process specification and verification.

Chapter 4 presents and considers the main issues which arise when one attempts to emulate as faithfully as possible the timed failures model in a discrete CSP-based setting.

Building on these observations, we then carefully construct a suitable discrete-time denotational model in Chapter 5. A congruent operational semantics is also given, and specifications as well as verification and the applicability of the model checker FDR are discussed.

Chapter 6 addresses the central question of the relationship between our discrete-time model and the continuous-time model of Chapter 3, and the impact of this relationship on verification. An intuitive and straightforward approach is first presented, providing insight and interesting results, but found to be wanting in certain respects. We then offer a more sophisticated (if more specialised) verification method which builds upon, and subsequently extends, a result of Henzinger, Manna, and Pnueli's. This probably constitutes the most important and significant application of our work to continuous-time system verification. A small railway level crossing case study is presented to illustrate our findings.

[1]However, a small number of continuous-time model checking tools, such as COSPAN [AK96], UPPAAL [BLL+96], KRONOS [DOTY96], and HYTECH [HHWT97] have since been developed in settings other than Timed CSP.

In Chapter 7, we build a second discrete-time model which circumvents some shortcomings observed in the first model with respect to our 'straightforward' approach to verification. The treatment is similar to that of Chapter 5, if brisker.

Chapter 8, likewise, offers a streamlined version of Chapter 6 in which we discuss the connections between this new model and the timed failures model, as well the upshots in terms of verification. We derive a crisper qualitative understanding of the scope and limitations of our discrete-time approach to modelling continuous-time systems, as well as of the costs associated with more faithful discrete-time modelling.

Lastly, we sum-up the entire enterprise in Chapter 9, compare our results with related work appearing in the literature, and propose a number of avenues for further research.

The reader will also find three appendices, regrouping technical proofs, a presentation of a model checking algorithm for our discrete-time models, and a congruent operational semantics for the more sophisticated timed failures-stability model for Timed CSP. The latter is referred to in the 'further work' section of Chapter 9.

This thesis is essentially self-contained. However, some basic familiarity with CSP would be useful, particularly in Chapter 4.

Chapter 2

Notation and Basic Concepts

We begin by laying out the syntax of Timed CSP and stating a few conventions. We then continue with some remarks on semantic modelling, and conclude by introducing some standard notation about sequences.

2.1 Timed CSP

We assume that we are given a finite[1] set of events Σ, with $\mathit{tock} \notin \Sigma$ and $\checkmark \notin \Sigma$. We write $\Sigma^{\checkmark}$ to denote $\Sigma \cup \{\checkmark\}$, Σ_{tock} to denote $\Sigma \cup \{\mathit{tock}\}$, and $\Sigma^{\checkmark}_{\mathit{tock}}$ to denote $\Sigma \cup \{\checkmark, \mathit{tock}\}$. In the notation below, we have $a \in \Sigma$, $A \subseteq \Sigma$, and $B \subseteq \Sigma^{\checkmark}$. The parameter n ranges over the set of non-negative integers $\mathbb{N}$. f represents a function $f : \Sigma \longrightarrow \Sigma$; it can also be viewed as a function $f : \Sigma^{\checkmark}_{\mathit{tock}} \longrightarrow \Sigma^{\checkmark}_{\mathit{tock}}$, lifted in the obvious way. The variable X is drawn from a fixed infinite set of process variables $\mathit{VAR} \mathrel{\widehat{=}} \{X, Y, Z, \ldots\}$.

Timed CSP terms are constructed according to the following grammar:

$$\begin{array}{rcl} P & := & \mathit{STOP} \mid \mathit{SKIP} \mid \mathit{WAIT}\ n \mid P_1 \overset{n}{\rhd} P_2 \mid \\ & & a \longrightarrow P \mid a : A \longrightarrow P(a) \mid P_1 \mathbin{\Box} P_2 \mid P_1 \sqcap P_2 \mid \\ & & P_1 \underset{B}{\parallel} P_2 \mid P_1 \interleave P_2 \mid P_1 \mathbin{;} P_2 \mid P \setminus A \mid \\ & & f^{-1}(P) \mid f(P) \mid X \mid \mu X \bullet P \text{ [if } P \text{ is time-guarded for } X]. \end{array}$$

These terms have the following intuitive interpretations:

[1] Our restriction that Σ be finite is perhaps not absolutely necessary, but is certainly sensible from a practical (i.e., automated verification) point of view.

- $STOP$ is the deadlocked, stable process which is only capable of letting time pass.

- $SKIP$ intuitively corresponds to the process $\checkmark \longrightarrow STOP$, i.e., a process which at any time is willing to terminate successfully (the latter being represented by communication of the event $\checkmark$), and then do nothing.

- $WAIT\ n$ is the process which will idle for n time units, and then become $SKIP$ (and offer to terminate).

- $a \longrightarrow P$ initially offers at any time to engage in the event a, and subsequently behaves like P; note that when the process P is thus 'activated', it considers that time has just started, even if the occurrence of a took place some positive amount of time after one first started to observe $a \longrightarrow P$—in other words, P is not 'turned on' until a is witnessed. The general prefixed process $a : A \longrightarrow P(a)$ is initially prepared to engage in any of the events $a \in A$, at the choice of the environment, and thereafter behave like $P(a)$; this corresponds to $STOP$ when $A = \emptyset$.

- $P \overset{n}{\triangleright} Q$ is the process that initially offers to become P for n time units, after which it silently becomes Q if no visible event (necessarily from P) has occurred. P is initially turned on, and Q gets turned on after n time units if P has failed to communicate any event in the meantime.

- $P \sqcap Q$ represents the nondeterministic (or internal) choice between P and Q. Which of these two processes $P \sqcap Q$ chooses to become is independent of the environment, and how this choice is resolved is considered to be outside the domain of discourse. This choice is effected without delay.

- $P \mathrel{\Box} Q$, on the other hand, denotes a process which is willing to behave either like P or like Q, at the choice of the environment. This decision is taken on the first visible event (and not before), and is nondeterministic only if this initial event is possible for both P and Q. Both P and Q are turned on as soon as $P \mathrel{\Box} Q$ is.

- The parallel composition $P_1 \underset{B}{\|} P_2$ of P_1 and P_2, over the interface set B, forces P_1 and P_2 to agree and synchronise on all events in B, and to behave independently of each other with respect to all other events. When $\checkmark \notin B$, $P_1 \underset{B}{\|} P_2$ terminates (and makes no further communications) as soon as either of its subcomponents does. P_1 and P_2 are turned on throughout. $P_1 \,|||\, P_2$ corresponds to parallel composition over an empty interface—each process behaves independently of the other, except again for termination, which halts any further progress altogether. Note however that it is assumed that time flows at a universal and constant rate, the same for all processes that are turned on.

- $P \,;\, Q$ corresponds to the sequential composition of P and Q: it denotes a process which behaves like P until P chooses to terminate (silently), at which point the process seamlessly starts to behave like Q. The process Q is turned on precisely when P terminates.

- $P \setminus A$ is a process which behaves like P but with all communications in the set A hidden (made invisible to the environment); the assumption of *maximal progress*, or *τ-urgency*, dictates that no time can elapse while hidden events are on offer—in other words, hidden events happen as soon as they become available.

- The renamed processes $f^{-1}(P)$ and $f(P)$ derive their behaviours from those of P in that, whenever P can perform an event a, $f^{-1}(P)$ can engage in any event b such that $f(b) = a$, whereas $f(P)$ can perform $f(a)$.

- The process variable X has no intrinsic behaviour of its own, but can imitate any process P under certain conditions—it is however best interpreted as a formal variable for the time being.

- Lastly, the recursion $\mu X \centerdot P$ represents the unique solution to the equation $X = P$ (where the variable X usually appears freely within P's body). The operator μX binds every free occurrence of X in P. The condition ("if P is time-guarded for X") ensures that the recursion is well-defined and has a unique solution; the formal definition of time-guardedness follows shortly.

Following [DS95], we will normally refer to the closed terms[2] of the free syntactic algebra thus generated as *programs*, rather than *processes*, reserving the latter terminology for the elements of the denotational models we will be considering. The two concepts are very closely related however, and since the distinction between them is often not explicitly made in the literature, we will on occasion abuse this convention ourselves and refer to both as processes (as we have done above).

[2]A *closed term* is a term with no free variable: every process variable X in it is within the scope of a μX operator.

We will occasionally use the following derived constructs: abbreviating $a \longrightarrow STOP$ as simply $\bar{a}$, and writing $a \overset{n}{\longrightarrow} P$ instead of $a \longrightarrow WAIT\ n\ ; P$, and similarly for the general prefix operator. $P \parallel Q$ stands for $P \underset{\Sigma^{\checkmark}}{\parallel} Q$. In the case of hiding a single event a, we write $P \setminus a$ rather than $P \setminus \{a\}$. The 'conditional choice' construct $T \not\leftarrow \mathbf{bool} \not\rightarrow F$ denotes the process T if **bool** is true, and the process F otherwise. Lastly, from time to time, we express recursions by means of the equational notation $X = P$, rather than the functional $\mu X \centerdot P$ prescribed by the definition.

Except where explicitly noted, we are only interested in *well-timed* programs (the definition of which we give below). Recall also that we require all *delays* (parameter n in the terms $WAIT\ n$ and $P_1 \overset{n}{\rhd} P_2$) to be integral. This restriction (in the absence of which the central problems considered in this thesis become theoretically intractable[3]) is in practice extremely benign, because of the freedom to scale time units—its only real effects are to forbid, within a single program, either incommensurable delays (e.g., rational and irrational), or infinite sets of delays with arbitrarily small modular fractional differences[4]; both cases would clearly be unrealistic when dealing with real-world programs. For these reasons, many authors adopt similar conventions.

The following definitions are adapted from [Sch00]. A term is *time-active* if some strictly positive amount of time must elapse before it terminates. A term is *time-guarded for X* if any execution of it must consume some strictly positive amount of time before a recursive call for X can be reached. Lastly, a program is *well-timed* when all of its recursions are time-guarded. Note that, because all delays are integral, some "strictly positive amount of time" in this context automatically means at least one time unit.

Definition 2.1 *The collection of* time-active *terms is the smallest set of terms such that:*

- *$STOP$ is time-active;*
- *$WAIT\ n$ is time-active for $n \geqslant 1$;*

[3]As an example, let γ denote the famous Euler-Mascheroni constant, and consider the processes $N = WAIT\ 1\ ; \bar{a} \overset{0}{\rhd} N$ and $E = WAIT\ \gamma\ ; \bar{a} \overset{0}{\rhd} E$. If we let $P = N \parallel E$, deciding whether or not P can communicate an a is equivalent to deciding whether or not γ is rational, a well-known open problem.

[4]This second situation could in fact only occur were we to allow *infinite* (parameterised) mutual recursion.

- *If P is time-active, then so are $a \longrightarrow P$, $P \underset{\{\checkmark\}\cup A}{\parallel} Q$, $Q \underset{\{\checkmark\}\cup A}{\parallel} P$, $P \,;\, Q$, $Q \,;\, P$, $P \setminus A$, $f^{-1}(P)$, $f(P)$, $\mu X \centerdot P$, as well as $P \overset{n}{\rhd} Q$ for $n \geqslant 1$;*
- *If P_1 and P_2 are time-active, then so are $P_1 \overset{n}{\rhd} P_2$, $P_1 \Box P_2$, $P_1 \sqcap P_2$, $P_1 \underset{B}{\parallel} P_2$, and $P_1 \interleave P_2$;*
- *If $P(a)$ is time-active for each $a \in A$, then $a : A \longrightarrow P(a)$ is time-active.*

Definition 2.2 *For any program variable X, the collection of terms which are* time-guarded for *X is the smallest set of terms such that:*

- *$STOP$, $SKIP$, $WAIT\ n$, and $\mu X \centerdot P$ are time-guarded for X;*
- *$Y \neq X$ is time-guarded for X;*
- *If P is time-guarded for X, then so are $a \longrightarrow P$, $P \setminus A$, $f^{-1}(P)$, $f(P)$, $\mu Y \centerdot P$, as well as $P \overset{n}{\rhd} Q$ for $n \geqslant 1$;*
- *If P_1 and P_2 are time-guarded for X, then so are $P_1 \overset{n}{\rhd} P_2$, $P_1 \Box P_2$, $P_1 \sqcap P_2$, $P_1 \,;\, P_2$, $P_1 \underset{B}{\parallel} P_2$, and $P_1 \interleave P_2$;*
- *If $P(a)$ is time-guarded for X for each $a \in A$, then $a : A \longrightarrow P(a)$ is time-guarded for X;*
- *If P is time-guarded for X and time-active, then $P \,;\, Q$ is time-guarded for X.*

Definition 2.3 *A term is* well-timed *if every subterm of the form $\mu X \centerdot P$ is such that P is time-guarded for X.*

The collection of well-timed Timed CSP terms is denoted **TCSP**, whereas the set of well-timed Timed CSP programs is written $\overline{\textbf{TCSP}}$. Note that our grammar only allows us to produce well-timed terms; thus it is always understood that terms and programs are well-timed unless explicitly stated otherwise.

2.2 Semantic Modelling

Semantic models for process algebras come in many different flavours, the main characteristics of which we briefly sketch below.

- An *algebraic* semantics attempts to capture the meaning of a program through a series of *laws* which equate programs considered different only in an 'inessential' way. One such law, for example, might be $\mathit{SKIP} \ ; P = P$. Surprisingly little work has been carried out on algebraic semantics for Timed CSP; we will briefly examine the question later on.

- An *operational* semantics typically models programs as *labelled transition systems*, with nodes corresponding to machine states and edges corresponding to actions. This semantics represents most concretely the possible executions of a program, and figures prominently in the design of model checking algorithms. This semantics is often augmented with some notion of *bisimulation* which provides a mechanism for determining when two processes are equivalent. We will present and study several operational semantic models for Timed CSP in this thesis.

- A *denotational* semantics maps programs into some abstract model (typically a structured set or a category). This model is itself equipped with the operators present in the language, and the map is required to be *compositional*, i.e., it must be a homomorphism preserving these operators. In other words, the denotational value, or *meaning*, of any program is entirely determined by the meanings of its subcomponents. A denotational semantics is often predicated upon an algebraic or operational semantics, and the relationship between these models is usually carefully studied. Denotational semantics have typically been the main modelling devices for CSP-based languages, and figure prominently in this work. In essence, a Timed CSP program is represented by its set of *behaviours*, which are timed records of both the communications the program has been observed to engage in as well as those it has shunned.

- Other types of semantics, such as *testing* and *game semantics*, are not dealt with in this work.

One of the first decisions to take when modelling timed systems is whether time will be modelled in a dense (usually continuous), or discrete, fashion. Since the aim of this thesis is to study the interplay between these two paradigms, we naturally consider models of both types. The timed failures model $\mathcal{M}_{TF}$, first described in [RR86, Ree88], augmented with an operational semantics in [Sch95], and presented in the Chapter 3, serves as the continuous-time representative. Time is modelled by a global continuous clock, and recorded via a non-negative real-numbered timestamp on communicated or refused events.

We also develop denotational and operational semantics for two discrete-time models, the *discrete-time refusal testing model* $\mathcal{M}_R$ and the *discrete-time unstable refusal testing model* $\mathcal{M}_{UR}$, in which time is modelled by the regular communication of the special event *tock*[5], which processes are required to synchronise on. These models are developed in Chapters 5 and 7 respectively, and their relationship to the timed failures model studied in Chapters 6 and 8.

2.3 Notation

We list a number of definitions and conventions. These apply throughout the thesis, with further, more specific definitions introduced along as needed.

Sequences and sequence manipulations figure prominently in our models, thus the need for a certain amount of notation. Sequences can be either finite or infinite. We will primarily write $\langle a_1, a_2, \ldots, a_k \rangle$ to denote a finite sequence of length k comprising the elements $a_1, a_2, \ldots, a_k$, although on occasion we will also use the notation $[a_1, a_2, \ldots, a_k]$, to distinguish between different types of sequences. In the context of operational semantics we will even represent executions by sequences devoid of any brackets. Most of what follows applies equally to all three types of notation, with the context usually clarifying any ambiguity.

The empty sequence is written $\langle \rangle$ (or $[]$, etc.).

Let $u = \langle a_1, a_2, \ldots, a_k \rangle$ and $v = \langle b_1, b_2, \ldots, b_{k'} \rangle$. Their *concatenation* $u ^\frown v$ is the sequence $\langle a_1, a_2, \ldots, a_k, b_1, b_2, \ldots, b_{k'} \rangle$. If u is infinite we let $u ^\frown v \mathrel{\widehat{=}} u$. v can also be infinite, with the obvious interpretation.

We now define an *exponentiation* operation as follows: for u a sequence, we let

[5]'*tock*' rather than '*tick*', since the latter could be confused with the termination event $\checkmark$.

$$\begin{aligned} u^0 &\mathrel{\widehat{=}} \langle\rangle \\ u^{k+1} &\mathrel{\widehat{=}} u\frown u^k. \end{aligned}$$

We also let $u^\infty \mathrel{\widehat{=}} u\frown u\frown u\frown \ldots$.

The sequence u is a *prefix* of the sequence v if there exists a sequence u' such that $u\frown u' = v$. In this case we write $u \leq v$.

The sequence u *appears in* the sequence v if there exist sequences u' and u'' such that $u'\frown u\frown u'' = v$. In this case we write $u \,\mathsf{in}\, v$.

For $u = \langle a_1, a_2, \ldots, a_k\rangle$ a non-empty finite sequence, $\check{u} \mathrel{\widehat{=}} \langle a_2, a_3, \ldots, a_k\rangle$ and $\hat{u} \mathrel{\widehat{=}} \langle a_1, a_2, \ldots, a_{k-1}\rangle$ represent respectively u minus its first element and u minus its last element. We will never apply these operators to empty or infinite sequences.

The operator $\sharp$ returns the number of elements in a given sequence (taking on the value ∞ if the sequence is infinite), including repetitions. Thus $\sharp\langle a_1, a_2, \ldots, a_k\rangle \mathrel{\widehat{=}} k$.

The operator $\restriction$ denotes the restriction of a sequence to elements of a given set. Specifically, if A is a set, then we define inductively

$$\begin{array}{rcll} \langle\rangle \restriction A & \widehat{=} & \langle\rangle & \\ (\langle a\rangle\frown u) \restriction A & \widehat{=} & \langle a\rangle\frown(u \restriction A) & \text{if } a \in A \\ (\langle a\rangle\frown u) \restriction A & \widehat{=} & u \restriction A & \text{if } a \notin A. \end{array}$$

(This definition can be extended in the obvious way to infinite sequences, although we will not need this.) If A is the singleton $\{a\}$ we write $u \restriction a$ instead of $u \restriction \{a\}$. We will further overload this notation in Chapters 3, 5, and 7; the context, however, should always make clear what the intended meaning is.

If A and B are sets of sequences, we write AB to denote the set $\{u\frown v \,|\, u \in A \wedge v \in B\}$.

Lastly, if A is a set, we write $A^\star$ to represent the set of finite sequences all of whose elements belong to A: $A^\star \mathrel{\widehat{=}} \{u \mid \sharp u < \infty \wedge \forall\langle a\rangle \,\mathsf{in}\, u \bullet a \in A\}$.

Chapter 3

The Timed Failures Model

The timed failures model was developed by Reed and Roscoe as a continuous-time denotational model for the language of Timed CSP, which they had proposed as an extension of CSP [RR86, RR87, Ree88]. A number of different semantic models in the same vein have since appeared, with fundamentally minor overall differences between them; references include, in addition to the ones just mentioned, [Sch89, Dav91, DS95, RR99, Sch00]. The denotational model $\mathcal{M}_{TF}$ which we present here incorporates the essential features common to all of these continuous-time models.

$\mathcal{M}_{TF}$ also has a congruent operational semantics, given by Schneider in [Sch95], which we reproduce along with a number of results and a synopsis of the links to the denotational model that it enjoys. A particularly important result for us is the digitisation lemma (Lemma 3.11), which we present in a separate section.

Lastly, we review the notions of refinement, specification, and verification. In addition to the sources already mentioned, [Jac92] provided a slice of the material presented here.

Our presentation is expository in nature and rather brief—the only statement we prove is the digitisation lemma, which is original and requires a significant amount of technical machinery. We otherwise refer the reader to the sources above for a more thorough and complete treatment.

3.1 Denotational Semantics

We lay out the denotational model $\mathcal{M}_{TF}$ for Timed CSP and the associated semantic function $\mathcal{F}_T[\![\cdot]\!] : \overline{\mathbf{TCSP}} \longrightarrow \mathcal{M}_{TF}$. Sources include [Ree88], [RR99], and [Sch00].

Timed failures are pairs $(s, \aleph)$, with s a *timed trace* and $\aleph$ a *timed refusal.* A *timed trace* is a finite sequence of *timed events* $(t, a) \in \mathbb{R}^+ \times \Sigma^{\checkmark}$, such that the times are in non-decreasing order. A *timed refusal* is a set of timed events consisting of a finite union of *refusal tokens* $[t, t') \times A$ (with $0 \leqslant t \leqslant t' < \infty$ and $A \subseteq \Sigma^{\checkmark}$). A timed failure $(s, \aleph)$ is interpreted as an observation of a process in which the events that the process has engaged in are recorded in s, whereas the intervals during which other events have been refused are recorded in $\aleph$. The set of timed traces is denoted by TT, the set of timed refusals by $RSET$, and the set of timed failures by TF.

We define certain operations on these objects. We overload some operators, although context usually makes clear what the intended meaning is. In what follows $s \in TT$, $\aleph \in RSET$, $t, t' \in \mathbb{R}^+ \cup \{\infty\}$, $A \subseteq \Sigma^{\checkmark}$, and $a \in \Sigma^{\checkmark}$.

$$\begin{aligned}
s \upharpoonright t &\mathrel{\widehat{=}} s \upharpoonright [0, t] \times \Sigma^{\checkmark} \\
s \parallel\!\upharpoonright t &\mathrel{\widehat{=}} s \upharpoonright [0, t) \times \Sigma^{\checkmark} \\
s \upharpoonright A &\mathrel{\widehat{=}} s \upharpoonright [0, \infty) \times A \\
s \setminus A &\mathrel{\widehat{=}} s \upharpoonright (\Sigma - A) \\
\sigma(s) &\mathrel{\widehat{=}} \{a \mid s \upharpoonright \{a\} \neq \langle\rangle\} \\
\mathsf{begin}(\langle\rangle) &\mathrel{\widehat{=}} \infty \\
\mathsf{begin}(\langle(t, a)\rangle^\frown s) &\mathrel{\widehat{=}} t \\
\mathsf{end}(\langle\rangle) &\mathrel{\widehat{=}} 0 \\
\mathsf{end}(s^\frown\langle(t, a)\rangle) &\mathrel{\widehat{=}} t \\
\aleph \upharpoonright [t, t') &\mathrel{\widehat{=}} \aleph \cap [t, t') \times \Sigma^{\checkmark} \\
\aleph \parallel\!\upharpoonright t &\mathrel{\widehat{=}} \aleph \upharpoonright [0, t) \\
\aleph \upharpoonright A &\mathrel{\widehat{=}} \aleph \cap [0, \infty) \times A \\
\sigma(\aleph) &\mathrel{\widehat{=}} \{a \mid \aleph \upharpoonright \{a\} \neq \emptyset\} \\
\mathsf{begin}(\aleph) &\mathrel{\widehat{=}} \inf(\{t \mid \exists\, a \bullet (t, a) \in \aleph\} \cup \{\infty\}) \\
\mathsf{end}(\aleph) &\mathrel{\widehat{=}} \sup(\{t \mid \exists\, a \bullet (t, a) \in \aleph\} \cup \{0\}) \\
(s, \aleph) \parallel\!\upharpoonright t &\mathrel{\widehat{=}} (s \parallel\!\upharpoonright t, \aleph \parallel\!\upharpoonright t) \\
\mathsf{begin}((s, \aleph)) &\mathrel{\widehat{=}} \min(\mathsf{begin}(s), \mathsf{begin}(\aleph)) \\
\mathsf{end}((s, \aleph)) &\mathrel{\widehat{=}} \max\{\mathsf{begin}(s), \mathsf{begin}(\aleph)\}.
\end{aligned}$$

We also define the *information ordering* $\prec$ on timed failures as follows: $(s', \aleph') \prec (s, \aleph)$ if there exists $s'' \in TT$ such that $s = s' \frown s''$ and $\aleph' \subseteq \aleph \Uparrow \mathsf{begin}(s'')$.

Definition 3.1 *The* timed failures model $\mathcal{M}_{TF}$ *is the set of all* $P \subseteq TF$ *satisfying the following axioms, where* $s, s' \in TT$, $\aleph, \aleph' \in RSET$, $a \in \Sigma^{\checkmark}$, *and* $t, u \in \mathbb{R}^{+}$.

TF1 $(\langle\rangle, \emptyset) \in P$

TF2 $((s,\aleph) \in P \wedge (s',\aleph') \prec (s,\aleph)) \Rightarrow (s',\aleph') \in P$

TF3 $((s,\aleph) \in P \wedge u > 0) \Rightarrow$

$$\begin{aligned}&\exists\, \aleph' \in RSET \bullet \aleph \subseteq \aleph' \wedge (s,\aleph') \in P \wedge \forall (t,a) \in [0,u) \times \Sigma^{\checkmark} \bullet\\ &\quad ((t,a) \notin \aleph' \Rightarrow (s \upharpoonright t \frown \langle (t,a)\rangle, \aleph' \Uparrow t) \in P) \ \wedge\\ &\quad ((t > 0 \wedge \nexists\, \varepsilon > 0 \bullet [t-\varepsilon, t) \times \{a\} \subseteq \aleph') \Rightarrow\\ &\qquad (s \Uparrow t \frown \langle (t,a)\rangle, \aleph' \Uparrow t) \in P)\end{aligned}$$

TF4 $\forall\, t \geqslant 0 \bullet \exists\, n \in \mathbb{N} \bullet ((s,\aleph) \in P \wedge \mathsf{end}(s) \leqslant t) \Rightarrow \sharp(s) \leqslant n$

TF5 $(s \frown \langle (t, \checkmark)\rangle \frown s', \aleph) \in P \Rightarrow s' = \langle\rangle$.

These axioms have the following intuitive interpretations:

TF1: The least we can observe about a process is that it has communicated no events and refused none.

TF2: Any observation could have been replaced by another observation containing less information.

TF3: Tells us how observations can be extended, and postulates the existence of *complete behaviours up to time u* (for any u), which are maximal observations under the information ordering $\prec$. Specifically, this axiom says that any event not refusable at a certain time could have been performed at that time (albeit possibly only 'after' a number of events also occurring at that precise time); however, if the event in question was not refusable over some interval, however small, leading to the time in question, then that event could have been the 'first' to occur at that time. The fact that complete behaviours always exist also indicates that any behaviour can be extended to one in which no event is infinitely repeatedly offered and withdrawn over a finite period of time, since refusals are *finite* unions of refusal tokens.

TF4: A process cannot perform infinitely many events in a finite time. This axiom (known as *finite variability*) precludes such anomalies as *Zeno processes*.

TF5: Stipulates that a process which has terminated may not communicate any further.

Note that Axiom *TF3* implies that processes (can be observed to) run indefinitely. In other words, $\mathcal{M}_{TF}$ processes cannot exhibit *timestops*.

We will return to these axioms later on when building our discrete-time models and compare them to their discrete-time counterparts.

We now list a few more definitions before presenting the denotational mapping $\mathcal{F}_T[\![\cdot]\!]$.

For $s = \langle (t_1, a_1), (t_2, a_2), \ldots, (t_k, a_k) \rangle$ and $t \geqslant -\mathsf{begin}(s) = -t_1$, we let $s + t \mathrel{\widehat{=}} \langle (t_1 + t, a_1), (t_2 + t, a_2), \ldots, (t_k + t, a_k) \rangle$. (Of course, $s - t$ means $s + (-t)$). For any $t \in \mathbb{R}$, define $\aleph + t \mathrel{\widehat{=}} \{(t' + t, a) \mid (t', a) \in \aleph \wedge t' \geqslant -t\}$. Lastly, if $t \geqslant -\mathsf{begin}(s)$, then $(s, \aleph) + t \mathrel{\widehat{=}} (s + t, \aleph + t)$.

Given $B \subseteq \Sigma^{\checkmark}$, we define an untimed merging operator $(\cdot) \underset{B}{\tilde{\|}} (\cdot) : TT \times TT \longrightarrow \mathcal{P}((\mathbb{R}^+ \times \Sigma^{\checkmark})^\star)$, en route to defining an adequate parallel operator on timed traces. In what follows, $s \in (\mathbb{R}^+ \times \Sigma^{\checkmark})^\star$, $s_1, s_2, s_1', s_2' \in TT$, $t \in \mathbb{R}^+$, and $a \in \Sigma^{\checkmark}$.

$$\begin{aligned}
\langle\rangle \in s_1 \underset{B}{\tilde{\|}} s_2 \quad &\Leftrightarrow \quad s_1 = s_2 = \langle\rangle \\
\langle (t, a) \rangle^\frown s \in s_1 \underset{B}{\tilde{\|}} s_2 \quad &\Leftrightarrow \quad (a \in B \wedge s_1 = \langle (t, a) \rangle^\frown s_1' \wedge \\
&\qquad\quad s_2 = \langle (t, a) \rangle^\frown s_2' \wedge s \in s_1' \underset{B}{\tilde{\|}} s_2') \vee \\
&\quad\ \ (a \notin B \wedge s_1 = \langle (t, a) \rangle^\frown s_1' \wedge s \in s_1' \underset{B}{\tilde{\|}} s_2) \vee \\
&\quad\ \ (a \notin B \wedge s_2 = \langle (t, a) \rangle^\frown s_2' \wedge s \in s_1 \underset{B}{\tilde{\|}} s_2').
\end{aligned}$$

We now define $(\cdot) \underset{B}{\|} (\cdot) : TT \times TT \longrightarrow \mathcal{P}(TT)$ by imposing the proper ordering on timed events, and throwing out traces which fail to satisfy Axiom *TF5*. The notation is as above.

$$s \in s_1 \underset{B}{\|} s_2 \quad \Leftrightarrow \quad s \in s_1 \underset{B}{\tilde{\|}} s_2 \wedge s \in TT \wedge (s = s'^\frown \langle (t, \checkmark) \rangle^\frown s'' \Rightarrow s'' = \langle\rangle).$$

We let $s_1 \,|||\, s_2 \mathrel{\widehat{=}} s_1 \underset{\emptyset}{\|} s_2$, for any $s_1, s_2 \in TT$.

A renaming function $f : \Sigma \longrightarrow \Sigma$ has an obvious extension to timed traces and timed refusals: if $s = \langle (t_1, a_1), (t_2, a_2), \ldots, (t_k, a_k) \rangle \in TT$ and $\aleph \in RSET$, then $f(s) \mathrel{\widehat{=}} \langle (t_1, f(a_1)), (t_2, f(a_2)), \ldots, (t_k, f(a_k)) \rangle$, $f(\aleph) \mathrel{\widehat{=}} \{(t, f(a)) \mid (t, a) \in \aleph\}$, and $f^{-1}(\aleph) \mathrel{\widehat{=}} \{(t, a) \mid (t, f(a)) \in \aleph\}$. Here f is extended to $\Sigma^{\checkmark}$ by setting $f(\checkmark) \mathrel{\widehat{=}} \checkmark$.

The *lambda abstraction* $\lambda\, x.F(x)$ formally represents the function F. For example, $\lambda\, x.(x^2+3)$ denotes the function F such that, for all x, $F(x) = x^2+3$.

A *semantic binding* is a function $\rho : \mathit{VAR} \longrightarrow \mathcal{M}_{TF}$. Given a semantic binding ρ and a process $P \in \mathcal{M}_{TF}$, we write $\rho[X := P]$ to denote a further semantic binding that is the same as ρ except that it returns P for X instead of $\rho(X)$. Naturally, if x is a variable ranging over $\mathcal{M}_{TF}$ (i.e., x is a *bona fide* variable, *not* an element of VAR!), $\lambda\, x.\rho[X := x]$ represents a function which, when fed an element P of $\mathcal{M}_{TF}$, returns a semantic binding (one that is identical to ρ but for mapping X to P).

If $F : \mathcal{M}_{TF} \longrightarrow \mathcal{M}_{TF}$ has a unique fixed point, we denote it by $\mathsf{fix}(F)$. ($P \in \mathcal{M}_{TF}$ is a fixed point of F if $F(P) = P$.)

We now define the function $\mathcal{F}_T[\![\cdot]\!]$ inductively over the structure of Timed CSP terms. Since a term $P \in \mathbf{TCSP}$ may contain free variables, we also require a semantic binding ρ in order to assign $\mathcal{M}_{TF}$ processes to the free variables of P.[1] In what follows, $s, s_1, s_2 \in TT$, $\aleph, \aleph_1, \aleph_2 \in RSET$, $t \in \mathbb{R}^+$, $a \in \Sigma$, $A \subseteq \Sigma$, and $B \subseteq \Sigma^{\checkmark}$. The rules are as follows:

$$\begin{aligned}
\mathcal{F}_T[\![\mathit{STOP}]\!]\rho &\mathrel{\widehat{=}} \{(\langle\rangle, \aleph)\} \\
\mathcal{F}_T[\![\mathit{SKIP}]\!]\rho &\mathrel{\widehat{=}} \{(\langle\rangle, \aleph) \mid \checkmark \notin \sigma(\aleph)\} \cup \\
&\quad\ \{(\langle (t, \checkmark) \rangle, \aleph) \mid t \geqslant 0 \wedge \checkmark \notin \sigma(\aleph \Vert t)\} \\
\mathcal{F}_T[\![\mathit{WAIT}\ n]\!]\rho &\mathrel{\widehat{=}} \{(\langle\rangle, \aleph) \mid \checkmark \notin \sigma(\aleph \upharpoonright [n, \infty))\} \cup \\
&\quad\ \{(\langle (t, \checkmark) \rangle, \aleph) \mid t \geqslant n \wedge \checkmark \notin \sigma(\aleph \upharpoonright [n, t))\} \\
\mathcal{F}_T[\![P_1 \overset{n}{\rhd} P_2]\!]\rho &\mathrel{\widehat{=}} \{(s, \aleph) \mid \mathsf{begin}(s) \leqslant n \wedge (s, \aleph) \in \mathcal{F}_T[\![P_1]\!]\rho\} \cup \\
&\quad\ \{(s, \aleph) \mid \mathsf{begin}(s) \geqslant n \wedge (\langle\rangle, \aleph \Vert n) \in \mathcal{F}_T[\![P_1]\!]\rho\ \wedge \\
&\quad\quad (s, \aleph) - t \in \mathcal{F}_T[\![P_2]\!]\rho\}
\end{aligned}$$

[1]In reality, we are actually defining $\mathcal{F}_T : \mathbf{TCSP} \times \mathit{BIND} \longrightarrow \mathcal{M}_{TF}$, where BIND stands for the set of all semantic bindings. Since our interest ultimately lies exclusively in the subset $\overline{\mathbf{TCSP}}$ of $\mathbf{TCSP}$ (in which case the choice of semantic binding becomes irrelevant), we shall not be overly concerned with this point.

$$\begin{array}{rcl}
\mathcal{F}_T[\![a \longrightarrow P]\!]\rho & \widehat{=} & \{(\langle\rangle, \aleph) \mid a \notin \sigma(\aleph)\} \cup \\
& & \{(\langle(t,a)\rangle^\frown s, \aleph) \mid t \geqslant 0 \wedge a \notin \sigma(\aleph \Vert\!\upharpoonright t) \wedge \\
& & \quad \mathsf{begin}(s) \geqslant t \wedge (s,\aleph) - t \in \mathcal{F}_T[\![P]\!]\rho\} \\
\mathcal{F}_T[\![a : A \longrightarrow P(a)]\!]\rho & \widehat{=} & \{(\langle\rangle, \aleph) \mid A \cap \sigma(\aleph) = \emptyset\} \cup \\
& & \{(\langle(t,a)\rangle^\frown s, \aleph) \mid a \in A \wedge t \geqslant 0 \wedge \\
& & \quad A \cap \sigma(\aleph \Vert\!\upharpoonright t) = \emptyset \wedge \mathsf{begin}(s) \geqslant t \wedge \\
& & \quad (s,\aleph) - t \in \mathcal{F}_T[\![P(a)]\!]\rho\} \\
\mathcal{F}_T[\![P_1 \Box P_2]\!]\rho & \widehat{=} & \{(\langle\rangle, \aleph) \mid (\langle\rangle, \aleph) \in \mathcal{F}_T[\![P_1]\!]\rho \cap \mathcal{F}_T[\![P_2]\!]\rho\} \cup \\
& & \{(s,\aleph) \mid s \neq \langle\rangle \wedge (s,\aleph) \in \mathcal{F}_T[\![P_1]\!]\rho \cup \mathcal{F}_T[\![P_2]\!]\rho \wedge \\
& & \quad (\langle\rangle, \aleph \Vert\!\upharpoonright \mathsf{begin}(s)) \in \mathcal{F}_T[\![P_1]\!]\rho \cap \mathcal{F}_T[\![P_2]\!]\rho\} \\
\mathcal{F}_T[\![P_1 \sqcap P_2]\!]\rho & \widehat{=} & \mathcal{F}_T[\![P_1]\!]\rho \cup \mathcal{F}_T[\![P_2]\!]\rho \\
\mathcal{F}_T[\![P_1 \underset{B}{\Vert} P_2]\!]\rho & \widehat{=} & \{(s,\aleph) \mid \exists\, s_1, s_2, \aleph_1, \aleph_2 \bullet s \in s_1 \underset{B}{\Vert} s_2 \wedge \\
& & \quad \aleph_1 \upharpoonright (\Sigma - B) = \aleph_2 \upharpoonright (\Sigma - B) = \aleph \upharpoonright (\Sigma - B) \wedge \\
& & \quad \aleph_1 \upharpoonright B \cup \aleph_2 \upharpoonright B = \aleph \upharpoonright B \wedge \\
& & \quad (s_1, \aleph_1 \Vert\!\upharpoonright \mathsf{begin}(s \upharpoonright \{\checkmark\})) \in \mathcal{F}_T[\![P_1]\!]\rho \wedge \\
& & \quad (s_2, \aleph_2 \Vert\!\upharpoonright \mathsf{begin}(s \upharpoonright \{\checkmark\})) \in \mathcal{F}_T[\![P_2]\!]\rho\} \\
\mathcal{F}_T[\![P_1 \interleave P_2]\!]\rho & \widehat{=} & \{(s,\aleph) \mid \exists\, s_1, s_2 \bullet s \in s_1 \interleave s_2 \wedge \\
& & \quad (s_1, \aleph \Vert\!\upharpoonright \mathsf{begin}(s \upharpoonright \{\checkmark\})) \in \mathcal{F}_T[\![P_1]\!]\rho \wedge \\
& & \quad (s_2, \aleph \Vert\!\upharpoonright \mathsf{begin}(s \upharpoonright \{\checkmark\})) \in \mathcal{F}_T[\![P_2]\!]\rho\} \\
\mathcal{F}_T[\![P_1 \,;\, P_2]\!]\rho & \widehat{=} & \{(s,\aleph) \mid \checkmark \notin \sigma(s) \wedge \\
& & \quad (s, \aleph \cup ([0, \mathsf{end}((s,\aleph))) \times \{\checkmark\})) \in \mathcal{F}_T[\![P_1]\!]\rho\} \cup \\
& & \{(s_1{}^\frown s_2, \aleph) \mid \checkmark \notin \sigma(s_1) \wedge (s_2,\aleph) - t \in \mathcal{F}_T[\![P_2]\!]\rho \wedge \\
& & \quad (s_1{}^\frown \langle(t,\checkmark)\rangle, \aleph \Vert\!\upharpoonright t \cup ([0,t) \times \{\checkmark\})) \in \mathcal{F}_T[\![P_1]\!]\rho\} \\
\mathcal{F}_T[\![P \setminus A]\!]\rho & \widehat{=} & \{(s \setminus A, \aleph) \mid (s, \aleph \cup ([0, \mathsf{end}((s,\aleph))) \times A)) \in \mathcal{F}_T[\![P]\!]\rho\} \\
\mathcal{F}_T[\![f^{-1}(P)]\!]\rho & \widehat{=} & \{(s,\aleph) \mid (f(s), f(\aleph)) \in \mathcal{F}_T[\![P]\!]\rho\} \\
\mathcal{F}_T[\![f(P)]\!]\rho & \widehat{=} & \{(f(s),\aleph) \mid (s, f^{-1}(\aleph)) \in \mathcal{F}_T[\![P]\!]\rho\} \\
\mathcal{F}_T[\![X]\!]\rho & \widehat{=} & \rho(X) \\
\mathcal{F}_T[\![\mu\, X \bullet P]\!]\rho & \widehat{=} & \mathsf{fix}(\lambda\, x.\mathcal{F}_T[\![P]\!](\rho[X := x])).
\end{array}$$

The following results are due to Reed [Ree88].

Proposition 3.1 *Well-definedness: for any term P, and any semantic binding ρ, $\mathcal{F}_T[\![P]\!]\rho \in \mathcal{M}_{TF}$.*

Proposition 3.2 *If P is a Timed CSP* program *(i.e., $P \in \overline{\mathbf{TCSP}}$), then, for any semantic bindings ρ and ρ', $\mathcal{F}_T[\![P]\!]\rho = \mathcal{F}_T[\![P]\!]\rho'$.*

We will henceforth drop mention of semantic bindings when calculating the semantics of programs.

3.2 Operational Semantics

The contents and style of this section derive essentially from [Sch95]. We present a collection of *inference rules* with the help of which any $\overline{\mathbf{TCSP}}$ program can be assigned a unique *labelled transition system*, or *LTS*. Such LTS's are the operational counterparts of denotational process representations. A fuller and more formal discussion of operational semantics (especially in CSP contexts) can be found in [Ros97].

An operational semantics can usually ascribe behaviours to programs that are not necessarily well-timed; moreover, because it is state-based, we must equip it with a means to describe intermediate computational states which in certain cases **TCSP** notation is unable to do. For this reason, we will consider terms generated by the following less restrictive grammar:

$$\begin{array}{lcl} P & := & STOP \mid SKIP \mid WAIT\ t \mid P_1 \overset{t}{\rhd} P_2 \mid \\ & & a \longrightarrow P \mid a : A \longrightarrow P(a) \mid P_1 \Box P_2 \mid P_1 \sqcap P_2 \mid \\ & & P_1 \underset{B}{\|} P_2 \mid P_1 \mid\mid\mid P_2 \mid P_1 \,;\, P_2 \mid P \setminus A \mid \\ & & f^{-1}(P) \mid f(P) \mid X \mid \mu X \bullet P. \end{array}$$

Here t can be any non-negative real number, and we have dropped any requirement of well-timedness.[2] The remainder of our conventions about Timed CSP syntax (see Section 2.1) however apply. We denote the set of terms which this grammar generates by $NODE_{TF}$ and the set of closed terms (not containing free variables) by $\overline{NODE}_{TF}$, dropping the subscripts when no confusion is likely. Elements of $NODE$ are called *(open) nodes* whereas elements of $\overline{NODE}$ are called *(closed) nodes.* We insist that our inference

[2] Forgoing well-timedness is not absolutely necessary, but does no harm and certainly greatly simplifies matters in the presence of fractional delays; in addition, we will see later on why it is convenient to be able to write certain specifications in terms of non-well-timed nodes.

rules only apply to closed nodes. Note that $\mathbf{TCSP} \subseteq \mathit{NODE}$ and $\overline{\mathbf{TCSP}} \subseteq \overline{\mathit{NODE}}$.

We list a few notational conventions: a and b stand for (non-*tock*) visible events, i.e., belong to $\Sigma^{\checkmark}$. $A \subseteq \Sigma$ and $B \subseteq \Sigma^{\checkmark}$. μ can be a visible event or a silent one ($\mu \in \Sigma^{\checkmark} \cup \{\tau\}$). $P \xrightarrow{\mu} P'$ means that the closed node P can perform an immediate and instantaneous μ-*transition*, and become the closed node P' (communicating μ in the process if μ is a visible event). $P \overset{\mu}{\not\longrightarrow}$ means that P cannot possibly do a μ at that particular time. $P \overset{t}{\rightsquigarrow} P'$ means that P can become P' simply by virtue of letting t units of time elapse, where t is a non-negative real number. If P and Q are open nodes and $X \in \mathit{VAR}$, $P[Q/X]$ represents the node P with Q substituted for every free occurrence of X.

The inference rules take the general form

$$\frac{\textit{antecedent(s)}}{\textit{conclusion}}\ [\,\textit{side condition}\,]$$

where either antecedents or side condition, or both, can be absent. (The side condition is an antecedent typically dealing with matters other than transitions or evolutions.) The rules are as follows.

$$\frac{}{\mathit{STOP} \overset{t}{\rightsquigarrow} \mathit{STOP}} \tag{3.1}$$

$$\frac{}{\mathit{SKIP} \overset{t}{\rightsquigarrow} \mathit{SKIP}} \tag{3.2}$$

$$\frac{}{\mathit{SKIP} \xrightarrow{\checkmark} \mathit{STOP}} \tag{3.3}$$

$$\frac{}{\mathit{WAIT}\ u \overset{t}{\rightsquigarrow} \mathit{WAIT}\ (u - t)}\ [\, t \leqslant u \,] \tag{3.4}$$

$$\frac{}{\mathit{WAIT}\ 0 \xrightarrow{\tau} \mathit{SKIP}} \tag{3.5}$$

$$\frac{P_1 \overset{t}{\rightsquigarrow} P_1'}{P_1 \overset{u}{\rhd} P_2 \overset{t}{\rightsquigarrow} P_1' \overset{u-t}{\rhd} P_2}\ [\, t \leqslant u \,] \tag{3.6}$$

$$\frac{}{P_1 \overset{0}{\rhd} P_2 \xrightarrow{\tau} P_2} \tag{3.7}$$

$$\frac{P_1 \xrightarrow{\tau} P_1'}{P_1 \overset{u}{\rhd} P_2 \xrightarrow{\tau} P_1' \overset{u}{\rhd} P_2} \tag{3.8}$$

$$\frac{P_1 \xrightarrow{a} P_1'}{P_1 \overset{u}{\rhd} P_2 \xrightarrow{a} P_1'} \tag{3.9}$$

$$\frac{}{(a \longrightarrow P) \overset{t}{\rightsquigarrow} (a \longrightarrow P)} \tag{3.10}$$

$$\frac{}{(a \longrightarrow P) \xrightarrow{a} P} \tag{3.11}$$

$$\frac{}{(a : A \longrightarrow P(a)) \overset{t}{\rightsquigarrow} (a : A \longrightarrow P(a))} \tag{3.12}$$

$$\frac{}{(a : A \longrightarrow P(a)) \xrightarrow{b} P(b)}\ [\, b \in A \,] \tag{3.13}$$

$$\frac{P_1 \overset{t}{\rightsquigarrow} P_1' \quad P_2 \overset{t}{\rightsquigarrow} P_2'}{P_1 \,\Box\, P_2 \overset{t}{\rightsquigarrow} P_1' \,\Box\, P_2'} \tag{3.14}$$

$$\frac{P_1 \xrightarrow{\tau} P_1'}{P_1 \,\Box\, P_2 \xrightarrow{\tau} P_1' \,\Box\, P_2} \qquad \frac{P_2 \xrightarrow{\tau} P_2'}{P_1 \,\Box\, P_2 \xrightarrow{\tau} P_1 \,\Box\, P_2'} \tag{3.15}$$

$$\frac{P_1 \xrightarrow{a} P_1'}{P_1 \mathbin{\square} P_2 \xrightarrow{a} P_1'} \qquad \frac{P_2 \xrightarrow{a} P_2'}{P_1 \mathbin{\square} P_2 \xrightarrow{a} P_2'} \tag{3.16}$$

$$\frac{}{P_1 \sqcap P_2 \xrightarrow{\tau} P_1} \qquad \frac{}{P_1 \sqcap P_2 \xrightarrow{\tau} P_2} \tag{3.17}$$

$$\frac{P_1 \overset{t}{\rightsquigarrow} P_1' \quad P_2 \overset{t}{\rightsquigarrow} P_2'}{P_1 \underset{B}{\parallel} P_2 \overset{t}{\rightsquigarrow} P_1' \underset{B}{\parallel} P_2'} \tag{3.18}$$

$$\frac{P_1 \xrightarrow{\mu} P_1'}{P_1 \underset{B}{\parallel} P_2 \xrightarrow{\mu} P_1' \underset{B}{\parallel} P_2} \; [\, \mu \notin B, \mu \neq \checkmark \,] \tag{3.19a}$$

$$\frac{P_2 \xrightarrow{\mu} P_2'}{P_1 \underset{B}{\parallel} P_2 \xrightarrow{\mu} P_1 \underset{B}{\parallel} P_2'} \; [\, \mu \notin B, \mu \neq \checkmark \,] \tag{3.19b}$$

$$\frac{P_1 \xrightarrow{a} P_1' \quad P_2 \xrightarrow{a} P_2'}{P_1 \underset{B}{\parallel} P_2 \xrightarrow{a} P_1' \underset{B}{\parallel} P_2'} \; [\, a \in B \,] \tag{3.20}$$

$$\frac{P_1 \xrightarrow{\checkmark} P_1'}{P_1 \underset{B}{\parallel} P_2 \xrightarrow{\checkmark} P_1'} \; [\, \checkmark \notin B \,] \qquad \frac{P_2 \xrightarrow{\checkmark} P_2'}{P_1 \underset{B}{\parallel} P_2 \xrightarrow{\checkmark} P_2'} \; [\, \checkmark \notin B \,] \tag{3.21}$$

$$\frac{P_1 \overset{t}{\rightsquigarrow} P_1' \quad P_2 \overset{t}{\rightsquigarrow} P_2'}{P_1 \mathbin{|||} P_2 \overset{t}{\rightsquigarrow} P_1' \mathbin{|||} P_2'} \tag{3.22}$$

$$\frac{P_1 \xrightarrow{\mu} P_1'}{P_1 \mathbin{|||} P_2 \xrightarrow{\mu} P_1' \mathbin{|||} P_2} \; [\, \mu \neq \checkmark \,] \tag{3.23a}$$

$$\frac{P_2 \xrightarrow{\mu} P_2'}{P_1 \mathbin{|||} P_2 \xrightarrow{\mu} P_1 \mathbin{|||} P_2'} \; [\, \mu \neq \checkmark \,] \tag{3.23b}$$

$$\frac{P_1 \xrightarrow{\checkmark} P_1'}{P_1 \mid\mid\mid P_2 \xrightarrow{\checkmark} P_1'} \qquad \frac{P_2 \xrightarrow{\checkmark} P_2'}{P_1 \mid\mid\mid P_2 \xrightarrow{\checkmark} P_2'} \tag{3.24}$$

$$\frac{P_1 \overset{t}{\rightsquigarrow} P_1' \quad P_1 \overset{\checkmark}{\not\longrightarrow}}{P_1 \,;\, P_2 \overset{t}{\rightsquigarrow} P_1' \,;\, P_2} \tag{3.25}$$

$$\frac{P_1 \xrightarrow{\checkmark} P_1'}{P_1 \,;\, P_2 \xrightarrow{\tau} P_2} \tag{3.26}$$

$$\frac{P_1 \xrightarrow{\mu} P_1'}{P_1 \,;\, P_2 \xrightarrow{\mu} P_1' \,;\, P_2} \; [\, \mu \neq \checkmark \,] \tag{3.27}$$

$$\frac{P \overset{t}{\rightsquigarrow} P' \quad \forall\, a \in A \centerdot P \overset{a}{\not\longrightarrow}}{P \setminus A \overset{t}{\rightsquigarrow} P' \setminus A} \tag{3.28}$$

$$\frac{P \xrightarrow{a} P'}{P \setminus A \xrightarrow{\tau} P' \setminus A} \; [\, a \in A \,] \tag{3.29}$$

$$\frac{P \xrightarrow{\mu} P'}{P \setminus A \xrightarrow{\mu} P' \setminus A} \; [\, \mu \notin A \,] \tag{3.30}$$

$$\frac{P \overset{t}{\rightsquigarrow} P'}{f^{-1}(P) \overset{t}{\rightsquigarrow} f^{-1}(P')} \tag{3.31}$$

$$\frac{P \xrightarrow{\mu} P'}{f^{-1}(P) \xrightarrow{\mu} f^{-1}(P')} \; [\, \mu \in \{\tau, \checkmark\} \,] \tag{3.32}$$

$$\frac{P \xrightarrow{f(a)} P'}{f^{-1}(P) \xrightarrow{a} f^{-1}(P')} \tag{3.33}$$

$$\frac{P \overset{t}{\rightsquigarrow} P'}{f(P) \overset{t}{\rightsquigarrow} f(P')} \tag{3.34}$$

$$\frac{P \xrightarrow{\mu} P'}{f(P) \xrightarrow{\mu} f(P')}\ [\,\mu \in \{\tau, \checkmark\}\,] \tag{3.35}$$

$$\frac{P \xrightarrow{a} P'}{f(P) \xrightarrow{f(a)} f(P')} \tag{3.36}$$

$$\frac{}{\mu X \centerdot P \xrightarrow{\tau} P[(\mu X \centerdot P)/X].} \tag{3.37}$$

The reader may have noticed that Rules 3.25 and 3.28 incorporate *negative premisses* ($P_1 \overset{\checkmark}{\not\longrightarrow}$ and $P \overset{a}{\not\longrightarrow}$), which could potentially yield an inconsistent definition. This does not occur, for the following reason: notice that the $\xrightarrow{x}$ relation can be defined, *independently of the* $\overset{t}{\rightsquigarrow}$ *relation*, as the smallest relation satisfying the relevant subset of rules, since no negative premisses are involved in its definition. Once the $\xrightarrow{x}$ relation has been defined, the $\overset{t}{\rightsquigarrow}$ relation can then itself be defined. Since the negative premisses are all phrased in terms of the previously defined (and fixed) $\xrightarrow{x}$ relation, they do not pose any problem.

We now present a number of results about the operational semantics. We begin with some definitions.

If P and Q are open nodes, we write $P \equiv Q$ to indicate that P and Q are syntactically identical.

If P is a closed node, we define $\mathsf{init}^{\tau}_{TF}(P)$ to be the set of visible and silent events that P can immediately perform: $\mathsf{init}^{\tau}_{TF}(P) \mathrel{\widehat{=}} \{\mu \mid P \xrightarrow{\mu}\}$. We also write $\mathsf{init}_{TF}(P)$ to represent the set of visible events that P can immediately perform: $\mathsf{init}_{TF}(P) \mathrel{\widehat{=}} \mathsf{init}^{\tau}_{TF}(P) \cap \Sigma^{\checkmark}$. We will usually write $\mathsf{init}^{\tau}(P)$ and $\mathsf{init}(P)$ for short when no confusion is likely.

For P a closed node, we define an *execution* of P to be a sequence $e = P_0 \overset{z_1}{\longmapsto} P_1 \overset{z_2}{\longmapsto} \ldots \overset{z_n}{\longmapsto} P_n$ (with $n \geqslant 0$), where $P_0 \equiv P$, the P_i's are nodes, and each subsequence $P_i \overset{z_{i+1}}{\longmapsto} P_{i+1}$ in e is either a transition $P_i \xrightarrow{\mu} P_{i+1}$ (with

$z_{i+1} = \mu$), or an evolution $P_i \overset{t}{\rightsquigarrow} P_{i+1}$ (with $z_{i+1} = t$). In addition, every such transition or evolution must be validly allowed by the operational inference Rules 3.1–3.37. The set of executions of P is written $\mathsf{exec}_{TF}(P)$, or $\mathsf{exec}(P)$ for short when no confusion with notation introduced later on is likely. By convention, writing down a transition (or sequence thereof) such as $P \xrightarrow{a} P'$ is equivalent to stating that $P \xrightarrow{a} P' \in \mathsf{exec}(P)$; the same, naturally, goes for evolutions.

For P a closed node, the P-rooted graph, or *labelled transition system*, incorporating all of P's possible executions is denoted $\mathsf{LTS}_{TF}(P)$, or $\mathsf{LTS}(P)$ if no confusion is likely.

Every execution e gives rise to a *timed τ-trace* $\mathsf{abs}(e)$ in the obvious way, by removing nodes and evolutions from the execution, but recording events' time of occurrence in agreement with e's evolutions. (A *timed τ-trace* is a timed trace over $\Sigma^{\checkmark} \cup \{\tau\}$.) The formal inductive definition of abs is as follows:

$$\begin{aligned} \mathsf{abs}(P) &\mathrel{\widehat{=}} \langle\rangle \\ \mathsf{abs}((P \xrightarrow{\mu})^\frown e) &\mathrel{\widehat{=}} \langle(0,\mu)\rangle^\frown \mathsf{abs}(e) \\ \mathsf{abs}((P \overset{t}{\rightsquigarrow})^\frown e) &\mathrel{\widehat{=}} \mathsf{abs}(e) + t. \end{aligned}$$

The *duration* of an evolution e is equal to the sum of its evolutions: $\mathsf{dur}(e) \mathrel{\widehat{=}} \mathsf{end}(\mathsf{abs}(e))$.

We then have the following results, adapted from [Sch95]. (Here P, P', P'' are closed nodes, t, t' are non-negative real numbers, etc.).

Proposition 3.3 *Time determinacy:*

$$(P \overset{t}{\rightsquigarrow} P' \wedge P \overset{t}{\rightsquigarrow} P'') \Rightarrow P' \equiv P''.$$

Proposition 3.4 *Persistency—the set of possible initial visible events remains constant under evolution:*

$$P \overset{t}{\rightsquigarrow} P' \Rightarrow \mathsf{init}(P) = \mathsf{init}(P').$$

Proposition 3.5 *Time continuity:*

$$P \overset{t+t'}{\rightsquigarrow} P' \Leftrightarrow \exists\, P'' \bullet P \overset{t}{\rightsquigarrow} P'' \overset{t'}{\rightsquigarrow} P'.$$

Proposition 3.6 *Maximal progress, or τ-urgency:*

$$P \xrightarrow{\tau} \Rightarrow \forall\, t > 0 \centerdot \nexists\, P' \centerdot P \overset{t}{\rightsquigarrow} P'.$$

Corollary 3.7

$$(P \overset{t}{\rightsquigarrow} P' \xrightarrow{\tau} \wedge\, P \overset{t'}{\rightsquigarrow} P'' \xrightarrow{\tau}) \Rightarrow t = t'.$$

Proposition 3.8 *A node P can always evolve up to the time of the next τ action, or up to any time if no τ action lies ahead:*

$$\forall\, t \geqslant 0 \centerdot (\nexists\, P' \centerdot P \overset{t}{\rightsquigarrow} P') \Rightarrow (P \xrightarrow{\tau} \vee\, \exists\, t' < t, P'' \centerdot P \overset{t'}{\rightsquigarrow} P'' \xrightarrow{\tau}).$$

Proposition 3.9 *Finite variability—a* program $P \in \overline{\mathbf{TCSP}}$ *cannot perform unboundedly many actions in a finite amount of time:*

$$\forall\, t \geqslant 0 \centerdot \exists\, n = n(P,t) \in \mathbb{N} \centerdot \forall\, e \in \mathsf{exec}(P) \centerdot \mathsf{dur}(e) \leqslant t \Rightarrow \sharp\mathsf{abs}(e) \leqslant n.$$

We remark that we owe this result to the fact that programs are well-timed. Note also that this notion of finite variability is stronger than that postulated by Axiom *TF4*, since it concerns both visible and silent events.

In [Sch95], it is shown that the operational semantics just given is congruent to the denotational semantics of Section 3.1, in a sense which we make precise below. We begin with some definitions.

A set of visible events is *refused* by a node P if P is stable (cannot perform a τ-transition) and has no valid initial transition labelled with an event from that set. Thus for $A \subseteq \Sigma^{\checkmark}$, we write P **ref** A if $P \not\xrightarrow{\tau} \wedge\, A \cap \mathsf{init}(P) = \emptyset$.

An execution e of a node P is said to *fail* a timed failure $(s, \aleph)$ if the timed trace s corresponds to the execution e, and the nodes of e can always refuse the relevant parts of $\aleph$; we then write e **fail** $(s, \aleph)$. The relation is defined inductively on e as follows:

$$\begin{array}{rcl}
P\ \mathbf{fail}\ (\langle\rangle, \emptyset) & \Leftrightarrow & \mathbf{true} \\
(P \xrightarrow{\tau})^\frown e'\ \mathbf{fail}\ (s, \aleph) & \Leftrightarrow & e'\ \mathbf{fail}\ (s, \aleph) \\
(P \xrightarrow{a})^\frown e'\ \mathbf{fail}\ (\langle(0,a)\rangle^\frown s', \aleph) & \Leftrightarrow & a \neq \tau \wedge e'\ \mathbf{fail}\ (s', \aleph) \\
(P \overset{t}{\rightsquigarrow})^\frown e'\ \mathbf{fail}\ (s, \aleph) & \Leftrightarrow & P\ \mathbf{ref}\ \sigma(\aleph \Vert\!\!\upharpoonright t) \wedge e'\ \mathbf{fail}\ (s - t, \aleph - t).
\end{array}$$

Finally, we define the function Φ_{TF}, which extracts the denotational representation of a node from its set of executions.

Definition 3.2 *For $P \in \overline{NODE}$, we set*

$$\Phi_{TF}(P) \quad \widehat{=} \quad \{(s,\aleph) \mid \exists\, e \in \mathsf{exec}_{TF}(P) \,\bullet\, e \ \mathbf{fail}\ (s,\aleph)\}.$$

We can now state the chief congruence result:

Theorem 3.10 *For any $\overline{\mathbf{TCSP}}$ program P, we have*

$$\Phi_{TF}(P) = \mathcal{F}_T[\![P]\!].$$

3.3 The Digitisation Lemma

The single result of this section, which we call the *digitisation lemma*, will enable us to relate this chapter's continuous-time semantics for Timed CSP to the discrete-time semantics introduced in Chapters 5 and 7. We first need a small piece of notation. Let $t \in \mathbb{R}^+$, and let $0 \leqslant \varepsilon \leqslant 1$ be a real number. Decompose t into its integral and fractional parts, thus: $t = \lfloor t \rfloor + \dot{t}$. (Here $\lfloor t \rfloor$ represents the greatest integer less than or equal to t.) If $\dot{t} < \varepsilon$, let $[t]_\varepsilon \mathrel{\widehat{=}} \lfloor t \rfloor$, otherwise let $[t]_\varepsilon \mathrel{\widehat{=}} \lceil t \rceil$. (Naturally, $\lceil t \rceil$ denotes the least integer greater than or equal to t.) The $[\cdot]_\varepsilon$ operator therefore shifts the value of a real number t to the preceding or following integer, depending on whether the fractional part of t is less than the 'pivot' ε or not.

Lemma 3.11 *Let $P \in \overline{\mathbf{TCSP}}$, and let $e = P_0 \xmapsto{z_1} P_1 \xmapsto{z_2} \ldots \xmapsto{z_n} P_n \in$ $\mathsf{exec}_{TF}(P)$. For any $0 \leqslant \varepsilon \leqslant 1$, there exists an execution $[e]_\varepsilon = P'_0 \xmapsto{z'_1} P'_1 \xmapsto{z'_2} \ldots \xmapsto{z'_n} P'_n \in \mathsf{exec}_{TF}(P)$ with the following properties:*

1. *The transitions and evolutions of e and $[e]_\varepsilon$ are in natural one-to-one correspondence. More precisely, whenever $P_i \xmapsto{z_{i+1}} P_{i+1}$ in e is a transition, then so is $P'_i \xmapsto{z'_{i+1}} P'_{i+1}$ in $[e]_\varepsilon$, and moreover $z'_{i+1} = z_{i+1}$. On the other hand, whenever $P_i \xmapsto{z_{i+1}} P_{i+1}$ in e is an evolution, then so is $P'_i \xmapsto{z'_{i+1}} P'_{i+1}$ in $[e]_\varepsilon$, with $|z_{i+1} - z'_{i+1}| < 1$.*
2. *All evolutions in $[e]_\varepsilon$ have integral duration.*
3. *$P'_0 \equiv P_0 \equiv P$; in addition, $P'_i \in \overline{\mathbf{TCSP}}$ and $\mathsf{init}_{TF}(P'_i) = \mathsf{init}_{TF}(P_i)$ for all $0 \leqslant i \leqslant n$.*

4. *For any prefix* $e(k) = P_0 \xmapsto{z_1} P_1 \xmapsto{z_2} \ldots \xmapsto{z_k} P_k$ *of* e*, the corresponding prefix* $[e]_\varepsilon(k)$ *of* $[e]_\varepsilon$ *is such that* $\mathsf{dur}([e]_\varepsilon(k)) = [\mathsf{dur}(e(k))]_\varepsilon$*.*

Executions all of whose evolutions have integral duration are called *integral* executions. The integral execution $[e]_\varepsilon$ as constructed above is called the *ε-digitisation* of e.[3] The special case $\varepsilon = 1$ is particularly important: each transition in $[e]_1$ happens at the greatest integer time less than or equal to the time of occurrence of its *vis-à-vis* in e; for this reason $[e]_1$ is termed the *lower digitisation* of e.

Proof (Sketch.) The proof proceeds by structural induction over Timed CSP syntax. Among the tools it introduces and makes substantial use of figures the notion of *indexed bisimulation*. It is interesting to note that the crucial property of P required in the proof is the fact that all delays in P are integral; well-timedness is irrelevant. Details can be found in Appendix A. ■

3.4 Refinement, Specification, Verification

An important partial order can be defined on $\mathcal{P}(TF)$, as follows:

Definition 3.3 *For* $P, Q \subseteq TF$ *(and in particular for* $P, Q \in \mathcal{M}_{TF}$*), we let* $P \sqsubseteq_{TF} Q$ *if* $P \supseteq Q$*. For* $P, Q \in \overline{\mathbf{TCSP}}$*, we write* $P \sqsubseteq_{TF} Q$ *to mean* $\mathcal{F}_T[\![P]\!] \sqsubseteq_{TF} \mathcal{F}_T[\![Q]\!]$*, and* $P =_{TF} Q$ *to mean* $\mathcal{F}_T[\![P]\!] = \mathcal{F}_T[\![Q]\!]$*.*

We may drop the subscripts and write simply $P \sqsubseteq Q$, $P = Q$ whenever the context is clear.

This order, known as *timed failure refinement*, has the following central property (as can be verified by inspection of the relevant definitions). Here P and Q are processes:

$$P \sqsubseteq Q \Leftrightarrow P \sqcap Q = P. \qquad (3.38)$$

For this reason, $\sqsubseteq$ is also referred to as the *order of nondeterminism*—$P \sqsubseteq Q$ if and only if P is 'less deterministic' than Q, or in other words if and only if Q's behaviour is more predictable (in a given environment) than P's.

[3]Although $[e]_\varepsilon$ is *not* necessarily unique for a given execution e, we consider any two such executions to be interchangeable for our purposes.

Similar refinement orders (all obeying Equation (3.38) above) can be defined in most other (timed and untimed) CSP models. These orders are often of significantly greater importance in untimed models as they constitute the basis for the computation of fixed points (whereas timed models are usually predicated upon ultrametric spaces which rely on a different fixed-point theory). Refinement orders also play a central rôle as specification formalisms within untimed or discrete-time CSP models, whereas they can prove problematic for that purpose in dense timed models (as we demonstrate below). Nonetheless, because this work specifically studies the relationship between discrete and continuous modelling paradigms for timed systems, it is imperative to include refinement in our investigations. In addition, since $P = Q \Leftrightarrow (P \sqsubseteq Q \wedge Q \sqsubseteq P)$, a decision procedure for refinement yields a decision procedure for process equivalence, a central and perennial problem in Computer Science. (Incidentally, the converse—deciding refinement from process equivalence—follows from Equation (3.38).)

For $P \subseteq TF$, let $\mathsf{TTraces}(P)$ be the set of timed traces of P. Using this, we define a second notion of refinement—*timed trace refinement*—between sets of timed failures (and in particular $\mathcal{M}_{TF}$ processes):

Definition 3.4 *For any $P, Q \subseteq TF$, we let $P \sqsubseteq_{TT} Q$ if $\mathsf{TTraces}(P) \supseteq \mathsf{TTraces}(Q)$.*

We overload the notation and extend $\sqsubseteq_{TT}$ to $\overline{\mathbf{TCSP}}$ programs in the obvious way.

A major aim of the field of process algebra is to model systems so as to be able to formulate and establish assertions concerning them. Such assertions are usually termed *specifications*. Depending upon the model under consideration, specifications can take wide-ranging forms. Our principal interest in this thesis concerns denotational models, and accordingly we will deal exclusively with specifications expressed in terms of these models.

In general, a specification $S = S(P)$ on processes is simply a predicate on P; for example it could be $S(P)$: 'P cannot perform any events'. This can be expressed in English (as we have done here), mathematically ($(s, \aleph) \in P \Rightarrow s = \langle\rangle$), refinement-theoretically ($STOP \sqsubseteq_{TT} P$ or $STOP \sqsubseteq_{TF} P$), or using some other formalism such as temporal logic[4]. All these formulations are easily seen to be equivalent over $\mathcal{M}_{TF}$, and in this work we shall not

[4]An excellent account of the use of temporal logic(s) as a specification formalism in both Timed and untimed CSP can be found in [Jac92].

be overly concerned with the particular formalism chosen. We do remark, however, that expressing specifications in terms of refinement can lead to problematic situations. For instance, one would naturally want to express the specification $S(P)$: 'P cannot perform the event a' as $RUN_{\Sigma^{\checkmark}-\{a\}} \sqsubseteq_{TT} P$, where $RUN_A = a : A \longrightarrow RUN_A$. The problem is that RUN_A is not a well-timed process, and even if, as we have seen, it can easily be modelled in an operational semantics, no-one has yet produced a consistent denotational model in which such Zeno processes could be interpreted.[5]

Note that even models allowing unbounded nondeterminism (see, e.g., [MRS95]) place restrictions disallowing, among others, an internal choice over the set of processes unable to perform an a. The attempt to express $S(P)$ as $\bigsqcap\{Q \in \overline{\mathbf{TCSP}} \mid \forall s \in \mathsf{TTraces}(\mathcal{F}_T[\![Q]\!]) \bullet a \notin \sigma(s)\} \sqsubseteq_{TF} P$ is therefore also doomed.

Thus while in practice one could conceivably still express, *for a given process* P, the desired specification as a refinement between P and a well-timed process, this example shows that one cannot rigorously do so on a general basis. We shall return to this question later on.

A process P meets, or *satisfies*, a specification S, if $S(P)$ is true; in that case we write $P \vDash S$.

Specifications fall naturally into certain categories. A *timed trace specification*, for example, is one that can be stated exclusively in terms of timed traces. We are particularly (though not exclusively) interested in a type of specifications known as *behavioural* specifications. A *behavioural* specification is one that is universally quantified over the behaviours of processes. In other words, $S = S(P)$ is a behavioural specification if there is a predicate $S'(s, \aleph)$ on timed failures such that, for any P,

$$P \vDash S \Leftrightarrow \forall (s, \aleph) \in P \bullet S'(s, \aleph).$$

In this case we may abuse notation and identify S with S'. Note that S' itself may be identified with a subset of TF, namely the set of $(s, \aleph) \in TF$ such that $S'(s, \aleph)$.

A *safety property* is a requirement that 'nothing bad happen'. For us this will translate as a behavioural timed trace specification: certain timed traces are prohibited.[6] A *liveness property* is one that says 'something good

[5]Aside from finite variability, the dual requirement that refusals should consist in *finite* unions of left-closed, right-open intervals makes embedding $\mathcal{M}_{TF}$ into a *domain* a challenging problem. Contrast this with our discussion on the same topic in Section 5.3.

[6]It can be argued that certain specifications, which cannot be expressed entirely in terms of timed traces, are in fact safety properties, but we will for simplicity nonetheless

is not prevented from happening'. Thus a liveness property in general simply corresponds to a behavioural timed failure specification, although in practice we expect such a specification to primarily concern refusals.

Any specification $S = S(P)$ which can be expressed, for a fixed process Q, as $Q \sqsubseteq P$, is automatically behavioural. (Note that the reverse refinement, $P \sqsubseteq Q$, is not.) Thus behavioural specifications are identified with *requirements*: the implementation (P) must have all the safety and liveness properties of the requirement (Q).

The discussion in the remainder of this section concerns behavioural specifications exclusively.

A number of techniques have been devised to help decide when a given Timed CSP process meets a particular specification. Schneider [Sch89] and Davies [Dav91] have produced *complete proof systems*, sets of rules enabling one to derive specification satisfaction, for various models of Timed CSP. A case study illustrating the use of such proof systems is presented in [Sch94]. These techniques, along with an impressive array of related methodologies, however require significant prior insight before they can be reasonably applied to particular problems, and do not appear likely to be mechanisable in their present form.

Another technique is that of *timewise refinement*, introduced by Reed in [Ree88, Ree89] and developed by Schneider in [Sch89, RRS91, Sch97]. It can sometimes be used to establish certain untimed properties of timed processes, by removing all timing information from them, and verifying that the corresponding (untimed) CSP processes exhibits the properties in question. Simple criteria exist to decide when this technique can be soundly applied. It is clearly mechanisable (since the verification of (untimed) CSP processes itself is), but suffers from obvious restrictions in its applicability. Nonetheless, it can prove enormously useful in those cases where it can be employed.

A third approach was taken by Jackson in [Jac92], in which he develops full-fledged temporal-logic-based specification languages, and, invoking the seminal *region graphs* methods of Alur, Courcoubetis, and Dill [ACD90, ACD93], shows how a restricted subset of Timed CSP yields processes for which the verification of certain temporal logic specifications can always *a priori* be model checked. His restrictions on Timed CSP ensure that processes remain, in a certain sense, finite state. He then translates such processes into *timed graphs*, and constructs an equivalence relation which identifies states that essentially cannot be distinguished by the clocks at hand. This yields a

stick with the proposed terminology in this work.

finite quotient space which can then be mechanically explored. Some current disadvantages of this technique are the sharp syntactic restrictions it imposes on Timed CSP, as well as the constraints on the sort of specifications which are allowable (excluding, for instance, refinement checks). It should be added that the complexity of the resulting model checking algorithm is quite high; we shall return to this point in Chapter 9.

Chapter 4

Discrete-Time Modelling: Motivation

In this chapter, we aim to provide the intuition behind the constructions of the discrete-time models presented in Chapters 5 and 7. We will look at each of the Timed CSP operators in turn, and discuss how best to interpret them within a CSP-based discrete-time framework. We assume some familiarity with the standard CSP semantics ([Ros97, Sch00] are two good references), which we will invoke throughout; however, the rest of this thesis is self-contained (with the exception of sections 5.4 and 6.7), so that this chapter may be skipped with no significant loss of continuity.

We will define a 'translation' function Ψ converting Timed CSP syntax into CSP syntax, in such a way that the behaviours of $\Psi(P)$, interpreted in a CSP framework, approximate as closely as possible those of the $\overline{\mathbf{TCSP}}$ program P, interpreted in the timed failures model. This translation, in other words, should preserve as much timing information as possible. (We will not later explicitly require Ψ, nor any other of the constructs introduced in this chapter, except in sections 5.4 and 6.7, to describe how $\overline{\mathbf{TCSP}}$ programs can be model checked on the automated tool FDR.)

We define Ψ inductively over Timed CSP syntax in the following few sections.

4.1 Event Prefixing

Consider the program $Q = a \longrightarrow P$. Interpreted in $\mathcal{M}_{TF}$, this process is initially prepared to let time pass at will; there is no requirement for a to occur within any time period. In standard CSP models, however, such a process is of course incapable of initially communicating any *tock*s, which we would interpret as 'forcing' a to occur within at most one time unit.

An adequate translation of Q, therefore, has to ensure the unhindered passage of time, or in other words that *tock* events are always permissible. The desired new program, then, should be written: $Q' = (a \longrightarrow P) \;\Box\; (\mathit{tock} \longrightarrow Q')$. We abbreviate this construct as $Q' = a \longrightarrow_{\mathrm{t}} P$ (where the subscript 't' stands for *tock*). In other words,

$$a \longrightarrow_{\mathrm{t}} P \quad \widehat{=} \quad \mu X \bullet ((a \longrightarrow P) \;\Box\; (\mathit{tock} \longrightarrow X)).$$

Of course, P must also be suitably translated as the process makes progress. We thus set, in general

$$\Psi(a \longrightarrow P) \quad \widehat{=} \quad a \longrightarrow_{\mathrm{t}} \Psi(P).$$

Extending this to the case of the general prefix operator yields

$$\Psi(a : A \longrightarrow P(a)) \quad \widehat{=} \quad a : A \longrightarrow_{\mathrm{t}} \Psi(P(a)).$$

4.2 Deadlock, Termination, Delay

Naturally, the treatment of *STOP* must follow a similar path: its interpretation in $\mathcal{M}_{TF}$ allows time to pass at will. Hence

$$\Psi(\mathit{STOP}) \quad \widehat{=} \quad \mu X \bullet \mathit{tock} \longrightarrow X \quad \widehat{=} \quad \mathit{STOP}_{\mathrm{t}}.$$

It should be equally clear how to handle *SKIP*:

$$\Psi(\mathit{SKIP}) \quad \widehat{=} \quad \checkmark \longrightarrow_{\mathrm{t}} \mathit{STOP}_{\mathrm{t}} \quad \widehat{=} \quad \mathit{SKIP}_{\mathrm{t}}.$$

As for *WAIT n*, a little thought reveals that

$$\Psi(\mathit{WAIT}\ n) \quad \widehat{=} \quad \overbrace{\mathit{tock} \longrightarrow \ldots \longrightarrow \mathit{tock} \longrightarrow}^{n\ \mathit{tock}\text{s}} \mathit{SKIP}_{\mathrm{t}} \quad \widehat{=} \quad \mathit{WAIT}_{\mathrm{t}}\ n$$

is the only reasonable proposal. Note, importantly, that this definition uses the $\longrightarrow$ operator (rather than $\longrightarrow_{\mathrm{t}}$), so as to guarantee that a $\checkmark$ be on offer after *exactly n tock*s.

4.3 External Choice

The program $Q = P_1 \Box P_2$ poses some slightly more intricate problems. Over $\mathcal{M}_{TF}$, Q will wait however long it takes for either P_1 or P_2 to communicate a visible event, at which point the choice will be resolved in favour of that process. Unfortunately, if *tock*s are interpreted as regular events, the choice will *ipso facto* be made within at most one time unit under CSP semantics.

The solution is to postulate a new operator $\Box_{\mathrm{t}}$ that behaves like $\Box$ in all respects, except that it lets *tock*s 'seep through' it without forcing the choice to be resolved. Here we are assuming, in addition, that P_1 and P_2 synchronise on every *tock* communication, for reasons discussed in the next section.

Direct operational and denotational definitions of $\Box_{\mathrm{t}}$ can be given, but the following construct (due to Steve Schneider) shows that $\Box_{\mathrm{t}}$ can in fact be expressed in terms of standard CSP operators, if we assume that P_1 and P_2 can never refuse *tock*:

First let $\Sigma_1 = \{1.a \mid a \in \Sigma\}$ and $\Sigma_2 = \{2.a \mid a \in \Sigma\}$. Next, define two functions $f_1, f_2 : \Sigma_{tock} \cup \Sigma_1 \cup \Sigma_2 \longrightarrow \Sigma_{tock} \cup \Sigma_1 \cup \Sigma_2$ such that

$$\begin{aligned} f_i(a) &= i.a \quad \text{if } a \in \Sigma \\ &= a \quad\;\; \text{otherwise.} \end{aligned}$$

Finally, we have

$$P_1 \Box_{\mathrm{t}} P_2 = f_1^{-1}(f_2^{-1}((f_1(P_1) \underset{\{tock\}}{\|} f_2(P_2)) \underset{\Sigma_1 \cup \Sigma_2}{\|} (RUN_{\Sigma_1} \Box RUN_{\Sigma_2})))$$

where $RUN_A = a : A \longrightarrow RUN_A$.

Naturally, we set

$$\Psi(P_1 \Box P_2) \;\widehat{=}\; \Psi(P_1) \Box_{\mathrm{t}} \Psi(P_2).$$

Note, however, that in most cases a much simpler translation can be obtained: $(a \longrightarrow_{\mathrm{t}} P_1) \Box_{\mathrm{t}} (b \longrightarrow_{\mathrm{t}} P_2)$, for instance, is equivalent to $Q = (tock \longrightarrow Q) \Box (a \longrightarrow P_1) \Box (b \longrightarrow P_2)$ in CSP models.

4.4 Concurrency

The main point concerning the parallel operators $\underset{B}{\|}$ and $|||$ is that they should ensure a *uniform* rate of passage of time: no process in a parallel composition should be allowed to run 'faster' than another. This is achieved, naturally, by forcing processes to synchronise on *tock*. Thus

$$\Psi(P_1 \underset{B}{\|} P_2) \quad \widehat{=} \quad \Psi(P_1) \underset{B\cup\{tock\}}{\|} \Psi(P_2) \quad \widehat{=} \quad \Psi(P_1) \underset{B}{\|_{\mathrm{t}}} \Psi(P_2).$$

Since interleaving normally corresponds to parallel composition over an empty interface, we set

$$\Psi(P_1 \,|||\, P_2) \quad \widehat{=} \quad \Psi(P_1) \underset{\{tock\}}{\|} \Psi(P_2) \quad \widehat{=} \quad \Psi(P) \,|||_{\mathrm{t}}\, \Psi(Q).$$

4.5 Hiding and Sequential Composition

An adequate discrete-time treatment of hiding and sequential composition pits us against greater difficulties than the other operators. This is a direct consequence of the assumption of maximal progress, or τ-urgency: hidden events must happen as quickly as possible.[1]

As an illustration, consider the program

$$P = ((a \longrightarrow STOP) \;\square\; WAIT\ 1)\ ;\ b \longrightarrow STOP.$$

Interpreted over $\mathcal{M}_{TF}$, P initially offers the event a for exactly one time unit, after which (if a has not occurred) the offer is instantly withdrawn and an open-ended offer of b is made. (This is one of the simplest possible examples of a timeout.)

Unfortunately, the behaviours of

$$P' = ((a \longrightarrow_{\mathrm{t}} STOP_{\mathrm{t}}) \;\square_{\mathrm{t}}\; (tock \longrightarrow SKIP_{\mathrm{t}}))\ ;\ b \longrightarrow_{\mathrm{t}} STOP_{\mathrm{t}},$$

in CSP models, are easily seen to allow an a to be communicated after an arbitrary number of *tock*s; this is because the hidden $\checkmark$, which $SKIP_{\mathrm{t}}$ can potentially communicate, is not made *urgent* by sequential composition, contrary to the situation with P in $\mathcal{M}_{TF}$. In other words, P' is not an adequate translation of P.

In order to faithfully capture, in a discrete-time setting, the timed behaviours of hiding and sequential composition, it is necessary to postulate

[1]The reasons for requiring the maximal progress assumption (in the absence of which it is impossible, for instance, to implement proper timeouts) are well-known and discussed in most expository texts on timed systems—see, for instance, [Sch00].

the urgency of hidden events. By this we mean that the event *tock* should not be allowable so long as a silent (τ) event is on offer.

Although it is easy to capture this requirement operationally (as we shall see in Chapter 5), it is impossible to render it in either the traces or the failures models, which are the main denotational models for CSP. (Traces are sequences of events that a process can be observed to communicate, whereas a failure is trace augmented by a refusal at the end of the trace: a record of events that the process cannot perform after the given trace.) The following example (originally due to Langerak [Lan90], and reproduced in [Sch00]) shows why failures are unable support τ-urgency in a compositional way.

Consider the programs P and Q, defined in terms of *TEA* and *COFFEE*:

$$\begin{aligned}
TEA &= tea \longrightarrow_{\mathrm{t}} STOP_{\mathrm{t}} \\
COFFEE &= coffee \longrightarrow_{\mathrm{t}} STOP_{\mathrm{t}} \\
P &= TEA \sqcap COFFEE \\
Q &= ((coffee \longrightarrow STOP_{\mathrm{t}}) \,\square\, (tock \longrightarrow TEA)) \sqcap \\
& ((tea \longrightarrow STOP_{\mathrm{t}}) \,\square\, (tock \longrightarrow COFFEE)).
\end{aligned}$$

P and Q have the same set of failures: they can both perform any number of *tock*s followed by a *tea* or a *coffee* (followed by further *tock*s); and, at any stage before a drink is delivered, either *tea* or *coffee*, but not both, can be refused.

However, there is a difference in their behaviour: P is always committed to the drink it first chooses to offer, whereas after one *tock* Q switches the drink it offers. Failure information is not detailed enough to identify this difference.

Nevertheless, this difference in behaviour means that, under τ-urgency, $P \setminus tea$ can provide a *coffee* after a *tock*, whereas $Q \setminus tea$ cannot—the hidden *tea* will occur either immediately or after one *tock*, in both cases preventing the possibility of *coffee* after *tock*. Hence we conclude that, under τ-urgency, the failures alone of a process P are not sufficient to determine even the traces of $P \setminus A$, let alone the failures.

The question then arises as how to capture τ-urgency denotationally. It turns out that, although τ events are introduced by other operators besides hiding and sequential composition, it is sufficient to require urgent behaviour of hiding and sequential composition alone to ensure τ-urgency in general.

Recall that behaviours of $P \setminus A$ are derived from those of P. To guarantee urgency, it is sufficient, in calculating the behaviours of $P \setminus A$, to dismiss any behaviour of P in which a *tock* is recorded while events from the set A were on offer. A similar proposal handles sequential composition.

In order to know whether events in A were refusable prior to an occurrence of *tock* or not, one must record refusal information *throughout* a trace, rather than exclusively at the end of it. This type of modelling, known as *refusal testing*, was introduced by Phillips [Phi87] and served as the basis for the refusal testing models of Mukarram [Muk93]. The denotational models we present in Chapter 5 and 7 are thus also refusal testing based.

As a reminder (especially when using an automated model checker) that τ-urgency is required, we will use the symbols $\setminus_{\mathrm{t}}$ and $;_{\mathrm{t}}$ to represent respectively urgent hiding and urgent sequential composition within CSP. Thus we set

$$\begin{array}{rcl} \Psi(P \setminus A) & \widehat{=} & \Psi(P) \setminus_{\mathrm{t}} A \\ \Psi(P_1 \, ; P_2) & \widehat{=} & \Psi(P_1) \;;_{\mathrm{t}}\; \Psi(P_2). \end{array}$$

4.6 Timeout

The $\mathcal{M}_{TF}$ behaviour of the timeout operator exhibits characteristics of both external choice and sequential composition; the problems it presents can therefore be tackled using techniques similar to those shown above. In practice, assuming that the event *trig* does not figure in the alphabet of either P_1 or P_2, one has the identity $P_1 \overset{n}{\rhd} P_2 = (P_1 \,\Box\, (\mathit{WAIT}\ n \,;\, \mathit{trig} \longrightarrow P_2)) \setminus \mathit{trig}$, and thus

$$\begin{array}{rcl} \Psi(P_1 \overset{n}{\rhd} P_2) & \widehat{=} & (\Psi(P_1) \,\Box_{\mathrm{t}}\, (\mathit{WAIT}_{\mathrm{t}}\ n \;;_{\mathrm{t}}\; \mathit{trig} \longrightarrow_{\mathrm{t}} \Psi(P_2))) \setminus_{\mathrm{t}} \mathit{trig} \\ & \widehat{=} & \Psi(P_1) \overset{n}{\rhd}_{\mathrm{t}} \Psi(P_2). \end{array}$$

4.7 Others

The remaining operators, namely internal choice, renaming, and recursion, pose no difficulties: each is left unchanged. Naturally, the same goes for program variables. We recapitulate the full definition of Ψ after listing the new constructs introduced in this chapter.

4.8 Summary

Definition 4.1 *We have defined the following syntactic operators.*

$$\begin{array}{rcl}
a \longrightarrow_{\mathrm{t}} P & \mathrel{\widehat=} & \mu X \bullet ((a \longrightarrow P) \mathrel{\Box} (tock \longrightarrow X)) \\
a : A \longrightarrow_{\mathrm{t}} P(a) & \mathrel{\widehat=} & \mu X \bullet ((a : A \longrightarrow P(a)) \mathrel{\Box} (tock \longrightarrow X)) \\
STOP_{\mathrm{t}} & \mathrel{\widehat=} & \mu X \bullet tock \longrightarrow X \\
SKIP_{\mathrm{t}} & \mathrel{\widehat=} & \checkmark \longrightarrow_{\mathrm{t}} STOP_{\mathrm{t}} \\
WAIT_{\mathrm{t}}\ n & \mathrel{\widehat=} & \overbrace{tock \longrightarrow \ldots \longrightarrow tock \longrightarrow}^{n\ tocks} SKIP_{\mathrm{t}} \\
P_1 \mathrel{\Box_{\mathrm{t}}} P_2 & \text{—} & \text{[Cf. Section 4.3]} \\
P \mathbin{\underset{B}{\|}_{\mathrm{t}}} Q & \mathrel{\widehat=} & P \mathbin{\underset{B \cup \{tock\}}{\|}} Q \\
P_1 \mathbin{|||_{\mathrm{t}}} P_2 & \mathrel{\widehat=} & P_1 \mathbin{\underset{\{tock\}}{\|}} P_2 \\
P \setminus_{\mathrm{t}} A,\ P_1 \mathbin{;_{\mathrm{t}}} P_2 & \text{—} & \text{[Cf. Section 4.5]} \\
P_1 \overset{n}{\rhd}_{\mathrm{t}} P_2 & \mathrel{\widehat=} & (P_1 \mathrel{\Box_{\mathrm{t}}} (WAIT_{\mathrm{t}}\ n \mathbin{;_{\mathrm{t}}} trig \longrightarrow_{\mathrm{t}} P_2)) \setminus_{\mathrm{t}} trig.
\end{array}$$

Definition 4.2 *The syntactic function* Ψ *is defined inductively as follows.*

$$\begin{array}{rcl}
\Psi(STOP) & \mathrel{\widehat=} & STOP_{\mathrm{t}} \\
\Psi(SKIP) & \mathrel{\widehat=} & SKIP_{\mathrm{t}} \\
\Psi(WAIT\ n) & \mathrel{\widehat=} & WAIT_{\mathrm{t}}\ n \\
\Psi(P_1 \overset{n}{\rhd} P_2) & \mathrel{\widehat=} & \Psi(P_1) \overset{n}{\rhd}_{\mathrm{t}} \Psi(P_2) \\
\Psi(a \longrightarrow P) & \mathrel{\widehat=} & a \longrightarrow_{\mathrm{t}} \Psi(P) \\
\Psi(a : A \longrightarrow P(a)) & \mathrel{\widehat=} & a : A \longrightarrow_{\mathrm{t}} \Psi(P(a)) \\
\Psi(P_1 \mathrel{\Box} P_2) & \mathrel{\widehat=} & \Psi(P_1) \mathrel{\Box_{\mathrm{t}}} \Psi(P_2) \\
\Psi(P_1 \sqcap P_2) & \mathrel{\widehat=} & \Psi(P_1) \sqcap \Psi(P_2) \\
\Psi(P_1 \mathbin{\underset{B}{\|}} P_2) & \mathrel{\widehat=} & \Psi(P_1) \mathbin{\underset{B}{\|}_{\mathrm{t}}} \Psi(P_2) \\
\Psi(P_1 \mathbin{|||} P_2) & \mathrel{\widehat=} & \Psi(P_1) \mathbin{|||_{\mathrm{t}}} \Psi(P_2) \\
\Psi(P_1 \mathbin{;} P_2) & \mathrel{\widehat=} & \Psi(P_1) \mathbin{;_{\mathrm{t}}} \Psi(P_2) \\
\Psi(P \setminus A) & \mathrel{\widehat=} & \Psi(P) \setminus_{\mathrm{t}} A \\
\Psi(f^{-1}(P)) & \mathrel{\widehat=} & f^{-1}(\Psi(P)) \\
\Psi(f(P)) & \mathrel{\widehat=} & f(\Psi(P)) \\
\Psi(X) & \mathrel{\widehat=} & X \\
\Psi(\mu X \bullet P) & \mathrel{\widehat=} & \mu X \bullet \Psi(P).
\end{array}$$

We point out once more that we have defined Ψ mainly as an aid to illustrate the issues at hand, and will not make any formal use of this function in the remainder of this work, other than as a tool to encode programs into FDR in Sections 5.4 and 6.7.

We have shown in this chapter that a number of points need to be addressed when attempting to build discrete-time operational and denotational semantics for $\overline{\textbf{TCSP}}$ programs. They are:

- The unhindered passage of time;
- The temporal soundness of external choice;
- The uniform rate of passage of time; and
- The assumption of maximal progress, or τ-urgency.

Chapter 5

The Discrete-Time Refusal Testing Model

We present a discrete-time denotational model for Timed CSP, $\mathcal{M}_R$, together with a congruent operational semantics; both of these are constructed in accordance with the remarks made in the previous chapter.

The use of the distinguished *tock* event to discretely model time in CSP was first proposed by Roscoe, and an excellent account of how this technique is applied within standard CSP models is given in [Ros97]. A refusal testing model for CSP was developed by Mukarram in [Muk93], drawing on the work of Phillips [Phi87]. The model we present here builds upon both these approaches. We also use ultrametric spaces to compute fixed points, a technique which has long been known and figures prominently, among others, in Reed and Roscoe's treatment of continuous Timed CSP [RR86, RR87, Ree88, RR99]. This technique, of course, was instrumental in the construction of $\mathcal{M}_{TF}$.

As seen in Chapter 3, $\mathcal{M}_{TF}$ has also been endowed with a congruent operational semantics by Schneider [Sch95]. Likewise, we present a congruent operational semantics of a similar flavour, thus offering denotational and operational models for Timed CSP which approximate as closely as possible the continuous-time models described in Chapter 3.

For obvious reasons, notation used in the present chapter overloads that of Chapter 3. We trust that the gains in simplicity outweigh the potential for confusion, which the context should help prevent.

5.1 Denotational Semantics

We define the *discrete-time refusal testing model* $\mathcal{M}_R$ and the associated semantic mapping $\mathcal{R}[\![\cdot]\!] : \overline{\mathbf{TCSP}} \longrightarrow \mathcal{M}_R$.

As with most CSP-based denotational models, the basic philosophy underlying $\mathcal{M}_R$ is one of *experimentation* and *observation*. We think of a process as a 'black box', equipped with a number of labelled buttons, together with a red light and a buzzer. The red light, when lit, indicates internal activity. If no notice is (was) taken of the light then one must assume that it is (was) lit; only when the light is actually seen turned off can one conclude that the process has reached stability.[1] The light is guaranteed to go off shortly after the process reaches stability, although not necessarily at the same instant.

Pressing buttons corresponds to experimenting with the process. If the button goes down, the corresponding event is deemed accepted, and a record is made to that effect. If the button does not go down, then two cases arise: if the red light is on, the process is deemed (potentially) unstable, and no firm conclusion as to the rejection of the corresponding event can be reached; on the other hand, if the red light is off, then we know that the process is in a stable state—the corresponding event will therefore forever be refused (at least until further buttons are pushed or the buzzer goes off), and a record of refusal of the event in question is made. This applies, naturally, to individual buttons as well as sets of such, although we insist that the black box only allow at most one button to be down at any one time.

The buzzer, finally, can go off at any time, and in fact does so with regularity. We record this occurrence, as explained earlier, by the event *tock*—this represents the passage of one time unit. Since we have no control over the buzzer, *tock*s never appear in refusals. We assume furthermore that the buzzer never goes off at exactly the same time as some button is being depressed.

We will see that several other assumptions, such as the impossibility of infinitely many events taking place in a finite amount of time, follow from our definitions; they are therefore listed explicitly as axioms of the denotational model.

The recorded observations thus consist of alternating sequences of refusals and events, with the understanding that no refusal information be recorded

[1] *Stability* here refers to the presence of hidden events on offer—it is related, but not identical, to Reed and Roscoe's notion of stability as invariance under the passage of time.

unless the red light is seen turned off. A logical consequence of this philosophy is that observations should be *downward-closed*: any observation could instead have given rise to another observation containing 'less' information.

We begin with some notation.[2] An *event* is an element of $\Sigma^{\checkmark}_{tock}$. A *refusal* is either a subset of $\Sigma^{\checkmark}$ or the *null refusal* $\bullet$; the set $\{\bullet\} \cup \mathcal{P}(\Sigma^{\checkmark})$ is denoted REF. A *test* is an alternating finite sequence of refusals and events, of length at least 1, beginning and ending with a refusal. In other words, tests are generated by the grammar $T := \langle A \rangle \mid T^\frown\langle a, A\rangle$, where a and A respectively stand for events and refusals. The set of all tests is denoted $TEST$.

If $u = \langle A_0, a_1, A_1, \ldots, a_k, A_k\rangle \in TEST$, we let $\mathsf{trace}(u) \mathrel{\widehat{=}} \langle a_1, a_2, \ldots, a_k\rangle$ denote the test u stripped of its refusals.

We define the following $\bullet$-friendly extensions of standard set-theoretic operations: $\in^*$, $\subseteq^*$, $\cup^*$, and $\cap^*$. The underlying principle is that these operators essentially treat $\bullet$ like an empty set, while retaining the usual properties of their un-starred counterparts. Rather than give comprehensive definitions, we list the salient $\bullet$-properties enjoyed by these operators; in what follows a is an event, and A is a non-$\bullet$ refusal. $a \notin^* \bullet$, $\bullet \subseteq^* \bullet \subseteq^* A$, $A \not\subseteq^* \bullet$, $\bullet \cup^* \bullet = \bullet$, $\bullet \cup^* A = A$, and $\bullet \cap^* \bullet = \bullet \cap^* A = \bullet$.

We also define an 'absolute set-value' on refusals: let $|\bullet| \mathrel{\widehat{=}} \emptyset$, and $|A| \mathrel{\widehat{=}} A$ if $\bullet \neq A \in REF$.

Let $u = \langle A_0, a_1, A_1, \ldots, a_k, A_k\rangle$ and $v = \langle B_0, b_1, B_1, \ldots, b_{k'}, B_{k'}\rangle$ be tests. We define the *information ordering* $\prec$ as follows: $u \prec v$ if $0 \leqslant k \leqslant k'$ and $A_0 \subseteq^* B_0$ and $\forall (1 \leqslant i \leqslant k) \centerdot a_i = b_i \wedge A_i \subseteq^* B_i$. Note that this makes $\prec$ a partial order on $TEST$.

Lastly, recall from Section 2.3 the definitions of $\check{u}$ and $\hat{u}$, which remove respectively the first and last elements of the test u (viewed as a sequence).

Definition 5.1 *The* discrete-time refusal testing model $\mathcal{M}_R$ *is the set of all $P \subseteq TEST$ satisfying the following axioms, where $u, v \in TEST$, $A \in REF$, and $a \in \Sigma^{\checkmark}_{tock}$.*

[2]Some of the definitions presented here will be slightly altered in Chapter 7; the context should however always make clear what application we have in mind.

R1 $\langle \bullet \rangle \in P$

R2 $(u \in P \wedge v \prec u) \Rightarrow v \in P$

R3 $\hat{u}^\frown \langle \bullet \rangle \in P \Rightarrow \hat{u}^\frown \langle \emptyset \rangle \in P$

R4 $(A \neq \bullet \wedge \hat{u}^\frown \langle A \rangle^\frown \check{v} \in P \wedge \hat{u}^\frown \langle A, a, \bullet \rangle \notin P) \Rightarrow$
$\hat{u}^\frown \langle A \cup \{a\} \rangle^\frown \check{v} \in P$

R5 $(A \neq \bullet \wedge \hat{u}^\frown \langle A, a, \bullet \rangle \in P) \Rightarrow \hat{u}^\frown \langle A, tock, \bullet, a, \bullet \rangle \in P$

R6 $\hat{u}^\frown \langle A, a \rangle^\frown v \in P \Rightarrow a \notin^* A$

R7 $\forall\, k \in \mathbb{N} \,\textbf{.}\, \exists\, n \in \mathbb{N} \,\textbf{.}\, (u \in P \wedge \sharp(\mathsf{trace}(u) \upharpoonright tock) \leqslant k) \Rightarrow \sharp\mathsf{trace}(u) \leqslant n$

R8 $u^\frown \langle \checkmark \rangle^\frown v \in P \Rightarrow \mathsf{trace}(v) \leq \langle tock \rangle^\infty$.

We now give intuitive explanations of each of these axioms, and compare them to the axioms of the timed failures model $\mathcal{M}_{TF}$.

R1: A non-emptiness condition which states that the least we can observe of a process is that it has not stabilised (or at least that we have failed to record that it has) and that no events have been communicated. This corresponds to Axiom *TF1* of $\mathcal{M}_{TF}$.

R2: Postulates the downward-closed nature of observations. This corresponds to Axiom *TF2* of $\mathcal{M}_{TF}$.

R3: States that all processes eventually stabilise, and moreover, that it is always possible to reach stability before the next event (in particular *tock*) occurs. One interpretation of this axiom is that divergences are excluded.[3] This axiom has no direct counterpart in $\mathcal{M}_{TF}$, since dense modelling of time precludes having a notion of 'next instant'.

R4: Tells us how observations can be extended; paraphrasing the contrapositive, it informs us that if an event cannot be stably refused within a test, then it would have been possible to witness it. A subtle point is the assumption of stability ($A \neq \bullet$) at the junction in question, to cater for the possibility that the first event of the remainder ($\check{v}$) of the test may only be communicable while the process is unstable (the option to perform this event, and *ergo* the rest of $\check{v}$, being removed once the process stabilises). The matching axiom in $\mathcal{M}_{TF}$ is *TF3*.

R5: States that any event which can stably occur at the end of some test cannot be prohibited from occurring after a further *tock*. The justification for this axiom lies in the fact that cessation of availability results

[3]This statement, while correct, can be misleading in that certain models will treat diverging processes as being capable of *any* behaviour, including stabilising.

from an internal change of state, i.e., a silent action; instability thus ensues. Again, this axiom has no exact counterpart in $\mathcal{M}_{TF}$ since that model lacks a notion of 'next instant'. Note nonetheless that, by *R6*, the refusal A appearing in Axiom *R5* cannot contain the event a. In this respect, $\mathcal{M}_{TF}$'s Axiom *TF3* mirrors *R5* in that an event which cannot be refused at a certain time has to be allowed to occur at some, possibly very early, strictly later time in the future.

R6: Enshrines the principle that a stable refusal is indeed stable: if an event is refused and the process has stabilised, this event will never be accepted later on (i.e., not until another event, possibly *tock*, is witnessed). Again, translating this axiom in $\mathcal{M}_{TF}$ would require a notion of next instant. This phenomenon is different from that of *global stability* discussed in Appendix C and Chapter 9. Notice, too, that in $\mathcal{M}_{TF}$ an event *may* both be refused and occur at the same time: consider the occurrence of a in $a \longrightarrow STOP$.

R7: Postulates that processes are *non-Zeno*; in other words, it is impossible for infinitely many events to occur in a finite amount of time.[4] The corresponding axiom in $\mathcal{M}_{TF}$ is *TF4*. This axiom is also known as *finite variability*.

R8: Stipulates that termination in effect ends the progress of a process: no event (apart from *tock*s, enabling the passage of time) can be witnessed after a $\checkmark$. This is $\mathcal{M}_{TF}$'s Axiom *TF5*.

We now show that, as in the case of the model $\mathcal{M}_{TF}$, processes are free of *timestops*, i.e., can always, at any time, run on forever only communicating *tock* events. It follows that *signals* cannot be defined in $\mathcal{M}_R$.

Proposition 5.1 *Let $P \in \mathcal{M}_R$ be a process, and let $w \in P$ be a test of P. For any $k \geqslant 0$, we have $w^\frown\langle tock, \emptyset\rangle^k \in P$.*

Proof We proceed by induction on k. Note that we can assume that the last refusal of w is non-$\bullet$, thanks to Axiom *R3* (we can always restore it to $\bullet$ later on, if need be, by invoking Axiom *R2*).

[4]Note, however, that no fixed bound is imposed on the number of non-*tock* events which can be performed between two *tock*s—achieving this would require some very severe syntactic restrictions, namely removing one of recursion, sequential composition, or the concurrency operators.

The base case $k = 0$ is, by our assumptions, trivial. For the inductive step $k+1$, let $u = w^\frown\langle tock, \emptyset\rangle^k$, and suppose that $u \in P$ but $u^\frown\langle tock, \emptyset\rangle \notin P$. Let A be the last refusal of u (the empty set except, possibly, if $k = 0$). In any case, we have $A \neq \bullet$. We can invoke Axiom *R4* (with a trivial $v = \langle\bullet\rangle$) to conclude that $\hat{u}^\frown\langle A \cup \{tock\}\rangle \in P$. This, of course, contradicts the definition of refusals, which are not allowed to contain the event *tock*. Consequently, we must have $u^\frown\langle tock, \emptyset\rangle = w^\frown\langle tock, \emptyset\rangle^{k+1} \in P$, as required. ■

In order to define the semantic mapping $\mathcal{R}[\![\cdot]\!]$, we need another set of definitions.

If $u = \langle A_0, a_1, A_1, \ldots, a_k, A_k\rangle$ is a test, we let $\mathsf{refusals}(u) \mathrel{\widehat{=}} \bigcup_{i=0}^{k}\{|A_i|\}$. Note that $\mathsf{refusals}(u)$ is always a set, namely the set of events that are explicitly refused, at one point or another, in u.

We also need the auxiliary function $(\cdot) \underset{B}{\|} (\cdot) : TEST \times TEST \longrightarrow \mathcal{P}(TEST)$. Here $B \subseteq \Sigma^{\checkmark}$, and $a, c \in \Sigma_{tock}$. Let us write $B_{\mathrm{t}} = B \cup \{tock\}$. We proceed inductively on both u and u' (in defining $u \underset{B}{\|} u'$).

$$
\begin{array}{rcll}
\langle A\rangle \underset{B}{\|} \langle C\rangle & \mathrel{\widehat{=}} & \{\langle D\rangle \mid D \subseteq^* (B \cap^* A) \cup^* & \\
 & & \quad (B \cap^* C) \cup^* (A \cap^* C)\} & \\
\langle A, a\rangle^\frown u \underset{B}{\|} \langle C\rangle & \mathrel{\widehat{=}} & \langle A\rangle \underset{B}{\|} \langle C\rangle & \text{if } a \in B_{\mathrm{t}} \\
 & \mathrel{\widehat{=}} & (\langle A\rangle \underset{B}{\|} \langle C\rangle)\{\langle a\rangle\}(u \underset{B}{\|} \langle C\rangle) & \text{if } a \notin B_{\mathrm{t}} \\
\langle A, \checkmark\rangle^\frown u \underset{B}{\|} \langle C\rangle & \mathrel{\widehat{=}} & \langle A\rangle \underset{B}{\|} \langle C\rangle & \text{if } \checkmark \in B_{\mathrm{t}} \\
 & \mathrel{\widehat{=}} & (\langle A\rangle \underset{B}{\|} \langle C\rangle)\{\langle\checkmark\rangle^\frown u\} & \text{if } \checkmark \notin B_{\mathrm{t}} \\
\langle A\rangle \underset{B}{\|} \langle C, c\rangle^\frown u & \mathrel{\widehat{=}} & \langle C, c\rangle^\frown u \underset{B}{\|} \langle A\rangle & \\
\langle A\rangle \underset{B}{\|} \langle C, \checkmark\rangle^\frown u & \mathrel{\widehat{=}} & \langle C, \checkmark\rangle^\frown u \underset{B}{\|} \langle A\rangle &
\end{array}
$$

$$
\begin{array}{rcll}
\langle A,a\rangle^\frown u_1 \underset{B}{\parallel} \langle C,c\rangle^\frown u_2 & \mathrel{\widehat{=}} & (\langle A\rangle \underset{B}{\parallel} \langle C\rangle)\{\langle a\rangle\}(u_1 \underset{B}{\parallel} u_2) & \text{if } a = c \in B_{\mathrm{t}} \\
& \mathrel{\widehat{=}} & \langle A\rangle \underset{B}{\parallel} \langle C\rangle & \text{if } a, c \in B_{\mathrm{t}} \wedge a \neq c \\
& \mathrel{\widehat{=}} & (\langle A\rangle \underset{B}{\parallel} \langle C\rangle)\{\langle c\rangle\} & \\
& & \quad (\langle A,a\rangle^\frown u_1 \underset{B}{\parallel} u_2) & \text{if } a \in B_{\mathrm{t}} \wedge c \notin B_{\mathrm{t}} \\
& \mathrel{\widehat{=}} & (\langle A\rangle \underset{B}{\parallel} \langle C\rangle)\{\langle a\rangle\} & \\
& & \quad (u_1 \underset{B}{\parallel} \langle C,c\rangle^\frown u_2) & \text{if } a \notin B_{\mathrm{t}} \wedge c \in B_{\mathrm{t}} \\
& \mathrel{\widehat{=}} & (\langle A\rangle \underset{B}{\parallel} \langle C\rangle)\{\langle a\rangle\} & \\
& & \quad (u_1 \underset{B}{\parallel} \langle C,c\rangle^\frown u_2) \cup & \\
& & (\langle A\rangle \underset{B}{\parallel} \langle C\rangle)\{\langle c\rangle\} & \\
& & \quad (\langle A,a\rangle^\frown u_1 \underset{B}{\parallel} u_2) & \text{if } a, c \notin B_{\mathrm{t}} \\
\langle A,\checkmark\rangle^\frown u_1 \underset{B}{\parallel} \langle C,\checkmark\rangle^\frown u_2 & \mathrel{\widehat{=}} & (\langle A\rangle \underset{B}{\parallel} \langle C\rangle)\{\langle \checkmark\rangle^\frown u_1\} & \\
\langle A,\checkmark\rangle^\frown u_1 \underset{B}{\parallel} \langle C,c\rangle^\frown u_2 & \mathrel{\widehat{=}} & (\langle A\rangle \underset{B}{\parallel} \langle C\rangle) & \text{if } \checkmark \in B_{\mathrm{t}} \wedge c \in B_{\mathrm{t}} \\
& \mathrel{\widehat{=}} & (\langle A\rangle \underset{B}{\parallel} \langle C\rangle)\{\langle c\rangle\} & \\
& & \quad (\langle A,\checkmark\rangle^\frown u_1 \underset{B}{\parallel} u_2) & \text{if } \checkmark \in B_{\mathrm{t}} \wedge c \notin B_{\mathrm{t}} \\
& \mathrel{\widehat{=}} & (\langle A\rangle \underset{B}{\parallel} \langle C\rangle)\{\langle \checkmark\rangle^\frown u_1\} & \text{if } \checkmark \notin B_{\mathrm{t}} \wedge c \in B_{\mathrm{t}} \\
& \mathrel{\widehat{=}} & (\langle A\rangle \underset{B}{\parallel} \langle C\rangle)\{\langle \checkmark\rangle^\frown u_1\} \cup & \\
& & (\langle A\rangle \underset{B}{\parallel} \langle C\rangle)\{\langle c\rangle\} & \\
& & \quad (\langle A,\checkmark\rangle^\frown u_1 \underset{B}{\parallel} u_2) & \text{if } \checkmark \notin B_{\mathrm{t}} \wedge c \notin B_{\mathrm{t}} \\
\langle A,a\rangle^\frown u_1 \underset{B}{\parallel} \langle C,\checkmark\rangle^\frown u_2 & \mathrel{\widehat{=}} & \langle C,\checkmark\rangle^\frown u_2 \underset{B}{\parallel} \langle A,a\rangle^\frown u_1. &
\end{array}
$$

Likewise, we define the auxiliary function $(\cdot) \mathbin{|||} (\cdot) : \mathit{TEST} \times \mathit{TEST} \longrightarrow \mathcal{P}(\mathit{TEST})$. Here $a, c \in \Sigma$. We proceed in a manner similar to the above:

$$
\begin{array}{rcl}
\langle A\rangle \mathbin{|||} \langle C\rangle & \mathrel{\widehat{=}} & \{\langle D\rangle \mid D \subseteq^* A \cap^* C\} \\
\langle A,a\rangle^\frown u \mathbin{|||} \langle C\rangle & \mathrel{\widehat{=}} & (\langle A\rangle \mathbin{|||} \langle C\rangle)\{\langle a\rangle\}(u \mathbin{|||} \langle C\rangle) \\
\langle A,\checkmark\rangle^\frown u \mathbin{|||} \langle C\rangle & \mathrel{\widehat{=}} & (\langle A\rangle \mathbin{|||} \langle C\rangle)\{\langle \checkmark\rangle^\frown u\} \\
\langle A,\mathit{tock}\rangle^\frown u \mathbin{|||} \langle C\rangle & \mathrel{\widehat{=}} & \langle A\rangle \mathbin{|||} \langle C\rangle
\end{array}
$$

$$\begin{array}{rcl}
\langle A\rangle \mathbin{|||} \langle C,c\rangle^\frown u & \mathrel{\widehat{=}} & \langle C,c\rangle^\frown u \mathbin{|||} \langle A\rangle \\
\langle A\rangle \mathbin{|||} \langle C,\checkmark\rangle^\frown u & \mathrel{\widehat{=}} & \langle C,\checkmark\rangle^\frown u \mathbin{|||} \langle A\rangle \\
\langle A\rangle \mathbin{|||} \langle C,\mathit{tock}\rangle^\frown u & \mathrel{\widehat{=}} & \langle C,\mathit{tock}\rangle^\frown u \mathbin{|||} \langle A\rangle \\
\langle A,a\rangle^\frown u_1 \mathbin{|||} \langle C,c\rangle^\frown u_2 & \mathrel{\widehat{=}} & (\langle A\rangle \mathbin{|||} \langle C\rangle)\{\langle a\rangle\} \\
 & & \quad (u_1 \mathbin{|||} \langle C,c\rangle^\frown u_2) \cup \\
 & & (\langle A\rangle \mathbin{|||} \langle C\rangle)\{\langle c\rangle\} \\
 & & \quad (\langle A,a\rangle^\frown u_1 \mathbin{|||} u_2) \\
\langle A,a\rangle^\frown u_1 \mathbin{|||} \langle C,\checkmark\rangle^\frown u_2 & \mathrel{\widehat{=}} & (\langle A\rangle \mathbin{|||} \langle C\rangle)\{\langle a\rangle\} \\
 & & \quad (u_1 \mathbin{|||} \langle C,\checkmark\rangle^\frown u_2) \cup \\
 & & (\langle A\rangle \mathbin{|||} \langle C\rangle)\{\langle\checkmark\rangle^\frown u_2\} \\
\langle A,a\rangle^\frown u_1 \mathbin{|||} \langle C,\mathit{tock}\rangle^\frown u_2 & \mathrel{\widehat{=}} & (\langle A\rangle \mathbin{|||} \langle C\rangle)\{\langle a\rangle\} \\
 & & \quad (u_1 \mathbin{|||} \langle C,\mathit{tock}\rangle^\frown u_2) \\
\langle A,\checkmark\rangle^\frown u_1 \mathbin{|||} \langle C,\checkmark\rangle^\frown u_2 & \mathrel{\widehat{=}} & (\langle A\rangle \mathbin{|||} \langle C\rangle)\{\langle\checkmark\rangle^\frown u_1\} \\
\langle A,\checkmark\rangle^\frown u_1 \mathbin{|||} \langle C,\mathit{tock}\rangle^\frown u_2 & \mathrel{\widehat{=}} & (\langle A\rangle \mathbin{|||} \langle C\rangle)\{\langle\checkmark\rangle^\frown u_1\} \\
\langle A,\mathit{tock}\rangle^\frown u_1 \mathbin{|||} \langle C,\mathit{tock}\rangle^\frown u_2 & \mathrel{\widehat{=}} & (\langle A\rangle \mathbin{|||} \langle C\rangle)\{\mathit{tock}\}(u_1 \mathbin{|||} u_2) \\
\langle A,\checkmark\rangle^\frown u_1 \mathbin{|||} \langle C,c\rangle^\frown u_2 & \mathrel{\widehat{=}} & \langle C,c\rangle^\frown u_2 \mathbin{|||} \langle A,\checkmark\rangle^\frown u_1 \\
\langle A,\mathit{tock}\rangle^\frown u_1 \mathbin{|||} \langle C,c\rangle^\frown u_2 & \mathrel{\widehat{=}} & \langle C,c\rangle^\frown u_2 \mathbin{|||} \langle A,\mathit{tock}\rangle^\frown u_1 \\
\langle A,\mathit{tock}\rangle^\frown u_1 \mathbin{|||} \langle C,\checkmark\rangle^\frown u_2 & \mathrel{\widehat{=}} & \langle C,\checkmark\rangle^\frown u_2 \mathbin{|||} \langle A,\mathit{tock}\rangle^\frown u_1.
\end{array}$$

We define inductively a hiding operator on tests, of the form $(\cdot) \setminus A : \mathit{TEST} \longrightarrow \mathit{TEST}$. Here $A \subseteq \Sigma$.

$$\begin{array}{rcll}
\langle B\rangle \setminus A & \mathrel{\widehat{=}} & \langle B\rangle & \text{if } A \subseteq^* B \\
 & \mathrel{\widehat{=}} & \langle\bullet\rangle & \text{if } A \not\subseteq^* B \\
(\langle B,a\rangle^\frown u) \setminus A & \mathrel{\widehat{=}} & u \setminus A & \text{if } a \in A \\
 & \mathrel{\widehat{=}} & (\langle B\rangle \setminus A)^\frown\langle a\rangle^\frown(u \setminus A) & \text{if } a \notin A.
\end{array}$$

Let $A \subseteq \Sigma^{\checkmark}$. A test u is *A-urgent* if, whenever $\langle B, \mathit{tock}\rangle$ in u, then $A \subseteq^* B$. If $A = \{a\}$ is a singleton, we write a-urgent instead of $\{a\}$-urgent.

We define a function $\mathsf{RefCl} : \mathit{TEST} \longrightarrow \mathcal{P}(\mathit{TEST})$: for u a test, we let $v \in \mathsf{RefCl}(u)$ if $v \prec u \wedge \mathsf{trace}(v) = \mathsf{trace}(u)$. We extend the definition of RefCl to sets of tests by setting, for $P \subseteq \mathit{TEST}$, $\mathsf{RefCl}(P) \mathrel{\widehat{=}} \bigcup\{\mathsf{RefCl}(u) \mid u \in P\}$.

If $f : \Sigma \longrightarrow \Sigma$ is a renaming function and $u = \langle A_0, a_1, A_1, \ldots, a_k, A_k\rangle \in \mathit{TEST}$, we define $f(u) \mathrel{\widehat{=}} \langle f(A_0), f(a_1), f(A_1), \ldots, f(a_k), f(A_k)\rangle$, where f is extended to $\Sigma^{\checkmark}_{tock}$ so that $f(tock) \mathrel{\widehat{=}} tock$ and $f(\checkmark) \mathrel{\widehat{=}} \checkmark$, and $f(A_i)$ is either $\bullet$ if $A_i = \bullet$, or $\{f(a) \mid a \in A_i\}$ otherwise.

As one would expect, *semantic bindings* in this model are functions $\rho : VAR \longrightarrow \mathcal{M}_R$. Again, we write $\rho[X := P]$ to denote the semantic binding that is the same as ρ except that it returns P for X instead of $\rho(X)$. Similarly, $\lambda\, x.\rho[X := x]$ represents a function which, when fed an element P of $\mathcal{M}_R$, returns the semantic binding $\rho[X := P]$. Here $X \in VAR$, and x is a *bona fide* variable (not an element of VAR!) ranging over $\mathcal{M}_R$.

For $F : \mathcal{M}_R \longrightarrow \mathcal{M}_R$, we define $\mathsf{fix}(F)$ to be the unique fixed point of F in $\mathcal{M}_R$. We stipulate for convenience that, should F not have a fixed point, or should this fixed point not be unique, $\mathsf{fix}(F)$ should denote an arbitrary, but fixed, element of $\mathcal{M}_R$.

We now define the function $\mathcal{R}[\![\cdot]\!]$ inductively over the structure of Timed CSP terms. Since a term P may contain free variables, we also require a semantic binding ρ in order to assign $\mathcal{M}_R$ processes to the free variables of P.[5] In what follows, $u, u_1, u_2, v, w \in TEST$, $a, c_i \in \Sigma^{\checkmark}_{tock}$, $A \subseteq \Sigma$, $B \subseteq \Sigma^{\checkmark}$, and $C_i \in REF$. The rules are as follows:

$$\begin{aligned}
\mathcal{R}[\![STOP]\!]\rho \;&\widehat{=}\; \{u \mid \mathsf{trace}(u) \leq \langle tock\rangle^{\infty}\}\\
\mathcal{R}[\![SKIP]\!]\rho \;&\widehat{=}\; \{u \mid \mathsf{trace}(u) \leq \langle tock\rangle^{\infty} \wedge \checkmark \notin \mathsf{refusals}(u)\} \cup\\
&\quad\; \{u^\frown\langle\checkmark\rangle^\frown v \mid \mathsf{trace}(u) \leq \langle tock\rangle^{\infty} \wedge\\
&\quad\quad \checkmark \notin \mathsf{refusals}(u) \wedge \mathsf{trace}(v) \leq \langle tock\rangle^{\infty}\}\\
\mathcal{R}[\![WAIT\ n]\!]\rho \;&\widehat{=}\; \{u \mid 0 \leqslant k < n \wedge \mathsf{trace}(u) = \langle tock\rangle^{k}\} \cup\\
&\quad\; \{\hat{u}^\frown v \mid \mathsf{trace}(u) = \langle tock\rangle^{n} \wedge\\
&\quad\quad \mathsf{trace}(v) \leq \langle tock\rangle^{\infty} \wedge \checkmark \notin \mathsf{refusals}(v)\} \cup\\
&\quad\; \{\hat{u}^\frown v^\frown\langle\checkmark\rangle^\frown w \mid \mathsf{trace}(u) = \langle tock\rangle^{n} \wedge\\
&\quad\quad \mathsf{trace}(v) \leq \langle tock\rangle^{\infty} \wedge \checkmark \notin \mathsf{refusals}(v) \wedge\\
&\quad\quad \mathsf{trace}(w) \leq \langle tock\rangle^{\infty}\}
\end{aligned}$$

[5] As was the case for the timed failures model, what we are really defining here is a mapping $\mathcal{R} : \mathbf{TCSP} \times BIND \longrightarrow \mathcal{M}_R$, where $BIND$ stands for the set of all semantic bindings. However, since it again turns out that the denotational values of $\overline{\mathbf{TCSP}}$ *programs* are independent of semantic bindings, this point should not overly concern us.

$$
\begin{aligned}
\mathcal{R}[\![P_1 \overset{n}{\rhd} P_2]\!]\rho &\mathrel{\widehat{=}} \{u \mid \langle tock\rangle^n \not\leq \mathsf{trace}(u) \wedge u \in \mathcal{R}[\![P_1]\!]\rho\} \cup \\
&\quad\ \{\hat{u}^\frown\langle\bullet\rangle^\frown\check{v} \mid \mathsf{trace}(u) = \langle tock\rangle^n \wedge \\
&\qquad \langle tock\rangle \not\leq \mathsf{trace}(v) \wedge \hat{u}^\frown v \in \mathcal{R}[\![P_1]\!]\rho\} \cup \\
&\quad\ \{\hat{u}^\frown v \mid \mathsf{trace}(u) = \langle tock\rangle^n \wedge u \in \mathcal{R}[\![P_1]\!]\rho \wedge \\
&\qquad v \in \mathcal{R}[\![P_2]\!]\rho\} \\
\mathcal{R}[\![a \longrightarrow P]\!]\rho &\mathrel{\widehat{=}} \{u \mid \mathsf{trace}(u) \leq \langle tock\rangle^\infty \wedge a \notin \mathsf{refusals}(u)\} \cup \\
&\quad\ \{u^\frown\langle a\rangle^\frown v \mid \mathsf{trace}(u) \leq \langle tock\rangle^\infty \wedge \\
&\qquad a \notin \mathsf{refusals}(u) \wedge v \in \mathcal{R}[\![P]\!]\rho\} \\
\mathcal{R}[\![a : A \longrightarrow P(a)]\!]\rho &\mathrel{\widehat{=}} \{u \mid \mathsf{trace}(u) \leq \langle tock\rangle^\infty \wedge A \cap \mathsf{refusals}(u) = \emptyset\} \cup \\
&\quad\ \{u^\frown\langle a\rangle^\frown v \mid a \in A \wedge \mathsf{trace}(u) \leq \langle tock\rangle^\infty \wedge \\
&\qquad A \cap \mathsf{refusals}(u) = \emptyset \wedge v \in \mathcal{R}[\![P(a)]\!]\rho\} \\
\mathcal{R}[\![P_1 \,\square\, P_2]\!]\rho &\mathrel{\widehat{=}} \{u \mid \mathsf{trace}(u) \leq \langle tock\rangle^\infty \wedge u \in \mathcal{R}[\![P_1]\!]\rho \cap \mathcal{R}[\![P_2]\!]\rho\} \cup \\
&\quad\ \{u^\frown\langle a\rangle^\frown v \mid \mathsf{trace}(u) \leq \langle tock\rangle^\infty \wedge a \neq tock \wedge \\
&\qquad u^\frown\langle a\rangle^\frown v \in \mathcal{R}[\![P_1]\!]\rho \cup \mathcal{R}[\![P_2]\!]\rho \wedge \\
&\qquad u \in \mathcal{R}[\![P_1]\!]\rho \cap \mathcal{R}[\![P_2]\!]\rho\} \\
\mathcal{R}[\![P_1 \sqcap P_2]\!]\rho &\mathrel{\widehat{=}} \mathcal{R}[\![P_1]\!]\rho \cup \mathcal{R}[\![P_2]\!]\rho \\
\mathcal{R}[\![P_1 \underset{B}{\|} P_2]\!]\rho &\mathrel{\widehat{=}} \{u \mid \exists\, u_1, u_2 \bullet u \in u_1 \underset{B}{\|} u_2 \wedge \\
&\qquad u_1 \in \mathcal{R}[\![P_1]\!]\rho \wedge u_2 \in \mathcal{R}[\![P_2]\!]\rho\} \\
\mathcal{R}[\![P_1 \,|||\, P_2]\!]\rho &\mathrel{\widehat{=}} \{u \mid \exists\, u_1, u_2 \bullet u \in u_1 \,|||\, u_2 \wedge \\
&\qquad u_1 \in \mathcal{R}[\![P_1]\!]\rho \wedge u_2 \in \mathcal{R}[\![P_2]\!]\rho\} \\
\mathcal{R}[\![P_1 \,;\, P_2]\!]\rho &\mathrel{\widehat{=}} \mathsf{RefCl}(\{u \setminus \checkmark \mid \mathsf{trace}(u) \upharpoonright \checkmark = \langle\rangle \wedge \\
&\qquad u \text{ is } \checkmark\text{-urgent} \wedge u \in \mathcal{R}[\![P_1]\!]\rho\}) \cup \\
&\quad\ \mathsf{RefCl}(\{\widehat{(u_1 \setminus \checkmark)}^\frown u_2 \mid \mathsf{trace}(u_1) \upharpoonright \checkmark = \langle\rangle \wedge \\
&\qquad u_1 \text{ is } \checkmark\text{-urgent} \wedge u_1{}^\frown\langle\checkmark, \bullet\rangle \in \mathcal{R}[\![P_1]\!]\rho \wedge \\
&\qquad u_2 \in \mathcal{R}[\![P_2]\!]\rho\}) \\
\mathcal{R}[\![P \setminus A]\!]\rho &\mathrel{\widehat{=}} \mathsf{RefCl}(\{u \setminus A \mid u \text{ is } A\text{-urgent} \wedge u \in \mathcal{R}[\![P]\!]\rho\}) \\
\mathcal{R}[\![f^{-1}(P)]\!]\rho &\mathrel{\widehat{=}} \{u \mid f(u) \in \mathcal{R}[\![P]\!]\rho\} \\
\mathcal{R}[\![f(P)]\!]\rho &\mathrel{\widehat{=}} \{\langle C_0, f(c_1), C_1, \ldots, f(c_k), C_k\rangle \mid k \geqslant 0 \wedge \\
&\qquad \langle f^{-1}(C_0), c_1, f^{-1}(C_1), \ldots, c_k, f^{-1}(C_k)\rangle \in \mathcal{R}[\![P]\!]\rho\} \\
\mathcal{R}[\![X]\!]\rho &\mathrel{\widehat{=}} \rho(X) \\
\mathcal{R}[\![\mu\, X \bullet P]\!]\rho &\mathrel{\widehat{=}} \mathsf{fix}(\lambda\, x.\mathcal{R}[\![P]\!](\rho[X := x])).
\end{aligned}
$$

We first need to show that $\mathcal{F}_T[\![\cdot]\!]$ is well-defined.

Proposition 5.2 *For any term $P \in \mathbf{TCSP}$, and any semantic binding ρ, $\mathcal{R}[\![P]\!]\rho \in \mathcal{M}_R$.*

Proof (Sketch.) The proof is a structural induction over terms. One must verify in turn that each term, under an arbitrary semantic binding ρ, satisfies the axioms of $\mathcal{M}_R$. The details can be found in Appendix A. ■

Proposition 5.3 *If P is a Timed CSP* program *(i.e., $P \in \overline{\mathbf{TCSP}}$), then for any semantic bindings ρ and ρ', $\mathcal{R}[\![P]\!]\rho = \mathcal{R}[\![P]\!]\rho'$.*

Proof (Sketch.) This follows easily by structural induction on terms. The induction hypothesis states that $\mathcal{R}[\![P]\!]\rho = \mathcal{R}[\![P]\!]\rho'$ as long as ρ and ρ' agree on all the free variables of P. Since a program has no free variables, the result follows. ■

We will henceforth drop mention of semantic bindings when calculating the semantics of programs.

We now show that recursions have unique fixed points. Our treatment is slightly more detailed than what is usually found in similar situations in the literature, making explicit some of the steps occasionally omitted elsewhere. We first require a number of preliminaries.

Let us define the *duration* of a test as the number of *tock*s that are recorded in its trace. If $P \in \mathcal{M}_R$, the *restriction of P to behaviours of duration k or less* is defined thus:

$$P(k) \quad \widehat{=} \quad \{u \in P \mid \sharp(\mathsf{trace}(u) \restriction tock) \leqslant k\}.$$

If $P \in \overline{\mathbf{TCSP}}$, we let $P(k) \mathrel{\widehat{=}} \mathcal{R}[\![P]\!](k)$.

We define a metric d on $\mathcal{M}_R$:

$$d(P,Q) \quad \widehat{=} \quad \inf(\{2^{-k} \mid P(k) = Q(k)\} \cup \{1\}).$$

Proposition 5.4 *$(\mathcal{M}_R, d)$ is a complete ultrametric space.*

Proof It is clear that $d(P,Q) = 0 \Leftrightarrow P = Q$ since tests are finite, and that $d(P,Q) = d(Q,P)$. To see that the strong triangle inequality $d(P,R) \leqslant \max\{d(P,Q), d(Q,R)\}$ also holds, assume $P \neq R$ (otherwise the result is trivial). It is now easy to see that there must be $k \in \mathbb{N}$ such that $d(P,R) =$

2^{-k}. In other words, $P(k) = R(k)$ but $P(k+1) \neq R(k+1)$. If $Q = P$, the result follows. Otherwise, we argue again that P and Q have the same behaviour up to, but not exceeding, duration k': $P(k') = Q(k')$ but $P(k'+1) \neq Q(k'+1)$. Now if $k' \leqslant k$, we are done. Otherwise, we must have $P(k+1) = Q(k+1)$, which of course implies that $Q(k+1) \neq R(k+1)$. Thus $d(Q, R) = 2^{-k}$, and again the triangle inequality is satisfied.

To establish completeness, let $\langle P_i \rangle_{i \in \mathbb{N}}$ be a Cauchy sequence in $\mathcal{M}_R$. Let $\langle n_j \rangle_{j \in \mathbb{N}}$ be a sequence of positive integers such that, for all j, and for all $m \geqslant n_j$, $d(P_{n_j}, P_m) \leqslant 2^{-j}$. Let $P = \bigcup_{j \geqslant 0} \{P_{n_j}(j)\}$.

We claim that $P \in \mathcal{M}_R$ and that $P = \lim_{i \to \infty} P_i$. All of the axioms of $\mathcal{M}_R$, with the possible exception of *R7*, are plain consequences of the definition of P and the fact that each $P_i \in \mathcal{M}_R$. For *R7*, note that whenever $u \in P$ is a test of duration k, then $u \in P_{n_k}(k)$. Since P_{n_k} satisfies *R7*, then so must P.

Lastly, we show that the P_i's converge to P. Let $j \geqslant 0$ be given. By definition of P we have $P_{n_j}(j) \subseteq P(j)$, and therefore for all $m \geqslant n_j$ we have $P_m(j) \subseteq P(j)$. On the other hand, if $u \in P(j)$ is a test of duration $k (\leqslant j)$, we must have $u \in P_m(k)$ for some m and hence for all m sufficiently large (in fact, it clearly suffices that $m \geqslant n_j$). Since $P_m(k) \subseteq P_m(j)$, we get $P_m(j) = P(j)$ for all sufficiently large m, or in other words $d(P_m, P) \leqslant 2^{-j}$ for all sufficiently large m, as required. ■

Note that ultrametric spaces are of course also metric spaces.

It is clear from the definition of the semantic mapping $\mathcal{F}_T[\![\cdot]\!]$ above that every Timed CSP operator apart from X and μX can be defined as an operator of the appropriate parity on $\mathcal{M}_R$. For example, *STOP* corresponds to a nullary operator, $a \longrightarrow (\cdot)$ corresponds to a unary operator, and $(\cdot) \;\square\; (\cdot)$ corresponds to a binary operator. The definition of these over $\mathcal{M}_R$ can be lifted verbatim from the definition of $\mathcal{F}_T[\![\cdot]\!]$, with an identical proof of well-definedness.

Terms in **TCSP** may be viewed as 'formal functions' in a certain algebra, with variables taken from the set *VAR*. By the above, each term represents a *bona fide* function $F : {\mathcal{M}_R}^k \longrightarrow \mathcal{M}_R$, for some $k \in \mathbb{N}$. The μX operator is once again interpreted as computing unique fixed points (yielding an arbitrary, but fixed, value should no unique fixed point exist). The value of k is the *arity* of the formal function. $a \longrightarrow X$, for instance, has arity 1, *STOP* has arity 0, and $\mu X \centerdot b \xrightarrow{3} (X \parallel Y)$ has arity 1, since the variable X in it is bound. Note that the formal functions X and Y, both of arity 1, technically represent the same function $Id : \mathcal{M}_R \longrightarrow \mathcal{M}_R$, but are formally considered to be distinct; and indeed, the functions associated with the terms $X \mid\mid\mid Y$ and $X \mid\mid\mid X$ are clearly different. From now on, we will abuse notation and identify terms with functions, while keeping track of the names of the free variables.

For any $k \in \mathbb{N}$, we construe the space ${\mathcal{M}_R}^k$ as an ultrametric space by imposing the 'sup' metric on it: if $\vec{P} = (P_1, P_2, \ldots, P_k)$, and similarly for $\vec{Q}$, we let $d(\vec{P}, \vec{Q}) \mathrel{\widehat{=}} \sup\{d(P_i, Q_i) \mid 1 \leqslant i \leqslant k\}$.

Let $F : (M_1, d_1) \longrightarrow (M_2, d_2)$ be a mapping of metric spaces. We say that F is *non-expanding* if, for all $x, y \in M_1$, $d_2(F(x), F(y)) \leqslant d_1(x, y)$, and that F is a *contraction* if there exists $\delta < 1$ such that, for all $x, y \in M_1$, $d_2(F(x), F(y)) \leqslant \delta d_1(x, y)$.

If $F = F(X, \vec{Y})$ represents a function of arity $k + 1$, we denote by $F_{\vec{Y}} = F_{\vec{Y}}(X)$ the associated function of arity 1 in which $\vec{Y}$ is fixed. (Strictly speaking, there is a whole family of such functions, one for each possible value that $\vec{Y}$ could take in ${\mathcal{M}_R}^k$.)

A function $F = F(X, \vec{Y})$ is *non-expanding in* X if $F_{\vec{Y}}$ is non-expanding for each choice of $\vec{Y} \in {\mathcal{M}_R}^k$, and likewise F is a *contraction in* X if $F_{\vec{Y}}$ is a contraction for all $\vec{Y}$. Note that a function is always a contraction in any variable which does not occur freely in it.

Lemma 5.5

1. *Any contraction is non-expanding.*
2. *A function is non-expanding (resp. a contraction) if and only if it is non-expanding (resp. a contraction) in each variable separately.*
3. *The composition of non-expanding functions is non-expanding.*
4. *If all the arguments to a non-expanding function represent contractions in a given variable, then the whole composition is a contraction in that variable.*
5. *Likewise, if all the arguments to a contraction are non-expanding functions, then the result is still a contraction.*

We omit the straightforward proof. Note that these results hold in arbitrary metric spaces, provided (for 2.) that we use the sup metric in product spaces.

Lemma 5.6 *Every term in* **TCSP** *is (i.e., corresponds to) a non-expanding function. Moreover, if $P = P(X, \vec{Y})$ is a term which is time-guarded for X, then P is a contraction in X.*

Proof (Sketch.) We show that every Timed CSP operator apart from recursion is non-expanding by noting that every behaviour of duration k prescribed by the operator originates from behaviours of duration no greater than k in its arguments. In the case of μX, an examination of the construction of the fixed point as per the Banach fixed point theorem (Theorem A.8) yields the required result. This rests on the fact that time-guardedness for a variable corresponds to contraction in that variable. We demonstrate this by observing that, if P is time-guarded for X, it follows that behaviours of X of duration k give rise to behaviours of P of duration at least $k+1$; this entails that P is a contraction in X. Details can be found in Appendix A. ■

Proposition 5.7 *Let $P = P(X, \vec{Y})$ be a* **TCSP** *term which is time-guarded for X. Then $\mu X \centerdot P$ is the unique function $F = F(\vec{Y})$ such that $F(\vec{Y}) = P(F(\vec{Y}), \vec{Y})$.*

Proof (Sketch.) This follows from the previous lemma and the Banach fixed point theorem. Details are in Appendix A. ■

A noteworthy special case arises when $P = P(X)$ has only one free variable. Proposition 5.7 then asserts that $\mu X \centerdot P \in \mathcal{M}_R$ is the unique solution to the equation $X = P$.

5.2 Operational Semantics

We now present an operational semantics for $\overline{\textbf{TCSP}}$ programs and establish a congruence theorem relating it to the denotational semantics given in the previous section. The format is similar to that of Section 3.2, listing a set of rules of inference from which one can construct the *labelled transition system (LTS)* of any program.

In contrast with the operational semantics laid out in Chapter 3, here it would be possible to develop an operational semantics in which all internal computational states (or *nodes*) have representations as $\overline{\textbf{TCSP}}$ programs.

However, both for consistency and for convenience when we later discuss specifications, let us define the set $NODE_R$ of *(open) nodes* to contain all integral-delayed Timed CSP terms but without any well-timedness requirement. $\overline{NODE}_R$, naturally, will represent the subset of all *(closed) nodes*. These sets may respectively be abbreviated $NODE$ and $\overline{NODE}$ if no confusion is likely. The remainder of our conventions on Timed CSP syntax, as listed in Section 2.1, apply. We again insist that our inference rules only apply to closed nodes. Note that $\mathbf{TCSP} \subseteq NODE$ and $\overline{\mathbf{TCSP}} \subseteq \overline{NODE}$.

Other conventions are as follows: a and b represent visible non-*tock* events, i.e., elements of $\Sigma^{\checkmark}$. μ can be a visible non-*tock* event or a silent one ($\mu \in \Sigma^{\checkmark} \cup \{\tau\}$), and x can be a μ or a *tock*. $P \xrightarrow{x} P'$ means that the node P can perform an immediate x*-transition*, and become the node P' (communicating x in the process if x is a visible event). $P \stackrel{x}{\not\longrightarrow}$ means that P cannot possibly do an x.

The transition rules are as follows:

$$\frac{}{STOP \xrightarrow{tock} STOP} \tag{5.1}$$

$$\frac{}{SKIP \xrightarrow{tock} SKIP} \tag{5.2}$$

$$\frac{}{SKIP \xrightarrow{\checkmark} STOP} \tag{5.3}$$

$$\frac{}{WAIT\ n \xrightarrow{tock} WAIT\ (n-1)}\ [\, n \geqslant 1\,] \tag{5.4}$$

$$\frac{}{WAIT\ 0 \xrightarrow{\tau} SKIP} \tag{5.5}$$

$$\frac{P_1 \xrightarrow{tock} P_1'}{P_1 \overset{n}{\rhd} P_2 \xrightarrow{tock} P_1' \overset{n-1}{\rhd} P_2}\ [\, n \geqslant 1\,] \tag{5.6}$$

$$\frac{}{P_1 \overset{0}{\rhd} P_2 \xrightarrow{\tau} P_2} \tag{5.7}$$

$$\frac{P_1 \xrightarrow{\tau} P_1'}{P_1 \overset{n}{\rhd} P_2 \xrightarrow{\tau} P_1' \overset{n}{\rhd} P_2} \tag{5.8}$$

$$\frac{P_1 \xrightarrow{a} P_1'}{P_1 \overset{n}{\rhd} P_2 \xrightarrow{a} P_1'} \tag{5.9}$$

$$\frac{}{(a \longrightarrow P) \xrightarrow{tock} (a \longrightarrow P)} \tag{5.10}$$

$$\frac{}{(a \longrightarrow P) \xrightarrow{a} P} \tag{5.11}$$

$$\frac{}{(a : A \longrightarrow P(a)) \xrightarrow{tock} (a : A \longrightarrow P(a))} \tag{5.12}$$

$$\frac{}{(a : A \longrightarrow P(a)) \xrightarrow{b} P(b)} \; [\, b \in A \,] \tag{5.13}$$

$$\frac{P_1 \xrightarrow{tock} P_1' \quad P_2 \xrightarrow{tock} P_2'}{P_1 \,\Box\, P_2 \xrightarrow{tock} P_1' \,\Box\, P_2'} \tag{5.14}$$

$$\frac{P_1 \xrightarrow{\tau} P_1'}{P_1 \,\Box\, P_2 \xrightarrow{\tau} P_1' \,\Box\, P_2} \qquad \frac{P_2 \xrightarrow{\tau} P_2'}{P_1 \,\Box\, P_2 \xrightarrow{\tau} P_1 \,\Box\, P_2'} \tag{5.15}$$

$$\frac{P_1 \xrightarrow{a} P_1'}{P_1 \,\Box\, P_2 \xrightarrow{a} P_1'} \qquad \frac{P_2 \xrightarrow{a} P_2'}{P_1 \,\Box\, P_2 \xrightarrow{a} P_2'} \tag{5.16}$$

$$\frac{}{P_1 \sqcap P_2 \xrightarrow{\tau} P_1} \qquad \frac{}{P_1 \sqcap P_2 \xrightarrow{\tau} P_2} \tag{5.17}$$

$$\frac{P_1 \xrightarrow{tock} P_1' \quad P_2 \xrightarrow{tock} P_2'}{P_1 \underset{B}{\parallel} P_2 \xrightarrow{tock} P_1' \underset{B}{\parallel} P_2'} \tag{5.18}$$

$$\frac{P_1 \xrightarrow{\mu} P_1'}{P_1 \underset{B}{\parallel} P_2 \xrightarrow{\mu} P_1' \underset{B}{\parallel} P_2} \; [\, \mu \notin B, \mu \neq \checkmark \,] \tag{5.19a}$$

$$\frac{P_2 \xrightarrow{\mu} P_2'}{P_1 \underset{B}{\parallel} P_2 \xrightarrow{\mu} P_1 \underset{B}{\parallel} P_2'} \; [\, \mu \notin B, \mu \neq \checkmark \,] \tag{5.19b}$$

$$\frac{P_1 \xrightarrow{a} P_1' \quad P_2 \xrightarrow{a} P_2'}{P_1 \underset{B}{\parallel} P_2 \xrightarrow{a} P_1' \underset{B}{\parallel} P_2'} \; [\, a \in B \,] \tag{5.20}$$

$$\frac{P_1 \xrightarrow{\checkmark} P_1'}{P_1 \underset{B}{\parallel} P_2 \xrightarrow{\checkmark} P_1'} \; [\, \checkmark \notin B \,] \qquad \frac{P_2 \xrightarrow{\checkmark} P_2'}{P_1 \underset{B}{\parallel} P_2 \xrightarrow{\checkmark} P_2'} \; [\, \checkmark \notin B \,] \tag{5.21}$$

$$\frac{P_1 \xrightarrow{tock} P_1' \quad P_2 \xrightarrow{tock} P_2'}{P_1 \mathbin{|||} P_2 \xrightarrow{tock} P_1' \mathbin{|||} P_2'} \tag{5.22}$$

$$\frac{P_1 \xrightarrow{\mu} P_1'}{P_1 \mathbin{|||} P_2 \xrightarrow{\mu} P_1' \mathbin{|||} P_2} \; [\, \mu \neq \checkmark \,] \tag{5.23a}$$

$$\frac{P_2 \xrightarrow{\mu} P_2'}{P_1 \mathbin{|||} P_2 \xrightarrow{\mu} P_1 \mathbin{|||} P_2'} \; [\, \mu \neq \checkmark \,] \tag{5.23b}$$

$$\frac{P_1 \xrightarrow{\checkmark} P_1'}{P_1 \mathbin{|||} P_2 \xrightarrow{\checkmark} P_1'} \qquad \frac{P_2 \xrightarrow{\checkmark} P_2'}{P_1 \mathbin{|||} P_2 \xrightarrow{\checkmark} P_2'} \tag{5.24}$$

$$\frac{P_1 \xrightarrow{tock} P_1' \quad P_1 \not\xrightarrow{\checkmark}}{P_1 \; ; P_2 \xrightarrow{tock} P_1' \; ; P_2} \tag{5.25}$$

$$\frac{P_1 \xrightarrow{\checkmark} P_1'}{P_1 \; ; P_2 \xrightarrow{\tau} P_2} \tag{5.26}$$

$$\frac{P_1 \xrightarrow{\mu} P_1'}{P_1 \; ; P_2 \xrightarrow{\mu} P_1' \; ; P_2} \; [\, \mu \neq \checkmark \,] \tag{5.27}$$

$$\frac{P \xrightarrow{tock} P' \quad \forall\, a \in A \bullet P \not\xrightarrow{a}}{P \setminus A \xrightarrow{tock} P' \setminus A} \tag{5.28}$$

$$\frac{P \xrightarrow{a} P'}{P \setminus A \xrightarrow{\tau} P' \setminus A} \; [\, a \in A \,] \tag{5.29}$$

$$\frac{P \xrightarrow{\mu} P'}{P \setminus A \xrightarrow{\mu} P' \setminus A} \; [\, \mu \notin A \,] \tag{5.30}$$

$$\frac{P \xrightarrow{tock} P'}{f^{-1}(P) \xrightarrow{tock} f^{-1}(P')} \tag{5.31}$$

$$\frac{P \xrightarrow{\mu} P'}{f^{-1}(P) \xrightarrow{\mu} f^{-1}(P')} \; [\, \mu \in \{\tau, \checkmark\} \,] \tag{5.32}$$

$$\frac{P \xrightarrow{f(a)} P'}{f^{-1}(P) \xrightarrow{a} f^{-1}(P')} \tag{5.33}$$

$$\frac{P \xrightarrow{tock} P'}{f(P) \xrightarrow{tock} f(P')} \tag{5.34}$$

$$\frac{P \xrightarrow{\mu} P'}{f(P) \xrightarrow{\mu} f(P')} \; [\, \mu \in \{\tau, \checkmark\} \,] \tag{5.35}$$

$$\frac{P \xrightarrow{a} P'}{f(P) \xrightarrow{f(a)} f(P')} \tag{5.36}$$

$$\frac{}{\mu X \centerdot P \xrightarrow{\tau} P[(\mu X \centerdot P)/X].} \tag{5.37}$$

The remark made in Section 3.2 concerning negative premisses (appearing in Rules 5.25 and 5.28) applies here as well.

The effects of these rules follow our intuition: Rule 5.10, for instance, ensures the unhindered passage of time in a prefixed process; Rule 5.14 guarantees the temporal soundness of external choice; Rules 5.18 and 5.22 force time to flow at a uniform rate; and Rules 5.25 and 5.28 implement τ-urgency. Notice, too, the one-to-one correspondence between these rules and their counterparts in Section 3.2; this will be formally exploited in Proposition 6.2 in the next chapter.

The operational semantics enjoys a number of properties, which we list after the following definitions.

If P and Q are open nodes, we write $P \equiv Q$ to indicate that P and Q are syntactically identical.

If P is a closed node, we define $\mathsf{init}^{\tau}_{R}(P)$ ($\mathsf{init}^{\tau}(P)$ for short) to be the set of visible non-*tock* and silent events that P can immediately perform: $\mathsf{init}^{\tau}_{R}(P) \mathrel{\widehat{=}} \{\mu \in \Sigma^{\checkmark} \cup \{\tau\} \mid P \xrightarrow{\mu}\}$. We also write $\mathsf{init}_{R}(P)$ ($\mathsf{init}(P)$ for short) to represent the set of visible non-*tock* events that P can immediately perform: $\mathsf{init}_{R}(P) \mathrel{\widehat{=}} \mathsf{init}^{\tau}_{R}(P) \cap \Sigma^{\checkmark}$.

For P a closed node, we define an *execution* of P to be a sequence $e = P_0 \xrightarrow{x_1} P_1 \xrightarrow{x_2} \ldots \xrightarrow{x_n} P_n$ (with $n \geqslant 0$), where $P_0 \equiv P$, the P_i's are nodes, and each transition $P_i \xrightarrow{x_{i+1}} P_{i+1}$ in e is validly allowed by the operational inference Rules 5.1–5.37. Here we have each $x_i \in \Sigma^{\checkmark}_{tock} \cup \{\tau\}$. The set of executions of P is written $\mathsf{exec}_{R}(P)$, or $\mathsf{exec}(P)$ for short when no confusion is likely. By convention, writing down a transition (or sequence thereof) such as $P \xrightarrow{a} P'$ is equivalent to stating that $P \xrightarrow{a} P' \in \mathsf{exec}(P)$.

For P a closed node, the P-rooted graph, or *labelled transition system*, incorporating all of P's possible executions is denoted $\mathsf{LTS}_R(P)$, or $\mathsf{LTS}(P)$ if no confusion is likely.

If $tr = \langle x_1, x_2, \ldots, x_n \rangle$ is a τ-*trace* (i.e., $tr \in (\Sigma^{\checkmark}_{tock} \cup \{\tau\})^{\star}$), we write $P \overset{tr}{\Longrightarrow} P'$ to mean that there is some execution $P_0 \xrightarrow{x_1} P_1 \xrightarrow{x_2} \ldots \xrightarrow{x_n} P_n$ of P, with $P \equiv P_0$ and $P' \equiv P_n$, which communicates the τ-trace tr. $P \overset{tr}{\Longrightarrow}$ means that there is some node P' such that $P \overset{tr}{\Longrightarrow} P'$, where $P \overset{\langle\rangle}{\Longrightarrow} P$ simply represents the trivial execution $P \in \mathsf{exec}(P)$.

The proofs of several of the propositions below are straightforward structural inductions on nodes, carried out in a manner similar to that of [Sch95]; in such cases we will therefore omit the details, with the exception of Proposition 5.13 (finite variability), the proof of which is quite interesting and instructive. Alternatively, these propositions can also be seen to follow directly from their counterparts in Section 3.9, thanks to Proposition 6.2, presented in the next chapter.

The following propositions are universally quantified over P, P', P'', which represent closed nodes; we are also using n, m, k, k' to represent natural numbers, etc. As mentioned above, the reader will notice the correspondence between the propositions listed here and those of Section 3.2; it is equally interesting to remark on the propositions of Section 3.2 which do *not* have counterparts here, such as Proposition 3.5 (time continuity)—this, of course, was to be expected given that the model studied in this chapter is discrete in nature.

Proposition 5.8 *Time determinacy:*

$$(P \xrightarrow{tock} P' \wedge P \xrightarrow{tock} P'') \Rightarrow P' \equiv P''.$$

Note that a trivial induction extends this result to an arbitrary number of *tock*s.

Proof (Sketch.) Let us illustrate two cases of the structural induction.

case $P_1 \mid\mid\mid P_2$**:** Observe that the only rule allowing $P_1 \mid\mid\mid P_2$ to communicate a *tock* is Rule 5.22, whose conclusion is $P_1 \mid\mid\mid P_2 \xrightarrow{tock} P_1' \mid\mid\mid P_2'$. The antecedents of this rule require that $P_1 \xrightarrow{tock} P_1'$ and $P_2 \xrightarrow{tock} P_2'$. Applying the induction hypothesis to P_1 and P_2 tells us that P_1' and P_2' are unique, and therefore that so too is $P_1' \mid\mid\mid P_2'$, as required.

case $\mu X \bullet P$**:** This case follows vacuously since no operational rule allows a recursive term to communicate a *tock*; the only behaviour permitted is the unwinding of the recursion, which results in a τ action.

Note that structural inductions such as the one above must technically be carried out over *open* nodes, together with syntactic bindings. However in many situations, including the one at hand, the case of recursion is discharged vacuously under any syntactic binding, which greatly simplifies matters. ■

Proposition 5.9 *Persistency—the set of possible initial visible events remains constant under the occurrence of tocks:*

$$P \xrightarrow{tock} P' \Rightarrow \mathsf{init}(P) = \mathsf{init}(P').$$

Again, this generalises to an arbitrary number of *tock*s. We omit the proof, but remark that a change in the set of possible initial visible events can only arise after the occurrence of a non-*tock*, visible or hidden, event.

Proposition 5.10 *Maximal progress, or τ-urgency:*

$$P \xrightarrow{\tau} \Rightarrow P \overset{tock}{\not\longrightarrow}.$$

Proof (Sketch.) This is again a straightforward structural induction. We cover the case of the hiding operator as an illustration.

First note that any *tock*-transition of $P \setminus A$ (with $A \subseteq \Sigma$) can only occur as a result of an application of Rule 5.28. Now suppose that $P \setminus A \xrightarrow{\tau}$. Two cases arise, depending on whether this τ-transition results from Rule 5.29 or from Rule 5.30. The first case violates the second antecedent of Rule 5.28, which therefore disallows $P \setminus A$ from performing a τ action. In the second case, we can apply the induction hypothesis (which informs us that $P \overset{tock}{\not\longrightarrow}$) to conclude that the first antecedent of Rule 5.28 fails to be satisfied. In either case, therefore, $P \setminus A$ is unable to communicate a *tock*. ■

Corollary 5.11

$$(P \overset{\langle tock \rangle^n}{\Longrightarrow} P' \xrightarrow{\tau} \wedge P \overset{\langle tock \rangle^m}{\Longrightarrow} P'' \xrightarrow{\tau}) \Rightarrow n = m.$$

Proposition 5.12 *A node P can always perform a sequence of tocks up to the point of the next τ action, or up to any point if no τ action lies ahead:*

$$\forall\, k \geqslant 1 \bullet (\nexists\, P' \bullet P \overset{\langle tock \rangle^k}{\Longrightarrow} P') \Rightarrow \exists\, k' < k, P'' \bullet P \overset{\langle tock \rangle^{k'}}{\Longrightarrow} P'' \xrightarrow{\tau}.$$

Proposition 5.13 *Finite variability—a* program $P \in \overline{\mathbf{TCSP}}$ *cannot perform unboundedly many actions within a finite number of tocks:*

$$\forall\, k \geqslant 0 \bullet \exists\, n = n(P,k) \bullet \forall\, tr \in (\Sigma^{\checkmark}_{tock} \cup \{\tau\})^{\star} \bullet$$
$$(P \overset{tr}{\Longrightarrow} \wedge \sharp(tr \restriction tock) \leqslant k) \Rightarrow \sharp tr \leqslant n.$$

We remark that this result holds on account of the fact that programs are well-timed. Note also that this formulation of finite variability is stronger than that postulated by Axiom *R7*, since the latter does not take silent events into account.

Proof (Sketch.) Two proofs of this proposition can be given. The easiest one, by contradiction, is to observe that any failure of finite variability in the current operational semantics would entail a similar failure of finite variability (Proposition 3.9) in the operational semantics for $\mathcal{M}_{TF}$, via Proposition 6.2. This proof is unfortunately not self-contained, since it hinges on an outside result (Proposition 3.9, taken from [Sch95]).

The second proof, this one direct, proceeds by structural induction. The difficult case is that of recursion, for which the notion of *time-bisimulation*, similar to that of indexed bisimulation, is introduced and extensively employed. We present the details in Appendix A. ■

We now establish a congruence theorem between the operational and denotational semantics presented in this chapter. Such a theorem provides instructions to elicit, from the labelled transition system of a program, a set of tests, and then asserts that this set is in fact equal to the denotational representation of the program in $\mathcal{M}_R$.

The techniques required to prove congruence theorems of this kind within CSP frameworks are well-known and described in, among others, [Ros97] (for standard, untimed CSP), [Muk93] (for the refusal testing model), and [Sch95] (for the timed failures model). Our result relies on similar methods.

A refusal $A \in REF$ is *refused* by a node P if either A is the null refusal $\bullet$, or P is stable (cannot perform a τ-transition) and has no valid initial transition labelled with an event from A: P **ref** A if $A = \bullet \vee (P \overset{\tau}{\not\longrightarrow} \wedge |A| \cap \mathsf{init}(P) = \emptyset)$.

Any execution of a program P gives rise to a set of tests by removing programs and τ-transitions from the execution, and inserting as refusals, between the remaining transitions, sets of events that cannot be performed. In detail, letting e be an execution of P and u a test, we define inductively the relation e **test** u by induction on e:

$$\begin{aligned} P\ \mathbf{test}\ \langle A\rangle &\Leftrightarrow P\ \mathbf{ref}\ A \\ (P \xrightarrow{\tau})^\frown e'\ \mathbf{test}\ u &\Leftrightarrow e'\ \mathbf{test}\ u \\ (P \xrightarrow{x})^\frown e'\ \mathbf{test}\ \langle A, x\rangle^\frown u' &\Leftrightarrow x \neq \tau \wedge P\ \mathbf{ref}\ A \wedge e'\ \mathbf{test}\ u'. \end{aligned}$$

We can now define the function Φ_R, which extracts the denotational representation of a program from its set of executions.

Definition 5.2 *For $P \in \overline{NODE}_R$, we set*

$$\Phi_R(P) \ \widehat{=}\ \{u \mid \exists\, e \in \mathsf{exec}_R(P) \centerdot e\ \mathbf{test}\ u\}.$$

The chief congruence result now reads:

Theorem 5.14 *For any* $\overline{\mathbf{TCSP}}$ *program P, we have*

$$\Phi_R(P) = \mathcal{R}[\![P]\!].$$

Proof (Sketch.) The statement must first be rephrased in slightly more general terms, as an assertion about **TCSP** terms and syntactic bindings rather than simply programs. In doing so, it proves convenient to temporarily abandon the axioms of $\mathcal{M}_R$ and work in the raw superspace $\mathcal{P}(TEST)$ instead. A structural induction is then carried out, establishing the stronger statement, which then yields the desired result. Details are in Appendix A. ∎

5.3 Refinement and Specification

We can define a partial order on $\mathcal{P}(TEST)$ in the same way as we did on $\mathcal{P}(TF)$:

Definition 5.3 *For $P, Q \subseteq TEST$ (and in particular for $P, Q \in \mathcal{M}_R$), we let $P \sqsubseteq_R Q$ if $P \supseteq Q$. For $P, Q \in \overline{\mathbf{TCSP}}$, we write $P \sqsubseteq_R Q$ to mean $\mathcal{R}[\![P]\!] \sqsubseteq_R \mathcal{R}[\![Q]\!]$, and $P =_R Q$ to mean $\mathcal{R}[\![P]\!] = \mathcal{R}[\![Q]\!]$.*

We may drop the subscripts and write simply $P \sqsubseteq Q$, $P = Q$ whenever the context is clear.

This order is known as *test refinement*, or the *order of nondeterminism.* It has the central property:

$$P \sqsubseteq Q \Leftrightarrow P \sqcap Q = P.$$

In [Oua97] we showed that this order makes $\mathcal{M}_R$ into a *complete partial order*, or *cpo*[6]. Moreover, we showed that every Timed CSP operator, apart from X and μX, corresponds to a *continuous* function[7]. μX, on the other hand, can be seen to be a continuous operator from the space of continuous endofunctions on $\mathcal{M}_R$ to $\mathcal{M}_R$.

$\mathcal{M}_R$, however, is not a *domain* because it lacks a least element, as a direct consequence of finite variability—Axiom *R7*. Finite variability, of course, follows from the fact that processes are required to be well-timed. It can be shown that it is possible to drop this requirement (as we have done for instance in the operational setting) and still get a consistent denotational model, although in that case it proves useful to include a *divergence* component in the representation of processes—a set of tests after the completion of any of which the process may exhibit divergent behaviour, corresponding to an unbounded sequence of internal (silent) transitions.

This model can be shown to be a domain—a cpo with a least element. $\mathcal{M}_R$ is then identified with the subset of processes which satisfy finite variability and never diverge. The continuity results we have quoted above persist in the larger model, which entails that any recursion has a *least* fixed point, as per Tarski's theorem (see, e.g., [Ros97]). Of course, this least fixed point agrees with the unique fixed point computed in $\mathcal{M}_R$ in the case of well-timed processes. An adequate congruence theorem between this model and the operational semantics given in the previous section can be established, which shows that it is possible to model Zeno processes in a manner which is fully consistent with $\mathcal{M}_R$. Not only is this an interesting result in its own right, it

[6]A partial order M is *complete* if every *directed* subset ($D \subseteq M$ is *directed* if for all $x, y \in D$ there exists $z \in D$ such that $x, y \sqsubseteq z$) of M has a least upper bound in M.

[7]A *continuous* function is one that is monotone and preserves least upper bounds of directed sets.

also enables us to use refinement as a versatile specification tool, as we shall shortly see.

We first define a second refinement order—known as *trace refinement*—between sets of tests (and in particular between $\mathcal{M}_R$ processes):

Definition 5.4 *For any $P, Q \subseteq TEST$, we let $P \sqsubseteq_T Q \Leftrightarrow \mathsf{trace}(P) \supseteq \mathsf{trace}(Q)$.*

(Here the trace operator was naturally extended to apply to sets of tests.) We overload the notation and extend $\sqsubseteq_T$ to $\overline{\mathbf{TCSP}}$ programs in the obvious way.

Specifications are assertions concerning processes. We write $P \vDash S$ to express the fact that process P (or $\mathcal{R}[\![P]\!]$ if P is a program) meets, or *satisfies*, the specification S.

As in the case of $\mathcal{M}_{TF}$, $\mathcal{M}_R$ specifications fall naturally into certain categories. We are particularly (though not exclusively) interested in a type of specification known as *behavioural* specifications. A *behavioural* specification is one that is universally quantified over the behaviours of processes. In other words, $S = S(P)$ is a behavioural specification if there is a predicate $S'(u)$ on tests such that, for any P,

$$P \vDash S \Leftrightarrow \forall\, u \in P \bullet S'(u).$$

In this case we may abuse notation and identify S with S'. Note that S' itself may be identified with a subset of $TEST$, namely the set of $u \in TEST$ such that $S'(u)$.

A *behavioural trace specification* $S = S(P)$ is one for which there exists a predicate $S'(tr)$ on $(\Sigma^{\checkmark}_{tock})^{\star}$ such that, for any P,

$$P \vDash S \Leftrightarrow \forall\, u \in P \bullet S'(\mathsf{trace}(u)).$$

Once again we may identify S with S' if no confusion arises as a result.

Behavioural trace specifications are paradigmatic examples of *safety properties*, requirements of the type: 'Nothing bad happens'. We will in fact not consider any other sort of safety properties in this work. Other types of behavioural specifications include *liveness properties*, requirements that 'something good not be prevented from happening'. Such specifications are typically primarily concerned with refusal information.

We remark as before that any specification $S = S(P)$ which can be expressed, for a fixed process Q, as $Q \sqsubseteq P$, is automatically behavioural. As an example, let us again consider the safety property $S(P)$: 'P cannot perform the event a'. Thanks to our earlier discussion, we can validly express this specification as $RUN_{\Sigma^{\checkmark}-\{a\}} \sqsubseteq_T P$, where $RUN_A = a : A \longrightarrow RUN_A$. We can even express the specification as $CHAOS_{\Sigma^{\checkmark}-\{a\}} \sqsubseteq_R P$, where $CHAOS_A = (\bigsqcap\{a \longrightarrow CHAOS_A \mid a \in A\} \sqcap STOP) \overset{1}{\triangleright} CHAOS_A$.[8]

As we shall see in the next section, the advantage of the refinement formalism is that it enables us to automatically verify the specification by performing a refinement check on a model checker such as FDR.

5.4 Verification

This section addresses the question of automatically checking whether a process meets a given specification. Since this is not the primary focus of this thesis, the exposition will remain rather brief and far from exhaustive.

Let P be a $\overline{\mathbf{TCSP}}$ program. Even though the set $\mathcal{R}[\![P]\!]$ of P's behaviours is always infinite, in many cases P's labelled transition system $\mathsf{LTS}_R(P)$ is finite. A specification S on P can thus often be checked by examining P's LTS. We demonstrate this by considering behavioural specifications of the type $Q \sqsubseteq_R P$ (for a fixed $Q \in \mathcal{M}_R$).

In [Muk93], it is shown that there is a natural projection from the refusal testing model for (untimed) CSP into the standard failures model, and moreover that the semantic mapping for the failures model factors through this projection. A similar result can be shown to hold for our discrete-time refusal testing model $\mathcal{M}_R$ provided that the four conditions listed at the end of Chapter 4—unhindered passage of time, temporal soundness of external choice, uniform rate of passage of time, and maximal progress—are somehow adequately incorporated to the failures model. As discussed in Chapter 4, this cannot be done in a purely denotational setting, but can still be achieved operationally, via a suitable syntactic translation (Ψ), together with the pos-

[8]Note that our use of the general internal choice operator $\bigsqcap$, in the definition of $CHAOS_A$, is simply shorthand for repeated use of the usual internal choice operator $\sqcap$, since $A \subseteq \Sigma^{\checkmark}$ is finite. We also remark that this definition of $CHAOS_A$, meant to capture the $\sqsubseteq_R$-least process with alphabet A, is slightly more complicated than the usual definition seen in standard models of (untimed) CSP, owing to the fact that refusal testing information is more discriminating than mere failure information.

tulation of τ-urgency—the prioritising of silent events over *tock*s.

Let us therefore consider a specification S on $\mathcal{M}_R$ processes which is only concerned with *failures*, i.e., traces together with a final refusal. For any $P \in \overline{\mathbf{TCSP}}$, to verify that $P \vDash S$ in $\mathcal{M}_R$ (i.e., that $\mathcal{R}[\![P]\!] \vDash S$), it is sufficient to show that $\Psi(P) \vDash S$ in the prioritised failures model, by which we mean the model in which failures are derived from the prioritised operational semantics. Since this last check can be performed on the model checker FDR, we are already able to mechanically verify a large class of behavioural specifications.[9] (In general this may also involve a suitable translation of the specification S).

An important subclass of such specifications are safety properties, since these are exclusively concerned with traces (and can therefore obviously be expressed in terms of failures). Thus to verify for example that $\mathcal{R}[\![P]\!]$ never communicates the event a (i.e., that $RUN_{\Sigma^{\checkmark}-\{a\}} \sqsubseteq_T P$), it is sufficient to check that $\Psi(RUN_{\Sigma^{\checkmark}-\{a\}}) \sqsubseteq_T \Psi(P)$ using FDR. Note that the second refinement *must* hold if $\mathcal{R}[\![P]\!]$ meets the specification, although the FDR check will succeed only if P has a finite (in fact manageable) LTS. We provide a concrete example of this technique in Section 6.7.

In general, however, we cannot always depend on this sort of analysis, since there are $\overline{\mathbf{TCSP}}$ programs which have distinct interpretations in $\mathcal{M}_R$ but have identical sets of failures. In the rare cases where this is likely to cause a problem (namely, in those cases in which the specification on tests under consideration cannot be expressed purely in terms of failures), one must directly perform an analysis of the LTS of the program to determine whether the specification holds or not. We provide an algorithm which achieves this task in Appendix B, drawing on the notion of power-simulation which lies at the heart of FDR checks.

[9]A 'priority' operator has been included in beta versions of FDR, but has not, to our knowledge, been officially released. The incorporation of priority in future versions of FDR2 is understood to be currently under consideration by Formal Systems Europe Ltd.

Chapter 6

Timed Analysis (I)

We now address the task of extracting information about the continuous-time behaviour $\mathcal{F}_T[\![P]\!]$ of a $\overline{\mathbf{TCSP}}$ program P from its discrete-time representation $\mathcal{R}[\![P]\!]$.

In Section 6.1 we introduce the notion of the *timed denotational expansion* of an $\mathcal{M}_R$ process. This is an entirely straightforward attempt to elicit, or more precisely to approximate, the continuous-time behaviour of a process from its set of tests. We then continue with a section examining a number of examples highlighting the types of inaccuracies that timed denotational expansions exhibit with respect to exact process behaviour.

In Section 6.3, we focus our analysis on timed traces, and develop tools enabling us to quantitatively assess how closely timed denotational expansions match exact process behaviour. We obtain soundness and quasi-completeness theorems which essentially show that timed denotational expansions are conservative approximations of exact behaviour, and moreover that these approximations get better and better as 'time granularity' (the number of *tock*s postulated to occur within one time unit) gets finer and finer.

Section 6.4 then seeks to exploit these observations and results towards automated verification, concentrating, once again, on timed traces. Several avenues are explored, with the most important and mature approach building upon a result of Henzinger, Manna, and Pnueli's. A number of examples are presented, many of which illustrating the limitations of the various techniques under consideration.

Section 6.5 studies the applicability of our techniques and results to full—i.e., timed failures, or safety and liveness—automated verification. This section in some sense represents the culmination of our work. We extend the previously cited result of Henzinger *et al.* from timed traces to timed failures, and use it as a foundation to produce exact verification theorems (as well as a corresponding model checking algorithm, described in Appendix B).

Section 6.6 answers many questions regarding the scope and limitations of the techniques thus described, particularly in the context of refinement-based specifications.

Finally, Section 6.7 offers a small verification case study illustrating the results and algorithm derived in Sections 6.4 and 6.5.

6.1 Timed Denotational Expansion

If P is a $\overline{\mathbf{TCSP}}$ program, we would like to convert the refusal testing information contained in $\mathcal{R}[\![P]\!]$ into useful information concerning the timed failures of $\mathcal{F}_T[\![P]\!]$. The general intuition behind our approach is in essence captured by the following example: suppose that in a particular test of $\mathcal{R}[\![P]\!]$, the event a occurs after three *tock*s, followed by a b, and then a further *tock*. We would conclude that a and b occurred at times t_a and t_b, respectively, such that $3 \leqslant t_a \leqslant t_b < 4$.[1] Ideally we would then expect there to be timed failures in $\mathcal{F}_T[\![P]\!]$ accounting for these behaviours, and vice-versa.

To this end, we define a function $\mathsf{Exp}_k : \mathcal{M}_R \longrightarrow \mathcal{P}(TF)$ (where Exp stands for '*expansion*', and the integral subscript k indicates how many *tock*s are meant to be signalled per time unit), to transform $\mathcal{M}_R$ processes into sets of compatible timed failures.

In detail, the construction goes as follows. Let $P \in \mathcal{M}_R$ and let $u = \langle A_0, a_1, A_1, a_2, \ldots, a_n, A_n\rangle$ be a test of P. We first define $\mathsf{preExp}_k(u)$ to be the set of timed failures $(s, \aleph) \in TF$ satisfying the following:

[1]For technical reasons, we require the third inequality to be strict, but otherwise there can be no guarantee that events, such as a and b or even *tock*, did not in fact happen 'simultaneously'.

$$\begin{aligned}
&\exists (t_i \in \mathbb{R}^+)_{0 \leqslant i \leqslant n+1} \bullet \\
&\quad 0 = t_0 \leqslant t_1 \leqslant \ldots \leqslant t_n < t_{n+1} = (1 + \sharp(\langle a_1, a_2, \ldots, a_i \rangle \restriction tock))/k \wedge \\
&\quad s = \langle (t_1, a_1), (t_2, a_2), \ldots, (t_n, a_n) \rangle \setminus tock \wedge \\
&\quad (\forall (1 \leqslant i \leqslant n) \bullet \\
&\qquad a_i = tock \Rightarrow (t_i = \sharp(\langle a_1, a_2, \ldots, a_i \rangle \restriction tock)/k \wedge t_{i-1} \neq t_i)) \wedge \\
&\quad \aleph \subseteq \bigcup_{i=0}^{n} \{[t_i, t_{i+1}) \times |A_i|\}.
\end{aligned}$$

We can now give

Definition 6.1 *For any $P \in \mathcal{M}_R$ and $k \geqslant 1$, we define the* timed denotational expansion $\mathsf{Exp}_k(P)$ *of P to be*

$$\mathsf{Exp}_k(P) \quad \widehat{=} \quad \bigcup \{\mathsf{preExp}_k(u) \mid u \in P\}.$$

(We will usually omit the subscript k when it is equal to 1.)

Note how the basic idea laid out earlier has been extended to refusal sets in the definition of Exp_k. A point worthy of mention is that the refusal $\bullet$ is treated as if it were the empty refusal set; we will return to this later on.

We also point out that in general, for $P \in \mathcal{M}_R$, it is *not* the case that $\mathsf{Exp}_k(P)$ is an $\mathcal{M}_{TF}$ process, as we shall soon see.

6.2 Timing Inaccuracies

One would ideally like to have the equality $\mathcal{F}_T[\![P]\!] = \mathsf{Exp}(\mathcal{R}[\![P]\!])$ for all $P \in \overline{\mathbf{TCSP}}$. Unfortunately this cannot be, for a variety of reasons, as we now demonstrate. In the analysis below we shall freely switch back and forth between denotational and operational interpretations of process execution.

Consider the program P_1, defined as follows:

$$P_1 = (\bar{a} \,\Box\, \mathit{WAIT}\ 1)\ ;\ \mathit{STOP}.$$

The behaviour of P_1 in the timed failures model essentially consists in offering a for one time unit, at which point, if a has not occurred, the subprocess WAIT 1 offers a $\checkmark$. Sequential composition then forces this $\checkmark$ to occur, silently, on the spot, unless a occurs instead at that very instant. This latter choice is nondeterministic (and in theory also subject to environmental cooperation). What is certain is that a cannot occur at any time $t > 1$.[2]

The tests of $\mathcal{R}[\![P_1]\!]$ exhibit some subtle but important differences: until the first *tock* is recorded, a does not appear in refusals, i.e., a is continuously on offer, as expected. However, this situation persists *after* the occurrence of *tock*, right up to the time when, operationally, a silent $\checkmark$ is performed (by the subprocess *WAIT* 1). Refusals in the interim (i.e., between the first *tock* and the silent $\checkmark$) are null ($\bullet$), and a is certainly not prevented from occurring. Our semantics only prescribes that the hidden $\checkmark$ occur before the next *tock*, which implies that in certain cases P_1 will still be able to communicate a at times which are arbitrarily close to that of the second *tock* occurrence. Accordingly, the timed denotational expansion of $\mathcal{R}[\![P_1]\!]$ is unable to rule out a purported occurrence of a before two full time units have elapsed. In some sense, the nondeterministic uncertainty witnessed throughout the half-open time interval of one time unit between the first and second *tock*s corresponds precisely to the point nondeterminism observed at the instant $t = 1$ in the timed failures model.

This example also shows that $\mathsf{Exp}(\mathcal{R}[\![P_1]\!])$ cannot possibly be an $\mathcal{M}_{TF}$ process; for example, the failure $(\langle(1.5, a)\rangle, [0, 1) \times (\Sigma^{\checkmark} - \{a\})) \in \mathsf{Exp}(\mathcal{R}[\![P_1]\!])$ is $\prec$-maximal over the interval $[0, 1.5)$; however, the failure $(\langle(1, b)\rangle, [0, 1) \times (\Sigma^{\checkmark} - \{a\}))$ does *not* appear in $\mathsf{Exp}(\mathcal{R}[\![P_1]\!])$, violating Axiom *TF3*. Incidentally, note that, for any $P \in \mathcal{M}_R$, Axiom *TF3* is the only one that $\mathsf{Exp}_k(P)$ will possibly fail to satisfy.

The following program exemplifies a slightly different kind of timing unfaithfulness:

$$P_2 = a \xrightarrow{1} \bar{b}.$$

The timed failures of $\mathcal{F}_T[\![P_2]\!]$ clearly indicate that the time difference between the occurrences of a and b must be greater than or equal to one time unit. In contrast, the tests of $\mathcal{R}[\![P_2]\!]$ require that there should be at least one *tock* between the two events. An application of Exp therefore yields (among many others) the timed trace $\langle(0.99, a), (1, b)\rangle$, in which the time difference between a and b is almost nil! (This difference can in fact be 'made' as small as desired, but never zero as the definition of denotational expansion requires a to 'occur' strictly before the following *tock*.)

A slightly more dramatic demonstration of the effect of the timing inac-

[2] P_1 in effect implements the timeout process $\bar{a} \overset{1}{\rhd} STOP$, but the lengthier expression we have chosen for it better illustrates the situation at hand.

curacies described above is achieved with the program P_3 below:

$$P_3 = (a \xrightarrow{1} (\bar{a} \overset{0}{\rhd} STOP)) \mathbin{|||} (b \xrightarrow{1} (\bar{b} \overset{0}{\rhd} STOP)).$$

$\mathcal{F}_T[\![P_3]\!]$ consists of two interleaved processes, each of which can, at any time, communicate an event, and then communicate it again exactly one time unit later, or not at all. If we suppose that two a's and two b's are witnessed in a given timed failure of $\mathcal{F}_T[\![P_3]\!]$, with the first a occurring strictly before the first b, then the last event to be communicated will have to be a b.

However, it is easy to see that $\mathsf{Exp}(\mathcal{R}[\![P_3]\!])$ contains timed failures whose timed traces are, for example, $\langle(0,a),(0.9,b),(1,b),(1.9,a)\rangle$. Here the second a ought to have occurred 0.9 time units *before* the second b, rather than 0.9 time units *after* it! We do note, however, that $\mathcal{F}_T[\![P_3]\!]$ does contain timed failures whose untimed traces are $\langle a,b,b,a\rangle$; but for this to happen one must have the initial a and b occurring simultaneously.

One might think that the rather disconcerting state of affairs exposed in the previous few paragraphs should spell doom for any sustained or valuable use of $\mathcal{M}_R$ for accurate analysis of process behaviour in $\mathcal{M}_{TF}$. But it turns out that the discrepancies noted are themselves worst-case scenarios, and in general the 'timing fuzziness' that our discrete analysis necessarily entails does not accumulate and remains satisfactorily bounded. This statement is made precise in Theorem 6.6 below.

The examples above have exclusively focused on (timed) traces. We now show that refusals bring their own unruly twist to the picture. Here problems arise because $\mathcal{M}_R$ fails to offer a satisfactory treatment of instability—the transient state a process finds itself in while hidden events are on offer. This is remedied with the model $\mathcal{M}_{UR}$, presented in the next chapter.

The presence of instability is dealt with in $\mathcal{M}_R$ by sweeping all refusal information under the rug, only allowing $\bullet$ as refusals. As far as the definition of Exp went, this left us with a choice of two evils: either conservatively treat $\bullet$ as if it were the maximum refusal set, $\Sigma^{\checkmark}$, or (as we did) ignore $\bullet$ altogether and treat it as the empty refusal set. The first alternative would clearly have made any refusal information in the timed denotational expansion of a process useless, since $\bullet$, as per Axiom *R1*, is always an allowable refusal. Unfortunately, the path we have chosen is only slightly better, as we must now cope with the eventuality of under-refusing events.

As an example, let us once more consider $P_2 = a \xrightarrow{1} \bar{b}$. Suppose that an a is recorded at time 0.7 in a particular timed failure of $\mathcal{F}_T[\![P_2]\!]$. Then b can be validly refused over the interval $[0, 1.7)$, as some other timed failure will bear out. In the timed denotational expansion of $\mathcal{R}[\![P_2]\!]$, however, b will at best only be refused over the interval $[0, 1)$ (again assuming a was decreed to have happened at time 0.7). Afterwards, a silent ✓ is on offer, leaving a $\bullet$ refusal until such time as it occurs, at which point any event but b will be stably refused.

6.3 Timed Trace Analysis

We now present some general results about the correspondence between $\mathcal{M}_R$ and $\mathcal{M}_{TF}$, mostly concerning (timed) traces.

Proposition 6.1 *Let $P \in \overline{\mathbf{TCSP}}$, and let $e \in \mathsf{exec}_{TF}(P)$ be an integral execution. There exists an integral execution $e' \in \mathsf{exec}_{TF}(P)$ all of whose evolutions have unit duration, and such that, for any timed failure $(s, \aleph)$, e'* **fail** *$(s, \aleph)$ if and only if e* **fail** *$(s, \aleph)$.*

Although e' is technically not necessarily unique, we will pretend that it is (choosing a particular execution if necessary) and call it the *normalisation* of e. Any integral execution only comprising unit duration evolutions, such as e', is termed a *normal* execution.

Proof One first shows by structural induction on Q that, whenever $Q \overset{0}{\leadsto} Q'$, then $Q' \equiv Q$. It also follows from Proposition 3.5 that, whenever $Q \overset{n}{\leadsto} Q'$ (for $n \geqslant 1$), there is a unique sequence $\langle Q_i \rangle_{i=0}^{n}$ such that $Q_0 \equiv Q$, $Q_n \equiv Q'$, and $Q_0 \overset{1}{\leadsto} Q_1 \overset{1}{\leadsto} \ldots \overset{1}{\leadsto} Q_n$. Moreover, for all $0 \leqslant i \leqslant n$, $\mathsf{init}_{TF}(Q_i) = \mathsf{init}_{TF}(Q)$.

Given $e \in \mathsf{exec}_{TF}(P)$ as above, one then replaces every $Q \overset{0}{\leadsto} Q'$ in e by Q and every $Q \overset{n}{\leadsto} Q'$ in e by $Q_0 \overset{1}{\leadsto} Q_1 \overset{1}{\leadsto} \ldots \overset{1}{\leadsto} Q_n$. The resulting execution is clearly the required e'. ■

Proposition 6.2 *For any $P \in \overline{\mathbf{TCSP}}$, the set of normal executions of P (in the operational semantics associated with $\mathcal{M}_{TF}$) is in natural one-to-one correspondence with the set $\mathsf{exec}_R(P)$ of executions of P (in the operational semantics associated with $\mathcal{M}_R$).*

More precisely, this bijection takes a normal execution $e = P_0 \overset{z_1}{\longmapsto} P_1 \overset{z_2}{\longmapsto} \ldots \overset{z_n}{\longmapsto} P_n \in \mathsf{exec}_{TF}(P)$ to the execution $e' = P_0 \xrightarrow{x_1} P_1 \xrightarrow{x_2} \ldots \xrightarrow{x_n} P_n \in$

$\mathsf{exec}_R(P)$, *where for all* $1 \leqslant i \leqslant n$, $x_i = tock$ *when* $\xmapsto{z_i} = \overset{1}{\rightsquigarrow}$, *and* $x_i = \mu$ *when* $\xmapsto{z_i} = \xrightarrow{\mu}$.

In particular, $\mathsf{init}^{\tau}_{TF}(P) = \mathsf{init}^{\tau}_{R}(P)$.

Proof (Sketch.) This is a straightforward inspection which rests on the perfect bijective correspondence between the operational inference Rules 3.1–3.37 and 5.1–5.37. A formal proof proceeds by structural induction on P. ■

We next define an integral multiplication operation on terms: for $P \in \mathbf{TCSP}$ and $k \geqslant 1$, $kP \in \mathbf{TCSP}$ is identical to P except that every delay n in P has been replaced by a delay of kn (in all subprocesses $WAIT\ n$ and $P_1 \overset{n}{\rhd} P_2$) in kP. This operation can be given a straightforward inductive definition, which we omit. Note that integral multiplication takes programs to programs.

We can also define an integral multiplication operation on timed failures. If $(s, \aleph) = (\langle (t_i, a_i) \rangle_{i=1}^{n}, \bigcup_{i=1}^{m} \{[b_i, e_i) \times A_i\}) \in TF$ and $k \geqslant 1$, we let $k(s, \aleph) = (\langle (kt_i, a_i) \rangle_{i=1}^{n}, \bigcup_{i=1}^{m} \{[kb_i, ke_i) \times A_i\}) \in TF$. This definition extends to sets of timed failures in the obvious way.

Lemma 6.3 *For any* $P \in \overline{\mathbf{TCSP}}$ *and* $k \geqslant 1$, $\mathcal{F}_T[\![kP]\!] = k\mathcal{F}_T[\![P]\!]$.

Proof (Sketch.) One first extends integral multiplication to executions, as follows. If $e = P_0 \xmapsto{z_1} P_1 \xmapsto{z_2} \ldots \xmapsto{z_n} P_n \in \mathsf{exec}_{TF}(P)$, for some arbitrary program P, we let $ke = kP_0 \xmapsto{kz_1} kP_1 \xmapsto{kz_2} \ldots \xmapsto{kz_n} kP_n$, where each transition $kP_i \xmapsto{kz_{i+1}} kP_{i+1}$ in ke is $kP_i \xrightarrow{\mu_{i+1}} kP_{i+1}$ if $P_i \xmapsto{z_{i+1}} P_{i+1} = P_i \xrightarrow{\mu_{i+1}} P_{i+1}$, and is $kP_i \overset{kt_{i+1}}{\rightsquigarrow} kP_{i+1}$ if $P_i \xmapsto{z_{i+1}} P_{i+1} = P_i \overset{t_{i+1}}{\rightsquigarrow} P_{i+1}$. This operation naturally extends to sets of executions.

One then shows by structural induction that, for any term P, and any syntactic binding η with the property that $\mathsf{exec}_{TF}(k\eta(X)) = k\mathsf{exec}_{TF}(\eta(X))$ for all $X \in VAR$, the equality $\mathsf{exec}_{TF}(k(P\eta)) = k\mathsf{exec}_{TF}(P\eta)$ holds. All inductive cases are straightforward apart from recursion, which is handled using the indexed bisimulation techniques which figured prominently in the proofs of the digitisation lemma (Lemma 3.11) and Proposition 5.13 (where time-bisimulations, rather than indexed bisimulations, were used for the latter).

One then concludes that, for any $Q \in \overline{\mathbf{TCSP}}$, $\mathsf{init}_{TF}(Q) = \mathsf{init}_{TF}(kQ)$.

The result then follows immediately from Theorem 3.10. ■

We also have the following result:

Lemma 6.4 *For any $P \in \mathcal{M}_R$ and $k \geqslant 1$, $\mathsf{Exp}(P) = k\mathsf{Exp}_k(P)$.*

Proof (Sketch.) Follows immediately from the definitions. ■

We can now give:

Theorem 6.5 *For any $P \in \overline{\mathbf{TCSP}}$ and $k \geqslant 1$,*

$$\mathsf{Exp}_k(\mathcal{R}[\![kP]\!]) \sqsubseteq_{TT} \mathcal{F}_T[\![P]\!].$$

Proof We first observe that, due to Lemmas 6.3 and 6.4, it suffices to prove the result for $k = 1$.

Fix $P \in \overline{\mathbf{TCSP}}$, and let $s \in \mathsf{TTraces}(\mathcal{F}_T[\![P]\!])$. Pick an execution $e \in \mathsf{exec}_{TF}(P)$ such that e **fail** $(s, \aleph)$ for some refusal $\aleph$, as per Theorem 3.10. Let $e' = [e]_1 \in \mathsf{exec}_{TF}(P)$ be the lower digitisation of e as per the digitisation lemma (Lemma 3.11), and let e'' be the normalisation of e', as per Proposition 6.1. By Proposition 6.2, there is a canonical execution $o \in \mathsf{exec}_R(P)$ matching each of e'''s unit-duration evolutions with *tock*-transitions, and identical to e'' in all other respects. Let $u \in \mathcal{R}[\![P]\!]$ be such that o **test** u, as per Theorem 5.14. It is easy to verify, given u's construction and the definition of preExp, that $s \in \mathsf{TTraces}(\mathsf{preExp}(u))$, and therefore that $s \in \mathsf{TTraces}(\mathsf{Exp}(\mathcal{R}[\![P]\!]))$, completing the proof. ■

This theorem can be interpreted as a (timed trace) *soundness* result: if a (presumably undesirable) behaviour is ruled out by the denotational expansion of $\mathcal{R}[\![P]\!]$, then it is certainly not a possible behaviour of $\mathcal{F}_T[\![P]\!]$. (Timed trace) *completeness*, on the other hand, would be the assertion that the reverse refinement holds, i.e., that the *only* timed traces absent from the denotational expansion of $\mathcal{R}[\![P]\!]$ are precisely those which $\mathcal{F}_T[\![P]\!]$ cannot perform. But as we have seen in the previous section, this is something we have to forgo. Nonetheless, we can get a good approximation to (timed trace) completeness, which we present below.

We first need to define a metric td on the space TT of timed traces. Let $s = \langle (t_1, a_1), (t_2, a_2), \ldots, (t_n, a_n) \rangle$ and $s' = \langle (t'_1, a_1), (t'_2, a_2), \ldots, (t'_n, a_n) \rangle$ be timed traces. We define the *timed distance* $\mathsf{td}(s, s')$ between them as follows:

$\mathsf{td}(s, s') \mathrel{\widehat{=}} \max\{(|t_i' - t_i|) \mid 1 \leqslant i \leqslant n\}$. In all other situations (namely, when the untimed versions of s and s' do not agree), we set $\mathsf{td}(s, s') = \infty$. td clearly makes TT into a (complete) metric space.

Theorem 6.6 *For any $P \in \overline{\mathbf{TCSP}}$, $k \geqslant 1$, and $s \in \mathsf{TTraces}(\mathsf{Exp}_k(\mathcal{R}[\![kP]\!]))$, there exists $s' \in \mathsf{TTraces}(\mathcal{F}_T[\![P]\!])$ such that $\mathsf{td}(s', s) < 1/k$. In fact, s' can be chosen so that each of its events is recorded earlier than or at the same time as its* vis-à-vis *in s.*

Proof Here again it follows easily from Lemmas 6.4 and 6.3 that it suffices to prove the result in the case $k = 1$.

So let $s \in \mathsf{TTraces}(\mathsf{Exp}(\mathcal{R}[\![P]\!]))$, and let u be a test of $\mathcal{R}[\![P]\!]$ yielding $(s, \aleph)$ (for some refusal $\aleph$) under preExp. Theorem 5.14 assures us there is an execution $e \in \mathsf{exec}_R(P)$ such that e **test** u, and Proposition 6.2 then informs us that there is an execution $e' \in \mathsf{exec}_{TF}(P)$ identical to e but for having unit-duration evolutions wherever e had *tock*-transitions. By Theorem 3.10, there is $(s', \aleph') \in \mathcal{F}_T[\![P]\!]$ such that e **fail** $(s', \aleph')$.

Now, by definition, preExp decrees that the time of occurrence of any event in a given test must lie in a unit interval consistent with the number of *tock*s witnessed prior to the event. Clearly, then, the timing of corresponding events in $\mathsf{preExp}(u)$ and s' cannot differ by more than one time unit. We observe that the inequality is strict since events in $\mathsf{preExp}(u)$ must by definition 'happen' strictly before the time postulated for the occurrence of the following *tock*. ■

Note that in the case $k = 1$, each event in s', as constructed in the proof, happens at the greatest integral time less than or equal to the time of occurrence of its *vis-à-vis* in s; in other words, s' is the lower digitisation of s.

Theorem 6.6 is a (timed trace) *quasi-completeness* result, as it gives us sharp bounds regarding the extent to which timed denotational expansions can stray from the $\mathcal{M}_{TF}$-computed timed traces of a given $\overline{\mathbf{TCSP}}$ program P. Together with Theorem 6.5, it also shows that, as k increases, $\mathsf{Exp}_k(\mathcal{R}[\![kP]\!])$ 'converges' towards $\mathcal{F}_T[\![P]\!]$ (at least as far as timed traces are concerned), enabling one's analysis of $\mathcal{F}_T[\![P]\!]$ to become arbitrarily 'precise', albeit potentially at an expense in computational complexity.

We point out, perhaps surprisingly, that $k \leqslant k'$ does *not* necessarily entail that $\mathsf{Exp}_k(\mathcal{R}[\![kP]\!]) \sqsubseteq_{TT} \mathsf{Exp}_{k'}(\mathcal{R}[\![k'P]\!])$ for an arbitrary program P.

For example, let $P = a \longrightarrow (\bar{b} \overset{1}{\rhd} STOP)$, and compare $\mathcal{R}[\![2P]\!]$ and $\mathcal{R}[\![3P]\!]$ (i.e., $k = 2$ and $k' = 3$). If the original time units are minutes, then two consecutive *tock*s by the first process are meant to delineate a $30s$ interval, whereas two consecutive *tock*s by the second process signify that $20s$ have elapsed between them. On the trace $\langle(25s, a)\rangle$, therefore, $E_2 = \mathsf{Exp}_2(\mathcal{R}[\![2P]\!])$ will 'simulate' a *tock*, after a, ahead of $E_3 = \mathsf{Exp}_3(\mathcal{R}[\![3P]\!])$. Because of this, the latest possible occurrence of b in E_2 will arrive sooner than in E_3. And indeed, a simple inspection shows that the traces $\langle(25s, a), (t, b)\rangle$, with $90s \leqslant t < 100s$, are all possible for E_3, but not for E_2. Hence it is not the case that $E_2 \sqsubseteq_{TT} E_3$.

Of course, under certain conditions the expected relationship will hold:

Proposition 6.7 *Let* $P \in \overline{\mathbf{TCSP}}$*, and let* $1 \leqslant k \leqslant k'$ *be such that* k *divides* k'*. Then* $\mathsf{Exp}_k(\mathcal{R}[\![kP]\!]) \sqsubseteq_{TT} \mathsf{Exp}_{k'}(\mathcal{R}[\![k'P]\!])$.

Proof (Sketch.) It clearly suffices to prove the result in the case $k = 1$, thanks to Lemma 6.4. Let $s \in \mathsf{TTraces}(\mathsf{Exp}_{k'}(\mathcal{R}[\![k'P]\!]))$. We must show that $s \in \mathsf{TTraces}(\mathsf{Exp}(\mathcal{R}[\![P]\!]))$. Let $u \in \mathcal{R}[\![k'P]\!]$ with $s \in \mathsf{TTraces}(\mathsf{preExp}_{k'}(u))$. By Theorem 5.14, there is $e \in \mathsf{exec}_R(k'P)$ such that e **test** u. Proposition 6.2 then provides a corresponding normal execution $e' \in \mathsf{exec}_{TF}(k'P)$. As in the proof of Lemma 6.3, we then therefore have that $e'/k' \in \mathsf{exec}_{TF}(P)$. Let $o \in \mathsf{exec}_{TF}(P)$ be the lower digitisation of e'/k', as per the digitisation lemma, and let $o' \in \mathsf{exec}_{TF}(P)$ be the normalisation of o, as per Proposition 6.1. We can then invoke Proposition 6.2 again to obtain $o'' \in \mathsf{exec}_R(P)$, which then yields $v \in \mathcal{R}[\![P]\!]$, with o'' **test** v, by Theorem 5.14. It is then not difficult to trace through this sequence of transformations and verify that one has $s \in \mathsf{TTraces}(\mathsf{preExp}(v))$, and therefore that $s \in \mathsf{TTraces}(\mathsf{Exp}(\mathcal{R}[\![P]\!]))$, as required. ■

It is interesting to note how the above proposition, which is stated entirely in terms of the discrete-time refusal testing model $\mathcal{M}_R$, is established using results, such as the digitisation lemma, which are themselves stated purely in terms of the timed failures model $\mathcal{M}_{TF}$.

6.4 Applications to Verification

We show how the results of the previous section can be exploited to reduce proof obligations in $\mathcal{M}_{TF}$ to proof obligations in $\mathcal{M}_R$.

The most natural instance concerns behavioural specifications. Specifically, if S is a behavioural timed trace specification on $\mathcal{M}_{TF}$ processes, Theorem 6.5 guarantees that, for any $P \in \overline{\mathbf{TCSP}}$ and any $k \geqslant 1$, if $\mathsf{Exp}_k(\mathcal{R}[\![kP]\!]) \vDash S$, then $\mathcal{F}_T[\![P]\!] \vDash S$ as well. The definition of Exp_k, on the other hand, makes it simple to manufacture a conservative specification $\mathbb{C}(S)$, derived from S, such that if $\mathcal{R}[\![kP]\!] \vDash \mathbb{C}(S)$, then $\mathsf{Exp}_k(\mathcal{R}[\![kP]\!]) \vDash S$.

As an example, let us consider the following timed trace specification S_1: 'Whenever the event b is observed, it must have been preceded at most 10 time units earlier by the event a'. This is a behavioural timed trace specification because it is a predicate required to hold for all the timed traces of a process.

A $\overline{\mathbf{TCSP}}$ program P will satisfy S_1 if $\mathcal{R}[\![P]\!]$ satisfies the test specification $\mathbb{C}(S_1)$ that reads: 'Whenever the event b is observed, it must have been preceded by the event a, with at most 9 *tock*s occurring between them.' Note how the time constraint was converted into a requirement on the number of *tock*s; it is in fact not hard to see intuitively that there is a systematic way to perform such conservative conversions (although making this precise would require us to set up a formal specification formalism). The drawback, on the other hand, is that there may be some programs P such that $\mathcal{F}_T[\![P]\!] \vDash S_1$ but $\mathcal{R}[\![P]\!] \nvDash \mathbb{C}(S_1)$.

In the case at hand, for example, a sharper analysis reveals that it is sufficient to consider the weaker test specification $\mathbb{Z}_R(S_1)$ that reads: 'Whenever the event b is observed, it must have been preceded by the event a, with at most *10 tock*s occurring between them.' Indeed, suppose that $\mathcal{F}_T[\![P]\!] \nvDash S_1$ for some program P. Then there is $s \in \mathsf{TTraces}(\mathcal{F}_T[\![P]\!])$ with $\langle (t_b, b)\rangle$ in s, and such that whenever $\langle (t_a, a)\rangle$ in s with $t_a \leqslant t_b$, $t_b - t_a > 10$. Let us assume that there is at least one such instance $\langle (t_a, a)\rangle$ in s (in which case we pick the one with largest t_a value), otherwise we trivially have $\mathcal{R}[\![P]\!] \nvDash \mathbb{Z}_R(S_1)$. If $\dot{t_b}$ denotes the fractional part of t_b, we can apply the digitisation lemma to (any execution yielding) the timed trace s, with pivot $\dot{t_b}$, to obtain (an integral execution yielding) the timed trace $[s]_{\dot{t_b}} \in \mathsf{TTraces}(\mathcal{F}_T[\![P]\!])$. It is then clear that the corresponding a and b events in $[s]_{\dot{t_b}}$ occur respectively at times $\lfloor t_a \rfloor$ and $\lceil t_b \rceil$, i.e., at least *11* time units apart. We can then invoke Proposition 6.2 to obtain a test $u \in \mathsf{exec}_R(P)$ with at least 11 *tock*s occurring between these very a and b events. In other words, $\mathcal{R}[\![P]\!] \nvDash \mathbb{Z}_R(S_1)$.

Since, on the other hand, it is clear (again thanks to Proposition 6.2) that for all $P \in \overline{\mathbf{TCSP}}$, $\mathcal{F}_T[\![P]\!] \vDash S_1$ implies that $\mathcal{R}[\![P]\!] \vDash \mathbb{Z}_R(S_1)$, we find ourselves in the highly desirable situation of having S_1-satisfiability *equivalent* to $\mathbb{Z}_R(S_1)$-satisfiability.

This state of affairs is a particular instance of Theorem 6.10, which we give below after introducing the necessary terminology.

For T a set of timed traces, let $\mathbb{Z}(T)$ denote the subset of *integral* timed traces of T, in other words those timed traces in T all of whose events occur at integral times. For $0 \leqslant \varepsilon \leqslant 1$, the operator $[\cdot]_\varepsilon$, defined in Section 3.3, naturally extends to timed traces (as was implicit in our use of the notation $[s]_{\dot{t_b}}$ above), by pointwise application to the time component of each of the trace's elements. We can therefore extend $[\cdot]_\varepsilon$ further by applying it to sets of timed traces. We then say that the set T is *closed under digitisation* if, for any $0 \leqslant \varepsilon \leqslant 1$, $[T]_\varepsilon \subseteq T$. On the other hand, we say that T is *closed under inverse digitisation* if, whenever a timed trace s is such that $[s]_\varepsilon \in T$ for all $0 \leqslant \varepsilon \leqslant 1$, then $s \in T$.

The following result is due to Henzinger, Manna, and Pnueli [HMP92]:

Theorem 6.8 *If $P \subseteq TT$ is closed under digitisation and $S \subseteq TT$ is closed under inverse digitisation, then*

$$\mathbb{Z}(P) \subseteq \mathbb{Z}(S) \Leftrightarrow P \subseteq S.$$

The next result is a direct consequence of the digitisation lemma:

Theorem 6.9 *For any program $P \in \overline{\mathbf{TCSP}}$, $\mathsf{TTraces}(\mathcal{F}_T[\![P]\!])$ is closed under digitisation.*

For any set $S \subseteq TT$ of timed traces, it is easy to see that one can always find a set of tests $\mathbb{Z}_R(S) \subseteq TEST$ having the property

$$\mathbb{Z}(S) = \mathbb{Z}(\mathsf{TTraces}(\mathsf{Exp}(\mathbb{Z}_R(S)))).$$

(There will always be several choices for $\mathbb{Z}_R(S)$; let us pick one for each $S \subseteq TT$, and thus think of $\mathbb{Z}_R$ as a function from $\mathcal{P}(TT)$ to $\mathcal{P}(TEST)$.)

Recall that any behavioural timed trace specification can be identified with a set of timed traces: for S such a specification and P a $\overline{\mathbf{TCSP}}$ program, $P \vDash S$ if and only if $\mathsf{TTraces}(\mathcal{F}_T[\![P]\!]) \subseteq S$. Given a timed trace specification S, let us therefore also write $\mathbb{Z}_R(S)$ to denote the corresponding behavioural test specification. We now give our main (timed trace) result:

Theorem 6.10 *Let $S = S(P)$ be a behavioural timed trace specification closed under inverse digitisation. Then, for any program $P \in \overline{\mathbf{TCSP}}$,*

$$\mathcal{R}[\![P]\!] \vDash \mathbb{Z}_R(S) \Leftrightarrow \mathcal{F}_T[\![P]\!] \vDash S.$$

Proof (Sketch.) Let S be as above. By Theorem 6.8, we have $\mathcal{F}_T[\![P]\!] \models S \Leftrightarrow \mathbb{Z}(\mathsf{TTraces}(\mathcal{F}_T[\![P]\!])) \models \mathbb{Z}(S)$. Suppose that $\mathcal{R}[\![P]\!] \models \mathbb{Z}_R(S)$, and let $s \in \mathbb{Z}(\mathsf{TTraces}(\mathcal{F}_T[\![P]\!]))$ be an integral trace of P. We aim to show that $s \in \mathbb{Z}(S)$. It is easy to show (by structural induction on P) that there is some integral execution $e \in \mathsf{exec}_{TF}(P)$ such that e **fail** $(s, \emptyset)$. We can then invoke Propositions 6.1 and 6.2 to obtain a corresponding execution $o \in \mathsf{exec}_R(P)$ such that o **test** u, where $u \in \mathcal{R}[\![P]\!]$ is a test of P with the property that $s \in \mathsf{TTraces}(\mathsf{preExp}(u))$. Since $\mathcal{R}[\![P]\!] \models \mathbb{Z}_R(S)$, we have $u \in \mathbb{Z}_R(S)$, and therefore $s \in \mathbb{Z}(S)$ by definition of $\mathbb{Z}_R(S)$, remembering that s is an integral timed trace. This establishes the left-to-right implication.

The procedure which yielded u from s can equally be carried out in the reverse direction—we leave the details to the reader. This takes care of the right-to-left implication and completes the proof. ■

Note that the behavioural timed trace specification S_1 defined earlier is closed under inverse digitisation; an application of Theorem 6.10 therefore immediately yields our earlier conclusion that S_1-satisfiability is equivalent to $\mathbb{Z}_R(S_1)$-satisfiability.

Theorem 6.10 yields the following central corollary:

Corollary 6.11 *Let* $Q \in \overline{\mathbf{TCSP}}$ *be a program whose set of timed traces* $\mathsf{TTraces}(\mathcal{F}_T[\![Q]\!])$ *is closed under inverse digitisation. Then, for any program* $P \in \overline{\mathbf{TCSP}}$,

$$\mathcal{R}[\![Q]\!] \sqsubseteq_T \mathcal{R}[\![P]\!] \Leftrightarrow \mathcal{F}_T[\![Q]\!] \sqsubseteq_{TT} \mathcal{F}_T[\![P]\!].$$

Proof (Sketch.) This result rests on the fact that $\mathcal{R}[\![Q]\!]$ satisfies the crucial property $\mathbb{Z}(\mathsf{TTraces}(\mathcal{F}_T[\![Q]\!])) = \mathbb{Z}(\mathsf{TTraces}(\mathsf{Exp}(\mathcal{R}[\![Q]\!])))$, which can be shown using the line of reasoning offered in the proof of Theorem 6.10. The desired conclusion then follows from Theorem 6.10. ■

An interesting fact is that closure under inverse digitisation *characterises* those (downward-closed) behavioural timed trace specifications[3] that obey the conclusion of Theorem 6.10, as the following proposition indicates:

Proposition 6.12 *Let* $S = S(P)$ *be a downward-closed behavioural timed trace specification which is* not *closed under inverse digitisation. Then there is a program* $P \in \overline{\mathbf{TCSP}}$ *such that* $\mathcal{R}[\![P]\!] \models \mathbb{Z}_R(S)$ *but* $\mathcal{F}_T[\![P]\!] \not\models S$.

[3]It makes little sense for behavioural specifications *not* to be downward-closed, since processes themselves are always downward-closed.

Proof (Sketch.) Let the specification S be as above. There must then be a trace $tr = \langle (t_1, a_1), (t_2, a_2), \ldots, (t_n, a_n) \rangle \notin S$ such that, for all $0 \leqslant \varepsilon \leqslant 1$, $[tr]_\varepsilon \in S$. We now construct a program P with the following property: any digitisation of a trace of $\mathcal{F}_T[\![P]\!]$ is equal to some digitisation of a prefix of tr, and moreover $\mathcal{F}_T[\![P]\!]$ contains the trace tr.

The idea is fairly simple: we build P in stages, by specifying increasingly stronger restrictions on the allowable times of occurrence of the successive events of tr. For any given index $1 \leqslant j \leqslant n$, the allowable times of occurrence of a_j depend on the occurrence times of all events preceding it. These restrictions are captured by processes B_i^j, each of which subordinates the allowable times of occurrence of a_j to the time of occurrence of a_i. (We allow i to be 0 to also subordinate the occurence of a_j to an imaginary event a_0 occurring at time 0.) Those time bounds are then combined by simply putting the B_i^j's in parallel.

To illustrate the idea, consider the allowable times of the first event, a_1. If a_1 has occurred at some integral time (i.e., if $t_1 \in \mathbb{N}$), then the only allowable time for a_1 is t_1. Otherwise, a_1 should be free to occur at any time t_1' such that $\lfloor t_1 \rfloor \leqslant t_1' \leqslant \lceil t_1 \rceil$: any digitisation of t_1' is a digitisation of t_1 (although not necessarily involving the same pivot ε). This restriction is captured by the process B_0^1.

Consider now the allowable times of occurrence of the second event, a_2. Like a_1, it must be constrained by some B_0^2 specifying its 'absolute' allowable times of occurrence. Moreover, it must also be constrained with respect to t_1' (a_1's time of occurrence). For instance, if the difference between t_2 and t_1 (in tr) is integral, then any digitisation of tr must have a_1 and a_2 occurring exactly the same number of time units apart. In case $t_2 - t_1$ is not integral, a_2 should be allowed to occur at any time t_2' such that $t_2' - t_1'$ lies in the same integral unit interval as $t_2 - t_1$. This second restriction, captured by the process B_1^2, ensures that any uniform digitisation of t_2' and t_1' yields the same difference as some uniform digitisation of t_2 and t_1. An appropriate parallel combination of B_0^2 and B_1^2 then gives us exactly the required restrictions on a_2.

This procedure can be repeated for all the events of tr. The resulting compound process is the required P.

We now give the details of the construction. We first define processes B_i^j, for $0 \leqslant i < j \leqslant n$, as follows:

$$\begin{array}{rcll} B_0^j & = & WAIT\ \lfloor t_j \rfloor\ ; & \\ & & \overline{a_j} \overset{0}{\rhd} STOP \mathrel{\lessdot} t_j \in \mathbb{N} \mathrel{\gtrdot} \overline{a_j} \overset{1}{\rhd} STOP & \text{for } 1 \leqslant j \leqslant n \\ B_i^j & = & a_i \longrightarrow WAIT\ \lfloor t_j - t_i \rfloor\ ; & \\ & & \overline{a_j} \overset{0}{\rhd} STOP \mathrel{\lessdot} t_j - t_i \in \mathbb{N} \mathrel{\gtrdot} \overline{a_j} \overset{1}{\rhd} STOP & \text{for } 1 \leqslant i < j \leqslant n. \end{array}$$

We now recursively define processes P_k, for $1 \leqslant k \leqslant n$, as follows:

$$\begin{array}{rcll} P_n & = & \underset{\{a_n\}}{\Big|\Big|} \{B_i^n \mid 0 \leqslant i \leqslant n-1\} & \\ P_k & = & \underset{\{a_k\}}{\Big|\Big|} \{B_i^k \mid 0 \leqslant i \leqslant k-1\} \underset{\{a_1, a_2, \ldots, a_k\}}{\|} P_{k+1} & \text{for } 1 \leqslant k \leqslant n-1. \end{array}$$

(Note that the induction starts at n and moves downward to 1.)

Finally, we simply let $P = P_1$.

The reader may wish to verify the following: $tr \in \mathsf{TTraces}(\mathcal{F}_T[\![P]\!])$, and $\mathbb{Z}(\mathsf{TTraces}(\mathcal{F}_T[\![P]\!])) = \{[u]_\varepsilon \mid u \leq tr \wedge 0 \leqslant \varepsilon \leqslant 1\}$.

Since S is downward closed, it follows that $\mathcal{R}[\![P]\!] \vDash \mathbb{Z}_R(S)$. However, $\mathcal{F}_T[\![P]\!] \nvDash S$ since $tr \in \mathsf{TTraces}(\mathcal{F}_T[\![P]\!])$, which completes the proof. ∎

Proposition 6.12 tells us that, insofar as verifying exclusively those (behavioural and downward-closed) timed trace specifications that are closed under inverse digitisation, the trace component of representations of processes in $\mathcal{M}_R$ is as abstract as can be. In other words, no redundant or irrelevant information is encapsulated within the discrete traces of processes. Together with Theorem 6.10, this constitutes a (timed trace) *full abstraction* result (with respect to the verification of specifications closed under inverse digitisation). We will see shortly (cf. Proposition 6.18) that $\mathcal{M}_R$ is unfortunately not fully abstract in this sense when it comes to timed failures, but discuss in Chapter 9 a proposed alternative to $\mathcal{M}_R$ which would be. One of the chief practical benefits of full abstraction is that, by having as simple, or abstract, a model as possible, model checking algorithms are correspondingly more efficient.

One may ask, on the other hand, why one should *a priori* choose the criterion of closure under inverse digitisation to distinguish the specifications that one is interested in verifying. This is certainly a good question, since it is possible to verify specifications which are *not* closed under inverse digitisation, as we demonstrate below.[4] Closure under inverse digitisation does appear to be the 'right' level to aim for when it comes to techniques involving discretisation, but the question certainly warrants further investigation. We shall return to this matter later on.

In view of Theorem 6.10 and Corollary 6.11, it is clear that it would be highly desirable to produce a procedure for deciding when a specification is closed under inverse digitisation. Henzinger *et al.* offer partial results to that effect in [HMP92], indentifying a large class of behavioural timed trace specifications which are closed under inverse digitisation. These include so-called *qualitative properties*, which are specifications not incorporating timing information, as well as *bounded-response* and *bounded-invariance properties.* Moreover, this class is closed under arbitrary intersections (corresponding to logical conjunction). Lastly, if a set $P \subseteq TT$ is closed under digitisation, then its complement $TT - P$ is itself closed under inverse digitisation. Thanks to Theorem 6.10 and Corollary 6.11, we are therefore able to verify *exactly* a large class of behavioural timed trace specifications over the $\mathcal{M}_{TF}$ representations of $\overline{\mathbf{TCSP}}$ programs.

This technique does have certain limitations. Not only does it apply exclusively to *behavioural* specifications, but it does not, in general, apply to behavioural specifications $S(P)$ of the form $Q \sqsubseteq_{TT} P$ (for some fixed program Q), since these are *not* necessarily closed under inverse digitisation, as our next example shows. In addition, by restricting themselves to a timed trace semantics, Henzinger *et al.* are unable to model nondeterminism and handle liveness issues.[5] As we shall see in Section 6.5 and also in Chapters 7 and 8, the inclusion of refusal information complicates the analysis, although we are able to offer satisfactory extensions of Theorems 6.8, 6.9, and 6.10, as well as Corollary 6.11.

We now turn our attention to a second example. Consider the following behavioural timed trace specification S_2: 'The only event that may be witnessed is a single communication of a, which may only occur precisely at time n, for $n \in \mathbb{N}$'. Note that this specification (on P) can be expressed as the refinement $Q \sqsubseteq_{TT} P$, where $Q = \bar{a} \overset{0}{\rhd} WAIT\ 1\ ;\ Q$.[6] The conservative trans-

[4]In fact, *region graphs* techniques [ACD90, ACD93] allow one to verify much wider-ranging specifications than simply those closed under inverse digitisation, albeit at a cost in complexity. This will be discussed in Chapter 9.

[5]This being said, the *timed transition systems* model featured in [HMP92] allows them to express certain properties, such as *bounded response*, which could only be formulated via liveness in the timed failures model.

[6]We remark that the fact that S_2 only allows an a in *transient* form is not essential

lation of S_2 as a test specification yields the specification $\mathbb{C}(S_2) = \mathbf{false}$—not something very useful! However, other trace-based test specifications on $\mathcal{R}[\![P]\!]$, such as $\mathbb{Z}_R(S_2) : \mathcal{R}[\![Q]\!] \sqsubseteq_T \mathcal{R}[\![P]\!]$, hardly seem better: in this case we have, for example, $\mathcal{R}[\![\bar{a}]\!] \vDash \mathbb{Z}_R(S_2)$ but $\mathcal{F}_T[\![\bar{a}]\!] \nvDash S_2$.

Naturally, this unfortunate situation has arisen because S_2 is not closed under inverse digitisation, so that Theorem 6.10 does not apply. We do however have that $\mathcal{R}[\![P]\!] \nvDash \mathbb{Z}_R(S_2) \Rightarrow \mathcal{F}_T[\![P]\!] \nvDash S_2$, for any $P \in \overline{\mathbf{TCSP}}$, by a direct application of Proposition 6.2.[7] As discussed earlier, the specification $\mathbb{Z}_R(S_2)$ is equivalent to a requirement on the integral traces of P: $\mathbb{Z}(\mathsf{TTraces}(\mathcal{F}_T[\![P]\!])) \subseteq \mathbb{Z}(S_2)$. A natural course of action is therefore to strengthen this specification into a similar requirement on all the 'half-integral' traces of P, namely those timed traces all of whose events occur at times that are integral multiples of $1/2$. Translated as a requirement on tests, this specification becomes $\mathbb{Z}^2_R(S_2) = (\mathbb{Z}^2_R(S_2))(P) : \mathcal{R}[\![2Q]\!] \sqsubseteq_T \mathcal{R}[\![2P]\!]$.

It is an easy consequence of Lemma 6.3 that the implication $\mathcal{R}[\![P]\!] \nvDash \mathbb{Z}^2_R(S_2) \Rightarrow \mathcal{F}_T[\![P]\!] \nvDash S_2$, for any $P \in \overline{\mathbf{TCSP}}$, continues to hold. Naturally, $\mathbb{Z}^2_R(S_2)$-satisfiability, on the other hand, entails $\mathbb{Z}_R(S_2)$-satisfiability. But the pleasant surprise is that in fact $\mathbb{Z}^2_R(S_2)$-satisfiability entails S_2-satisfiability; in other words, for any $P \in \overline{\mathbf{TCSP}}$, $\mathcal{F}_T[\![P]\!] \vDash S_2$ if and only if $\mathcal{R}[\![P]\!] \vDash \mathbb{Z}^2_R(S_2)$. (This follows from the fact that $\mathsf{TTraces}(\mathcal{F}_T[\![2Q]\!])$ is closed under inverse digitisation, as the reader may wish to verify.) This state of affairs, it turns out, does not always hold. In general, given a behavioural timed trace specification S, one can manufacture increasingly stronger test specifications of the form $\mathbb{Z}^k_R(S)$, each of which is (not necessarily strictly) weaker than S. One can in fact construct examples where the desired S- and $\mathbb{Z}^k_R(S)$-equivalence occurs arbitrarily late, or even not at all. In addition, in the worst case the computational complexity of model checking a program against a specification of the form $\mathbb{Z}^k_R(S)$ will increase exponentially as a function of k. This technique is therefore likely to be useful only in a restricted number of situations.

Returning to our specification S_2, it is interesting to see that there exists a behavioural test specification on $\mathcal{R}[\![P]\!]$ which is equivalent to S_2: let $\mathbf{Z}_R(S_2) = (\mathbf{Z}_R(S_2))(P)$ be the specification $\mathcal{R}[\![Q]\!] \sqsubseteq_R \mathcal{R}[\![P]\!]$, which differs from $\mathbb{Z}_R(S_2)$ in that we require full test refinement instead of mere trace refinement. (Here again $Q = (\bar{a} \overset{0}{\rhd} \mathit{WAIT}\ 1) \ ; Q)$. Rather surprisingly, we have, for any $P \in \overline{\mathbf{TCSP}}$, that $P \vDash \mathbf{Z}_R(S_2) \Rightarrow P \vDash S_2$. (The converse also

to our argument; the same effect would be achieved by setting the allowable occurrence intervals for a to be $[2n, 2n+1]$ for $n \in \mathbb{N}$.

[7]This implication holds for *any* behavioural timed trace specification S_2.

holds, but our interest primarily lies in the left-to-right implication.) Note that $P = \bar{a}$, which served as counterexample in the case of $\mathbb{Z}_R(S_2)$, is not a problem here: $\langle \emptyset, a, \bullet \rangle \in \mathcal{R}[\![\bar{a}]\!]$, whereas $\langle \emptyset, a, \bullet \rangle \notin \mathcal{R}[\![Q]\!]$ (although, of course, $\langle \bullet, a, \bullet \rangle \in \mathcal{R}[\![Q]\!]$). In other words, any a that Q communicates necessarily originates from an unstable state, which is not the case for $\bar{a}$. The refinement $\mathcal{R}[\![Q]\!] \sqsubseteq_R \mathcal{R}[\![\bar{a}]\!]$ therefore does *not* hold, i.e., $\bar{a} \nvDash \mathbf{Z}_R(S_2)$.

To see that $P \vDash \mathbf{Z}_R(S_2) \Rightarrow P \vDash S_2$, assume the contrary, and consider a program $P \in \overline{\mathbf{TCSP}}$ which violates the implication. It is not hard to see that any such violation must have $(\langle (t, a) \rangle, \aleph) \in \mathcal{F}_T[\![P]\!]$, for some $t \notin \mathbb{N}$ and some refusal $\aleph$. Let $e \in \mathsf{exec}_{TF}(P)$ be such that e **fail** $(\langle (t, a) \rangle, \aleph)$. e must consist in random evolutions interspersed with τ-transitions, followed by an a-transition, and then possibly further behaviour which we can freely ignore. One first shows (by structural induction on P) that all τ-transitions in e occurring prior to the a-transition must 'take place' at integral times. It follows that the a-transition must have been immediately preceded by an evolution, which we can clearly assume to have non-zero duration. We can then invoke the digitisation lemma to stretch this evolution into one having positive and integral duration. The corresponding normalised execution $e' \in \mathsf{exec}_{TF}(P)$ is therefore of the form $e' = o^\frown(P_{n-2} \overset{1}{\leadsto} P_{n-1} \xrightarrow{a} P_n)$. Since $P_{n-2} \overset{1}{\leadsto}$, maximal progress implies that $P_{n-2} \stackrel{\tau}{\not\longrightarrow}$. On the other hand, persistency shows that $a \in \mathsf{init}_{TF}(P_{n-2})$. Since we know that $\mathsf{init}^\tau_R(P_{n-2}) = \mathsf{init}^\tau_{TF}(P_{n-2})$, one has an execution $e'' = o'^\frown(P_{n-2} \xrightarrow{a} P') \in \mathsf{exec}_R(P)$. Since $P_{n-2} \stackrel{\tau}{\not\longrightarrow}$, i.e., since P_{n-2} is stable, we have e'' **test** $\hat{u}^\frown\langle \emptyset, a, \bullet \rangle$ (for some test u whose trace consists of a sequence of *tock* events.) Denoting this test by v, we conclude that $v \in \mathcal{R}[\![P]\!]$, whereas, for reasons stated earlier, we cannot have $v \in \mathcal{R}[\![Q]\!]$. Hence $\mathcal{R}[\![Q]\!] \not\sqsubseteq_R \mathcal{R}[\![P]\!]$, i.e., $P \nvDash \mathbf{Z}_R(S_2)$, the required contradiction.

We have shown that $\mathcal{R}[\![Q]\!] \sqsubseteq_R \mathcal{R}[\![P]\!]$ implies that $\mathcal{F}_T[\![Q]\!] \sqsubseteq_{TT} \mathcal{F}_T[\![P]\!]$. Since $\mathcal{F}_T[\![Q]\!]$ can always refuse any set of events over any time period, we in fact have the stronger implication $\mathcal{R}[\![Q]\!] \sqsubseteq_R \mathcal{R}[\![P]\!] \Rightarrow \mathcal{F}_T[\![Q]\!] \sqsubseteq_{TF} \mathcal{F}_T[\![P]\!]$. Both implications hold with $Q = (\bar{a} \overset{0}{\rhd} WAIT\ 1)\ ;\ Q$ and any $P \in \overline{\mathbf{TCSP}}$. From this example and others, one could be led to conjecture that these implications actually hold for *any* pair of programs P and Q. As Proposition 6.19 of Section 6.6 shows, this is however unfortunately not the case.

The previous paragraph raises an interesting question: can one devise a discrete model, obviously less abstract than $\mathcal{M}_R$, which would unconditionally yield the above implications? We believe that no discretisation-based framework could achieve this, a view partially supported by Propositions 6.12 and 6.22. We shall return to this question in Chapter 9.

We conclude this section by mentioning that certain non-behavioural timed trace specifications can also be handled within our framework. A very important class of non-behavioural specifications, in particular, consists in 'reachability' assertions such as S_3: 'It is possible to witness the event a'. S_3 can clearly be verified exactly by examining whether the $\mathcal{M}_R$ representation of a given process satisfies it or not. In cases where the specification contains specific timing information, a conservative conversion may be necessary.

6.5 Timed Failure Analysis

We now turn to the question of the relationship between refusal information in $\mathcal{M}_R$ and refusal information in $\mathcal{M}_{TF}$.

The observations we have made in Section 6.2 concerning the process $P_2 = a \xrightarrow{1} \bar{b}$ clearly show that, for any $k \geqslant 1$, the sets $\mathsf{Exp}_k(\mathcal{R}[\![kP]\!])$ and $\mathcal{F}_T[\![P]\!]$ are in general $\sqsubseteq_{TF}$-incomparable. Consequently, no soundness result extending Theorem 6.5 can possibly exist. As discussed in Section 6.2, this is because $\mathcal{M}_R$ does not offer a satisfactory treatment of stability: while a process is in an unstable state, it is difficult to extract precise and reliable refusal information from its representation in $\mathcal{M}_R$. This problem is remedied with the introduction of the discrete-time unstable refusal testing model $\mathcal{M}_{UR}$ in the next chapter, which then provides us with the timed failure soundness result sought.

Surprisingly, we are nonetheless able to produce a quasi-completeness result extending Theorem 6.6. We first extend the metric td to timed failures, by letting $\mathsf{td}((s,\aleph),(s',\aleph')) \mathrel{\widehat{=}} \max\{\mathsf{td}(s,s'),\mathsf{d}_H(\aleph,\aleph')\}$, for $(s,\aleph),(s',\aleph') \in TF$; $\mathsf{d}_H(\aleph,\aleph') \mathrel{\widehat{=}} \max\{\sup_{x\in\aleph}\inf_{y\in\aleph'}\mathsf{d}(x,y), \sup_{y\in\aleph'}\inf_{x\in\aleph}\mathsf{d}(x,y)\}$ is the Hausdorff metric on timed refusals, with $\mathsf{d}((t,a),(t',a')) \mathrel{\widehat{=}} |t-t'|$ provided $a=a'$, and is ∞ otherwise. We also stipulate that $\mathsf{d}_H(\emptyset,\emptyset) \mathrel{\widehat{=}} 0$. It is then easy to see that td makes TF into a (non-complete) metric space.

Theorem 6.13 *For any $P \in \overline{\mathbf{TCSP}}$, $k \geqslant 1$, and $(s,\aleph) \in \mathsf{Exp}_k(\mathcal{R}[\![kP]\!])$, there exists $(s',\aleph') \in \mathcal{F}_T[\![P]\!]$ such that $\mathsf{td}((s',\aleph'),(s,\aleph)) < 1/k$.*

Proof (Sketch.) The proof proceeds in a manner very similar to that of Theorem 6.6, with refusal information now incorporated. It may be necessary to use the downward-closedness of refusals to 'trim' $\aleph'$ to a suitable size. ■

We now present what is in our view the most important and effective approach to extract useful information from the $\mathcal{M}_R$ representation of processes.

Let $0 \leqslant \varepsilon \leqslant 1$ be given. For $(s, \aleph) \in TF$, we define the *ε-digitisation* of $(s, \aleph)$ to be $[(s, \aleph)]_\varepsilon \mathrel{\widehat{=}} ([s]_\varepsilon, [\aleph]_\varepsilon)$, where $[\aleph]_\varepsilon \mathrel{\widehat{=}} \bigcup\{[[b_i]_\varepsilon, [e_i]_\varepsilon) \times A_i \,|\, 1 \leqslant i \leqslant n\}$ if $\aleph = \bigcup\{[b_i, e_i) \times A_i \mid 1 \leqslant i \leqslant n\}$[8]. Naturally, this definition extends to sets of timed failures in the obvious way. We now say that a set $P \subseteq TF$ is *closed under digitisation* if, for any $0 \leqslant \varepsilon \leqslant 1$, $[P]_\varepsilon \subseteq P$. On the other hand, we say that P is *closed under inverse digitisation* if, whenever a timed failure $(s, \aleph)$ is such that $[(s, \aleph)]_\varepsilon \in P$ for all $0 \leqslant \varepsilon \leqslant 1$, then $(s, \aleph) \in P$.

Theorem 6.14 *For any $P \in \overline{\mathbf{TCSP}}$, $\mathcal{F}_T[\![P]\!]$ is closed under digitisation.*

Proof (Sketch.) Follows easily from the digitisation lemma and the congruence theorem. ■

For $P \subseteq TF$, let $\mathbf{Z}(P)$ denote the subset of *integral* timed failures of P, in other words those timed failures in P consisting of an integral trace together with a refusal all of whose tokens have integral time bounds. We can now extend Henzinger *et al.*'s result (Theorem 6.8) to timed failures:

Theorem 6.15 *If $P \subseteq TF$ is closed under digitisation and $S \subseteq TF$ is closed under inverse digitisation, then*

$$\mathbf{Z}(P) \subseteq \mathbf{Z}(S) \Leftrightarrow P \subseteq S.$$

Proof The right-to-left implication is immediate (and in fact requires no assumptions on either P or S). For the converse, assume $\mathbf{Z}(P) \subseteq \mathbf{Z}(S)$, and let $(s, \aleph) \in P$. Since P is closed under digitisation, $[(s, \aleph)]_\varepsilon \in P$ for all $0 \leqslant \varepsilon \leqslant 1$. But since each $[(s, \aleph)]_\varepsilon$ is an integral timed failure, it must belong to $\mathbf{Z}(P)$, and hence to $\mathbf{Z}(S)$. Since the latter is closed under inverse digitisation, $(s, \aleph) \in S$ follows, as required. ■

For any set $S \subseteq TF$ of timed failures, it is straightforward to show that there always exists a largest set of tests $\mathbf{Z}_R(S) \subseteq TEST$ with the property

$$\mathbf{Z}(S) = \mathbf{Z}(\mathsf{Exp}(\mathbf{Z}_R(S))).$$

[8]Note that this definition is independent of the particular representation of $\aleph$ as a finite union of refusal tokens.

$\mathbf{Z}_R$ therefore represents a function from $\mathcal{P}(TF)$ to $\mathcal{P}(TEST)$.

Recall that any behavioural timed failure specification can be identified with a set of timed failures: for S such a specification and P a $\overline{\mathbf{TCSP}}$ program, $P \vDash S$ if and only if $\mathcal{F}_T[\![P]\!] \subseteq S$. Given a timed failure specification S, let us therefore also write $\mathbf{Z}_R(S)$ to denote the corresponding behavioural test specification. Our main result reads:

Theorem 6.16 *Let $S = S(P)$ be a behavioural timed failure specification closed under inverse digitisation. Then, for any program $P \in \overline{\mathbf{TCSP}}$,*

$$\mathcal{R}[\![P]\!] \vDash \mathbf{Z}_R(S) \Leftrightarrow \mathcal{F}_T[\![P]\!] \vDash S.$$

Proof The proof proceeds in a manner almost identical to that of Theorem 6.10, resting on Theorem 6.15 rather than Theorem 6.8. ■

This theorem allows us to handle both safety and liveness specifications on Timed CSP processes, as long as the specifications in question are closed under inverse digitisation. An example of how this technique can be applied in practice will be given in Section 6.7.

Theorem 6.16 yields the following central corollary:

Corollary 6.17 *Let $Q \in \overline{\mathbf{TCSP}}$ be a program such that $\mathcal{F}_T[\![Q]\!]$ is closed under inverse digitisation. Then, for any program $P \in \overline{\mathbf{TCSP}}$,*

$$\mathcal{R}[\![Q]\!] \sqsubseteq_R \mathcal{R}[\![P]\!] \Leftrightarrow \mathcal{F}_T[\![Q]\!] \sqsubseteq_{TF} \mathcal{F}_T[\![P]\!].$$

Proof (Sketch.) This result rests on the fact that $\mathcal{R}[\![Q]\!]$ satisfies the crucial property $\mathbf{Z}(\mathcal{F}_T[\![Q]\!]) = \mathbf{Z}(\mathsf{Exp}(\mathcal{R}[\![Q]\!]))$, which can be shown using the line of reasoning offered in the proof of Theorem 6.10. (Note for the right-to-left implication that, even though $\mathcal{R}[\![Q]\!]$ may fail to equal $\mathbf{Z}_R(\mathcal{F}_T[\![Q]\!])$, all we need is the downward-closedness of $\mathcal{R}[\![Q]\!]$.) The desired conclusion then follows from Theorem 6.16. ■

We remark that closure under inverse digitisation does *not* here characterise those downward-closed behavioural timed failure specifications that obey the conclusion of Theorem 6.16, as the following proposition shows:

Proposition 6.18 *There exists a program $Q \in \overline{\mathbf{TCSP}}$ with $\mathcal{F}_T[\![Q]\!]$* not *closed under inverse digitisation, but such that, for any program $P \in \overline{\mathbf{TCSP}}$, $\mathcal{R}[\![Q]\!] \sqsubseteq_R \mathcal{R}[\![P]\!] \Leftrightarrow \mathcal{F}_T[\![Q]\!] \sqsubseteq_{TF} \mathcal{F}_T[\![P]\!]$.*

Proof Recall the program $Q = (\bar{a} \overset{0}{\rhd} \mathit{WAIT}\ 1)\ ;\ Q$ of Section 6.4. The left-to-right implication has already been observed. The other direction is left to the reader. ■

Proposition 6.18 suggests that the model $\mathcal{M}_R$ contains extraneous information with respect to the goal of verifying exclusively those specifications that are closed under inverse digitisation. However, we discuss in Chapter 9 a proposal for a simpler discrete-time model which we believe will lead to a full abstraction result extending Proposition 6.12. As a result, it should be possible to improve the efficiency of the model checking algorithm described in Appendix B.

6.6 Refinement Analysis

Even though Corollaries 6.11 and 6.17 have provided us with powerful exact verification tools, we still have no means to handle refinement-based specifications in general. The results of this section suggest that this is in fact a very difficult goal, at least within our discretisation context.

The proof of the following proposition is based on a counterexample suggested by Bill Roscoe.

Proposition 6.19 *There are programs $P, Q \in \overline{\mathbf{TCSP}}$ such that $Q \sqsubseteq_R P$ but $Q \not\sqsubseteq_{TF} P$.*[9]

Proof We first let $T = (\bar{a} \overset{0}{\rhd} \mathit{WAIT}\ 1)\ ;\ T$, and then define $Q = T \mid\mid\mid \bar{b}$ and $P = (T \mid\mid\mid b \longrightarrow T) \underset{\{a\}}{\|} \bar{a}$. We now show that $\mathcal{R}[\![Q]\!] = \mathcal{R}[\![P]\!]$ but $\mathsf{TTraces}(\mathcal{F}_T[\![Q]\!]) \neq \mathsf{TTraces}(\mathcal{F}_T[\![P]\!])$ (which of course implies the desired result).

To see that $\mathcal{R}[\![Q]\!] = \mathcal{R}[\![P]\!]$, note that either process may communicate, in any order, a single a and a single b. In either process, communication of an a necessarily occurs from an unstable state, whereas communication of a b can always happen from a stable state in which the event a is refused.

[9]As also pointed out by Bill Roscoe, one can even require in addition that P and Q be free of transient events: let $R = (\mathit{WAIT}\ 1 \sqcap \mathit{WAIT}\ 2)\ ;\ \bar{a}$, and define $Q = R \mid\mid\mid \bar{b}$ and $P = (R \mid\mid\mid b \longrightarrow (\mathit{STOP} \sqcap \mathit{WAIT}\ 1\ ;\ \bar{a})) \underset{\{a\}}{\|} \bar{a}$. It is then easy to verify that P and Q, both free of transient events, satisfy the statement of the proposition.

On the other hand, the timed trace $\langle (0.5, b), (1.5, a) \rangle$ is a timed trace of $\mathcal{F}_T[\![P]\!]$ but not of $\mathcal{F}_T[\![Q]\!]$, which concludes the proof. ■

Together with Proposition 6.19, the next proposition shows that the orders $\sqsubseteq_R$ and $\sqsubseteq_{TF}$ are incomparable over $\overline{\mathbf{TCSP}}$.[10]

Proposition 6.20 *There are programs $P, Q \in \overline{\mathbf{TCSP}}$ such that $Q \sqsubseteq_{TF} P$ but $Q \not\sqsubseteq_R P$.*

Proof Define

$$\begin{aligned} P &= \bar{a} \sqcap (\bar{a} \Box \bar{c}) \sqcap ((\bar{b} \overset{1}{\rhd} STOP) \underset{\{b\}}{\|} (a \xrightarrow{1} \bar{b})) \\ Q &= \bar{a} \sqcap (\bar{a} \Box \bar{c}) \sqcap ((\bar{b} \overset{1}{\rhd} STOP) \underset{\{b\}}{\|} ((a \xrightarrow{1} \bar{b}) \Box \bar{c})). \end{aligned}$$

We claim that $P =_{TF} Q$. To see this, observe that any distinction in P and Q's behaviours must originate from the third term of their respective constructions as internal choices. In that component, either process can communicate an a at any time, or an a at time 0 possibly followed by a b at time 1. Note that both processes are in any case able to continuously refuse b, since b is only available in transient form. In addition, Q can communicate c on the empty trace, whereas P can refuse c on the empty trace. But thanks to P's second term, P too can communicate a c on the empty trace while recording any refusal Q could have, whereas thanks to Q's first term, Q too is able to refuse c on the empty trace and communicate a later on. Lastly, on the trace where a happens at time 0 and b happens at time 1, both processes are able to refuse a, b, and c throughout. They therefore have exactly the same set of timed failures as claimed.

However, $Q \not\sqsubseteq_R P$, because the test $\langle \{c\}, a, \bullet, tock, \bullet, b, \bullet \rangle$ clearly belongs to $\mathcal{R}[\![P]\!]$ but not to $\mathcal{R}[\![Q]\!]$ ($\mathcal{R}[\![Q]\!]$ is unable to stably refuse c, prior to communicating a, on any test where b is communicated later on). ■

The following result offers an interesting partial converse to Proposition 6.19.

Theorem 6.21 *Let $P, Q \in \overline{\mathbf{TCSP}}$, and suppose that $kQ \sqsubseteq_R kP$ for arbitrarily large values of k. Then $Q \sqsubseteq_{TF} P$.*

[10] We discuss in Chapter 9 a proposal for an alternative simpler discrete-time model equipped with a refinement order strictly coarser than both test and timed failure refinement, but nonetheless strong enough for an equivalent version of Theorem 6.16 to hold.

Proof (Sketch.) Let $(s, \aleph) \in \mathcal{F}_T[\![P]\!]$, with $\aleph = \bigcup_{i=1}^{n}\{[b_i, e_i) \times A_i | 1 \leqslant i \leqslant n\}$, where the union is taken over non-empty sets. (The case in which $\aleph$ itself is empty is addressed below.) Choose k large enough so that $kQ \sqsubseteq_R kP$ and $k \cdot \min\{(|e_i - b_i|) \mid 1 \leqslant i \leqslant n\} > 2$.[11] Imitating the idea of the proofs of Theorems 6.6 and 6.13, pick $e_{kP} \in \mathsf{exec}_{TF}(kP)$ such that e_{kP} **fail** $k(s, \aleph)$. Let $e'_{kP} \in \mathsf{exec}_{TF}(kP)$ be the lower digitisation of e_{kP}. There is then $(s', \aleph') \in \mathcal{F}_T[\![P]\!]$ such that e'_{kP} **fail** $k(s', \aleph')$ and $\mathsf{td}(k(s', \aleph'), k(s, \aleph)) < 1$, which clearly implies that $\mathsf{td}((s', \aleph'), (s, \aleph)) < 1/k$. (If $\aleph$ is empty, $\aleph'$ must be chosen empty as well for the estimate to hold.)

Let $e''_{kP} \in \mathsf{exec}_{TF}(kP)$ be the normalisation of e'_{kP}. There is then a corresponding execution $o_{kP} \in \mathsf{exec}_R(kP)$, as per Proposition 6.2. Let u be the $\prec$-maximal test of $\mathcal{R}[\![kP]\!]$ such that o_{kP} **test** u. By our assumption that $kQ \sqsubseteq_R kP$, we have $u \in \mathcal{R}[\![kQ]\!]$. Let $o_{kQ} \in \mathsf{exec}_R(kQ)$ be such that o_{kQ} **test** u, and let $e_{kQ} \in \mathsf{exec}_{TF}(kQ)$ be the normal execution corresponding to o_{kQ}. Using the fact that, for any program R, $\mathsf{init}_R(R) = \mathsf{init}_{TF}(R)$, the reader will easily verify that e_{kQ} **fail** $k(s', \aleph')$, and therefore that $(s', \aleph') \in \mathcal{F}_T[\![Q]\!]$.

Since this procedure can be repeated for arbitrarily large k, we get a sequence $\{(s'_i, \aleph'_i)\} \subseteq \mathcal{F}_T[\![Q]\!]$ such that $\mathsf{td}((s'_i, \aleph'_i), (s, \aleph)) < 1/i$, for $i \geqslant 1$. We finish the proof by showing that this implies that $(s, \aleph) \in \mathcal{F}_T[\![Q]\!]$, in other words that the set $\mathcal{F}_T[\![Q]\!]$ is a closed subset of the metric space (TF, td). Since $(s, \aleph)$ was an arbitrary timed failure of P, this will establish the required refinement $Q \sqsubseteq_{TF} P$.

We proceed by structural induction on Q, over all timed failures $(s, \aleph) \in TF$. This requires letting Q range over the set **TCSP** of terms, together with the introduction of suitable syntactic bindings. The inductive case for recursion then follows via the indexed bisimulation techniques which we have used in previous proofs, and we shall therefore be omitting those details. Select remaining cases of the structural induction (in which syntactic bindings are omitted) are presented below.

case $a \longrightarrow Q$**:** We assume that Q satisfies the induction hypothesis (for any timed failure). Let $(s, \aleph) \in TF$ be the limit of timed failures $\{(s_i, \aleph_i)\}$, where each $(s_i, \aleph_i) \in \mathcal{F}_T[\![a \longrightarrow Q]\!]$. Either s is the empty trace, in which case the result trivially holds, or $s = \langle (t, a) \rangle ^\frown s'$, since each s_i can only begin with an a. The induction hypothesis easily yields that $(s', \aleph) - t \in \mathcal{F}_T[\![Q]\!]$, and the required result follows.

[11]The second condition on k is necessary for our forthcoming estimate $\mathsf{td}(k(s', \aleph'), k(s, \aleph)) < 1$ to hold, since the Hausdorff distance between an empty set and a non-empty set is infinite.

case $Q_1 \underset{B}{\|} Q_2$**:** Follows from the induction hypothesis on Q_1 and Q_2 and the denotational definition of parallel composition.

case $Q_1 \,;\, Q_2$**:** Let $(s, \aleph)$ be the limit of $\{(s_i, \aleph_i)\} \subseteq \mathcal{F}_T[\![Q_1 \,;\, Q_2]\!]$. We can assume that only a finite number (and hence, without loss of generality, none) of the timed failures $(s_i, \aleph_i \cup ([0, \mathsf{end}((s_i, \aleph_i))) \times \{\checkmark\}))$ actually belong entirely to $\mathcal{F}_T[\![Q_1]\!]$, otherwise the result follows immediately from the induction hypothesis. We therefore have, for each i, $s_i = s_i^1 \frown s_i^2$, where each $(s_i^1 \frown \langle (t_i, \checkmark) \rangle, \aleph_i \Uparrow t_i \cup ([0, t_i) \times \{\checkmark\})) \in \mathcal{F}_T[\![Q_1]\!]$, and each $(s_i^2, \aleph_i) - t_i \in \mathcal{F}_T[\![Q_2]\!]$.

Note that the t_i's must be bounded, otherwise we could find arbitrarily large values of i such that $t_i \geqslant \mathsf{end}((s, \aleph)) + 1$. Since s_i converges to s, s_i^2 would eventually have to be the empty trace, and $(s_i, \aleph_i \cup ([0, \mathsf{end}((s_i, \aleph_i))) \times \{\checkmark\}))$ would belong entirely to $\mathcal{F}_T[\![Q_1]\!]$, contradicting our earlier assumption.

Since s is finite there must exist some division $s = s^1 \frown s^2$ of s and some infinite subsequence $\{s_{i_j}\}$ of $\{s_i\}$ such that $\{s_{i_j}^1\}$ converges to s^1.

The set $\{t_{i_j}\}$ is bounded, and hence has an infinite subsequence which converges to some t, by the Bolzano-Weierstrass theorem (cf., e.g., [Mar74]). For ease of notation, let us assume that the sequence $\{t_{i_j}\}$ itself has this property.

It then follows that the sequence $\{(s_{i_j}^1 \frown \langle (t_{i_j}, \checkmark) \rangle, \aleph_{i_j} \Uparrow t_{i_j} \cup ([0, t_{i_j}) \times \{\checkmark\}))\}$ converges to $(s^1 \frown \langle (t, \checkmark) \rangle, \aleph \Uparrow t \cup ([0, t) \times \{\checkmark\}))$. By the induction hypothesis on Q_1, we then have $(s^1 \frown \langle (t, \checkmark) \rangle, \aleph \Uparrow t \cup ([0, t) \times \{\checkmark\})) \in \mathcal{F}_T[\![Q_1]\!]$.

On the other hand, we also have the sequence $\{(s_{i_j}^2, \aleph_{i_j}) - t_{i_j}\}$ converging to $(s^2, \aleph) - t$, and therefore we conclude by the induction hypothesis on Q_2 that $(s^2, \aleph) - t \in \mathcal{F}_T[\![Q_2]\!]$.

The desired result, $(s, \aleph) \in \mathcal{F}_T[\![Q_1 \,;\, Q_2]\!]$, then follows by the denotational definition of sequential composition.

case $Q \setminus A$**:** This case essentially uses the same ideas as those introduced for sequential composition, above, to handle maximal progress. No other particular difficulties are encountered.

■

A legitimate question is whether, given a program Q, one can always find some k such that $\mathcal{F}_T[\![kQ]\!]$ is closed under inverse digitisation. This is not the case, as the following proposition indicates. In view of Proposition 6.12, this means that Theorem 6.21 above is, in some sense, the sharpest possible partial converse to Proposition 6.19, at least as far as timed trace refinement is concerned.

Proposition 6.22 *There exists a program $Q \in \overline{\mathbf{TCSP}}$ such that $\mathcal{F}_T[\![kQ]\!]$ is not closed under inverse digitisation, for any $k \geqslant 1$.*

Proof Let $T = (\bar{a} \overset{0}{\rhd} \mathit{WAIT}\ 1)\ ;\ T$, and let $Q = (a \longrightarrow T) \mathbin{|||} \bar{a}$. Let $s = \langle (0.1, a), (0.2, a), (0.3, a) \rangle$. Observe that, for any $k \geqslant 1$, we have $(s, \emptyset) \notin \mathcal{F}_T[\![kQ]\!]$, even though every digitisation of $(s, \emptyset)$ clearly belongs to $\mathcal{F}_T[\![kQ]\!]$. $\mathcal{F}_T[\![kQ]\!]$ is therefore not closed under inverse digitisation. ■

It is interesting to note that *sequential* programs—programs in which the parallel or interleaving operators do not figure—have the property that $P \sqsubseteq_R Q$ implies that $kP \sqsubseteq_R kQ$ for any $k \geqslant 1$ (here both P and Q are assumed to be sequential). This follows from the fact that for such programs P, there is a simple, $\sqsubseteq_R$-monotone procedure to calculate $\mathcal{R}[\![kP]\!]$ from $\mathcal{R}[\![P]\!]$. Indeed, a sequential program is one which always ever has at most one 'clock' running; and moreover it is straightforward to know at any point whether a clock is in fact running or not. Concurrent programs, on the other hand, can have arbitrarily many clocks running simultaneously, and it seems to be very difficult to systematically keep track of all of them at the same time.

It follows from Theorem 6.21 that, whenever P and Q are sequential programs, one can establish that $Q \sqsubseteq_{TF} P$ merely by showing that $Q \sqsubseteq_R P$. This in itself is a result of relatively minor significance, since one would expect most interesting Timed CSP processes to exhibit some form of concurrency, but it draws one's attention to an interesting contrast between CSP and Timed CSP: standard untimed models for CSP, as well as extensions such as refusal testing, have the desirable property that any process is equivalent to a sequential one. Typically, the sequential process in question is some kind of *normal form* of the original one [Ros97, Muk93]. This of course cannot be the case for timed models, in view of Proposition 6.19 and Theorem 6.21. In fact, [RR99] observe that the process $\bar{a} \mathbin{|||} \bar{b}$, for example, does *not* have an equivalent sequential form in their version of the timed failures model. While this is of course true in their model, owing to the fact that every communication of an event in a sequential process is postulated to be followed by a small period of 'inaction', $\bar{a} \mathbin{|||} \bar{b} = (a \longrightarrow \bar{b}) \mathbin{\Box} (b \longrightarrow \bar{a})$ in any other model, such as $\mathcal{M}_R$ and $\mathcal{M}_{TF}$, which exhibits *instant causality.* Nonetheless, these models still fail to admit sequential normal forms, as the following proposition shows directly:

Proposition 6.23 *There exist* $\overline{\mathbf{TCSP}}$ *programs which are not equivalent to any sequential program, in either* $\mathcal{M}_R$ *or* $\mathcal{M}_{TF}$*.*[12]

Proof (Sketch.) Consider the program $P = \bar{a} \mathbin{|||} (WAIT\ 1\ ;\ \bar{b})$. Let $N \in \overline{\mathbf{TCSP}}$ be a sequential program. Show by induction on N that

1. If $(\langle (0.5, a) \rangle, [0.5, 1) \times \{b\}) \in \mathcal{F}_T[\![N]\!]$, then $(\langle (0.5, a) \rangle, [0.5, 1.5) \times \{b\}) \in \mathcal{F}_T[\![N]\!]$.
2. If $\langle \emptyset, a, \{b\} \rangle \in \mathcal{R}[\![N]\!]$, then $\langle \bullet, \mathit{tock}, \bullet, a, \{b\} \rangle \in \mathcal{R}[\![N]\!]$.

Conclude that $P \neq_{TF} N$ and $P \neq_R N$. ■

6.7 Example: Railway Level Crossing

We conclude this chapter by presenting a larger verification example based on a simplified version of the well-known railway level crossing problem [HJL93].[13]

We describe in Timed CSP a closed system made up of four distinct components: trains, travelling at bounded speeds on a stretch of rail incorporating a level crossing; cars, able to cross the tracks in a bounded amount of time; a traffic light, meant to signal cars not to attempt crossing the railway when a train is nearby; and a monitor, whose rôle is to signal that a collision has happened. For simplicity we assume that only at most one train and one car are present at any one time within the system.

Trains are modelled via the process T: in its initial state, this process assumes that there are no trains on the tracks, and offers the event $t.in$. This event represents a sensor signal which indicates that an incoming train is at least $60s$ away from the crossing. When the train reaches the crossing, the event $t.on$ is triggered, and as soon as the train is a safe distance past the crossing, the event $t.out$ registers. We assume that the train takes at

[12] A 'head standard form' for Timed CSP processes has been proposed in [Sch92], and any process in $\mathcal{M}_{TF}$ can be shown equivalent to one in head standard form. Processes in head standard form are built from unbounded nondeterministic choice, timeout, and a generalised prefix operator.

[13] A '*tock*-time' version of this problem is studied in [Ros97]; it is an interesting example to look at as it highlights many of the differences between our approach to discrete-time modelling and Roscoe's, the latter being much closer to untimed CSP.

least 20s to cross the crossing, and that the process T returns to its initial state (allowing further trains to arrive) 1s after the event $t.out$ is received.

The process C models the cars: initially there are no cars on the crossing, and C offers the event $c.on$, subject to synchronisation with the traffic light, indicating that a car is just about to drive onto the crossing. The car stays in this vulnerable position for at most 10s, sending out the event $c.out$ as soon as it is safely on the other side. For simplicity we will make the conservative assumption that the time taken to cross the tracks is actually exactly 10s. In order to ensure that the car does step out immediately after this time, however, we will later on hide the event $c.out$, to enforce its urgency. A new car is allowed on the crossing 1s after the car ahead has left it.

The traffic light is green to start with, modelled by the process GL, and becomes red as soon as a train ($t.in$) is detected. While it is red, the event $c.on$ is disabled (modelling the assumption that any car not yet on the crossing obeys the red light), and is only re-enabled 1s after $t.out$ has registered.

A collision will occur if the train enters the crossing while a car is already there, or vice-versa; in either case this will cause the monitoring process M to send out the catastrophic event $coll$.

The entire level crossing system LC is modelled as the parallel composition of these four components, with $c.out$ hidden.

Translating this description into Timed CSP, we get:

$$\begin{aligned}
T &= t.in \xrightarrow{60} t.on \xrightarrow{20} t.out \xrightarrow{1} T \\
C &= c.on \xrightarrow{10} c.out \xrightarrow{1} C \\
GL &= (t.in \longrightarrow RL) \,\Box\, (c.on \longrightarrow ((t.in \longrightarrow RL) \overset{1}{\rhd} GL)) \\
RL &= t.out \xrightarrow{1} GL \\
M &= (t.on \longrightarrow Mt) \,\Box\, (c.on \longrightarrow Mc) \\
Mt &= (c.on \longrightarrow coll \longrightarrow STOP) \,\Box\, (t.out \xrightarrow{1} M) \\
Mc &= (t.on \longrightarrow coll \longrightarrow STOP) \,\Box\, (c.out \xrightarrow{1} M) \\
LC &= ((T \underset{U}{\parallel} (GL \underset{V}{\parallel} C)) \underset{W}{\parallel} M) \setminus \{c.out\}
\end{aligned}$$

where $U = \{t.in, t.out\}$, $V = \{c.on\}$, and $W = \{t.on, c.on, t.out, c.out\}$.

We would now like to prove that this system is both *safe* and *efficient*.

Safety means that no collision can ever occur. Since a useless process such as $STOP$ (forbidding cars from moving at all!) is clearly 'safe', we also require that our system be efficient, in the sense that it not block car traffic any longer than it has to.

The safety condition S, on $\mathcal{F}_T[\![LC]\!]$, is expressed thus: 'The event *coll* is never witnessed'.

S is a qualitative behavioural timed trace specification, and is therefore closed under inverse digitisation, as seen in Section 6.4. Theorem 6.10 then tells us that $\mathcal{F}_T[\![LC]\!] \vDash S$ if and only $\mathcal{R}[\![LC]\!] \vDash \mathbb{Z}(S)$. For model checking purposes, we express the latter as the refinement $\mathcal{R}[\![RUN_{\Sigma-\{coll\}}]\!] \sqsubseteq_T \mathcal{R}[\![LC]\!]$, where $\Sigma = \{t.in, t.on, t.out, c.on, c.out, coll\}$ represents LC's alphabet. As discussed in Section 5.4, this is equivalent to the assertion $\Psi(RUN_{\Sigma-\{coll\}}) \sqsubseteq_T \Psi(LC)$ in the prioritised version of the traces model for CSP. Since $\Psi(LC)$ is clearly a finite-state process (as it only comprises *tail-recursions*), this last refinement can be established using FDR.

Efficiency is slightly more complicated. If $(s, \aleph)$ is a particular run of the system (i.e., $(s, \aleph) \in \mathcal{F}_T[\![LC]\!]$), we first determine whether a car has entered the crossing in the last 10s (i.e., whether $c.in \in \sigma(s \upharpoonright [\mathsf{end}(s) - 10, \mathsf{end}(s)] \times \Sigma^{\checkmark})$, and then whether the event $t.in$ has at some point been witnessed with no subsequent $t.out$ event. If neither eventuality is the case, a car should be allowed on the crossing within at most 1s; in other words, $c.in \notin \aleph - (\mathsf{end}(s) + 1)$. We write this specification as E.

E is a liveness specification, i.e., a behavioural timed failure specification. It is closed under inverse digitisation: indeed, if $(s, \aleph)$ is a behaviour banned by E, then so is $([s]_\varepsilon, [\aleph]_\varepsilon)$, where ε represents the fractional part of $\mathsf{end}(s)$. We can therefore apply Theorem 6.16 to conclude that $\mathcal{F}_T[\![LC]\!] \vDash E$ if and only if $\mathcal{R}[\![LC]\!] \vDash \mathbf{Z}(E)$.

Rather than express the assertion $\mathcal{R}[\![LC]\!] \vDash \mathbf{Z}(E)$ mathematically, we will write it as the refinement $\mathcal{R}[\![EFF]\!] \sqsubseteq_F \mathcal{R}[\![LC]\!]$, where EFF is a process displaying all of the behaviours specified by $\mathbf{Z}(E)$, and $\sqsubseteq_F$ denotes *failure refinement*, the logical choice since we are exclusively interested in refusals occurring at the *end* of traces.

To construct the process EFF, we reason as follows. EFF will have five different types of states, to represent each of the different situations described by the specification $\mathbf{Z}(E)$:

1. The initial state, GO, is the one in which cars are immediately allowed onto the crossing.
2. The state TRN is reached when a train has been detected and has not

yet exited. This state is also understood to indicate that no car is on the crossing.

3. The indexed states CAR_n, for $1 \leqslant n \leqslant 10$, represent the situation in which a car has most recently driven onto the crossing $10 - n$ *tock*s (representing seconds) earlier. These states also indicate that no train is currently on the tracks.

4. To describe the situation in which both a train and a car have been detected (in either order), we use the indexed states TC_n, for $1 \leqslant n \leqslant 10$, which have obvious meanings.

5. Lastly, to effect the transition back from one of these non-GO states into GO, we define the state GO_1, which holds up the free passage of cars for exactly one *tock*, representing the small delay it takes for the system to settle back into its initial state after the exit of a car or a train.

The process EFF will move from state to state in the expected way; for example, it will move from GO to CAR_{10} upon communication of the $c.on$ event; it will move from CAR_4 to CAR_3 upon communication of a *tock*; it will move from TRN to GO_1 upon communication of $t.out$; it will stay in CAR_8 upon communication of $t.out$ (even though this is a behaviour that should have earlier led to a collision in LC!), and so on. EFF will be so designed as never to refuse the event $c.on$ when in the state GO, whereas events (apart from *tock*s) in all other situations will (nondeterministically) both be refusable and acceptable, to reflect the fact that EFF's behaviour is as nondeterministic as possible outside of what is prescribed by the specification $\mathbf{Z}(E)$.

Without further ado, we give

$$\begin{array}{rcl}
GO & = & (c.on \longrightarrow CAR_{10}) \,\Box \\
 & & ((t.in \longrightarrow TRN) \sqcap \\
 & & \;(a : \Sigma - \{c.on, t.in\} \longrightarrow GO) \sqcap \\
 & & \;STOP) \\
TRN & = & (t.out \longrightarrow GO_1) \sqcap \\
 & & (c.on \longrightarrow TC_{10}) \sqcap \\
 & & (a : \Sigma - \{t.out, c.on\} \longrightarrow TRN) \sqcap \\
 & & STOP
\end{array}$$

$$\begin{array}{rcl}
CAR_n & = & ((t.in \longrightarrow TC_n) \sqcap \\
& & (c.on \longrightarrow CAR_{10}) \sqcap \\
& & (a : \Sigma - \{t.in, c.on\} \longrightarrow CAR_n)) \overset{0}{\rhd} \\
& & WAIT\ 1\ ; (GO_1 \mathrel{\langle\!|} n = 1 \mathrel{|\!\rangle} CAR_{n-1}) \\
TC_n & = & ((t.out \longrightarrow CAR_n) \sqcap \\
& & (c.on \longrightarrow TC_{10}) \sqcap \\
& & (a : \Sigma - \{t.out, c.on\} \longrightarrow TC_n)) \overset{0}{\rhd} \\
& & WAIT\ 1\ ; (TRN \mathrel{\langle\!|} n = 1 \mathrel{|\!\rangle} TC_{n-1}) \\
GO_1 & = & ((c.on \longrightarrow CAR_{10}) \sqcap \\
& & (t.in \longrightarrow TRN) \sqcap \\
& & (a : \Sigma - \{c.on, t.in\} \longrightarrow GO_1)) \overset{0}{\rhd} \\
& & WAIT\ 1\ ; GO.
\end{array}$$

We now simply set $EFF = GO$.

We remark that, in this construction, we have taken advantage of the fact that EFF is intended for *failure* comparison, rather than *test* comparison. Had this not been the case, our set of equations would have had to be slightly more complex in order to achieve maximum nondeterminism outside of the behaviour prescribed by $\mathbf{Z}(E)$.

As discussed in Section 5.4, checking whether $\mathcal{R}[\![EFF]\!] \sqsubseteq_F \mathcal{R}[\![LC]\!]$ is equivalent to checking whether $\Psi(EFF) \sqsubseteq_F \Psi(LC)$ in the prioritised version of the failures model for CSP. Here again, it is not difficult to see that both the processes involved are finite-state, and therefore that the check can be carried out on FDR.

The system LC can thus be shown to be both safe and efficient, as surmised.

Chapter 7

The Discrete-Time Unstable Refusal Testing Model

We now present an extension of the discrete-time refusal testing model $\mathcal{M}_R$ which is able to satisfactorily model *unstable* refusal information. We recall that a process is in an unstable state if it is immediately offering silent events. In $\mathcal{M}_R$, the only refusal that such a process could record was the null refusal $\bullet$. In this new model, called the *discrete-time unstable refusal testing model*, denoted $\mathcal{M}_{UR}$, sequences of *unstable refusals* are recorded prior to reaching stability, at which point the modelling becomes similar to that of $\mathcal{M}_R$. As we shall see, the extra information provides some useful insight into a process's behaviour, and in particular will allow us in the next chapter to derive a timed failure soundness result extending Theorem 6.5.

Aside from the denotational model $\mathcal{M}_{UR}$, we present a congruent operational semantics, in the same style as that associated with the model $\mathcal{M}_R$.

As expected, the model $\mathcal{M}_{UR}$ is strictly less abstract than $\mathcal{M}_R$. $\mathcal{M}_{UR}$ representations of programs therefore offer greater insight into their 'true', continuous-time behaviour, albeit at a significant cost in complexity (within the denotational and operational models, and also in terms of model checking). If, as we believe, the paradigm of concentrating exclusively on specifications that are closed under inverse digitisation constitutes the 'right' way to exploit our general discretisation approach, then the model $\mathcal{M}_{UR}$ is clearly inferior to $\mathcal{M}_R$. On the other hand, the broader goal of this thesis being the study of the relationship between continuous-time and discrete-time modelling approaches to real-time concurrent systems, we would be remiss not to investigate possible alternatives to $\mathcal{M}_R$, especially given the 'inaccuracies' observed in the latter. Two further discrete-time model proposals will in fact be briefly discussed in Chapter 9.

Proofs in this chapter are, for the most part, essentially omitted, as they often follow the same patterns as their counterparts in Chapter 5. Our notation also almost systematically either borrows from, overloads, or extends, that used earlier. We trust this enhances the clarity of our exposition.

7.1 Denotational Semantics

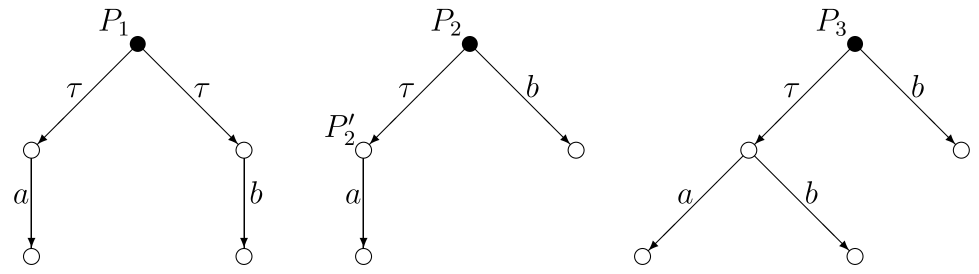
We define the semantic model $\mathcal{M}_{UR}$ and the associated semantic mapping $\mathcal{R}_U[\![\cdot]\!] : \overline{\mathbf{TCSP}} \longrightarrow \mathcal{M}_{UR}$.

The philosophy underlying the model is once again one of *experimentation* and *observation*. Processes are black boxes equipped with labelled buttons, together with a red light and a buzzer. This red light, when lit, indicates potential internal activity. We are allowed to experiment with the process while the red light is on, and record our observations as follows: if a button does not go down when pressed upon, either individually or as part of a group, a corresponding event or set of events is recorded as an *unstable* refusal. However, since the red light is on, it is possible at any time for the process to silently change states; we can therefore repeat the experiment (with either the same set or a different set of buttons), and record a further unstable refusal (assuming no button goes down), and so on. These unstable refusals are then collated into a finite unstable refusal sequence.

At any point, it is possible, if pressed upon, for a *single* button to go down, in which case the corresponding event is recorded as having occurred in our ongoing observation of the process. A second possibility is for the red light to go off. In that case, we conclude that the process has reached stability, and therefore that any further refusal observed will not be subject to change until either some other button goes down or the buzzer goes off. Any such refusal is then recorded as a single *stable* refusal. Note that if a button goes down while the red light is on, we must record the null refusal $\bullet$ in lieu of a stable refusal. Lastly, a third possibility is for the buzzer to go off, which in fact is assumed to happen regularly. This is then recorded as the event *tock*. We assume that the black box is so designed as never to allow the buzzer to go off at exactly the same time as some button is being depressed, so that events are recorded in unambiguous linear order.

Observations therefore consist of sequences, known as *tests*, in which unstable refusal sequences, stable refusals, and events, cycle in order. Naturally, we require that the set of possible observations be downward-closed.

A few comments on the temporal nature of instability are in order. Consider the processes $P_1 = \bar{a} \sqcap \bar{b}$, $P_2 = \bar{b} \overset{0}{\rhd} \bar{a}$, and $P_3 = ((c \longrightarrow (\bar{a} \,\Box\, \bar{b})) \,\Box\, \bar{b}) \setminus c$, depicted operationally below, with solid disks representing unstable states. (We have omitted the evolutions/*tock*-transitions which should appear from each stable state back to itself.)



Let us now investigate what consequences instability would have on the denotational modelling of these processes, with the caveat that this preliminary analysis will subsequently be revised.

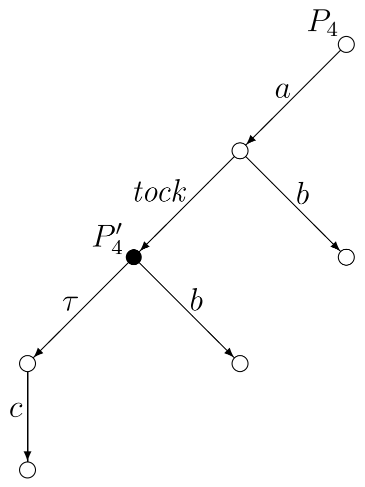
In the unstable initial state P_1, any unstable refusal sequence can be recorded, since no visible events are on offer. As soon as one of the τ-transitions is taken, however, the process is able to stabilise and offer an appropriate stable refusal.

The situation is slightly more subtle when it comes to the second process. In the initial state P_2, the unstable refusals recorded should not normally include the event b, since it is on offer. However, if and when the τ-transition to state P_2' is taken, we see that b can be stably refused. Since we may however fail to realise immediately that the process has indeed stabilised, it is possible for us to then record a refusal, deemed unstable, of b. Note, however, that while in state P_2 an unstable refusal of a can be recorded, as soon as a (stable or unstable) refusal of b is recorded, a cannot be refused any further.

The third process exhibits yet a different behaviour. In the initial state P_3, b cannot be recorded as part of an unstable refusal since it is, once again, on offer. But when the τ-transition is taken, b can still not be (stably or unstably) refused, which means that no refusal of b whatsoever is possible on the empty trace.

While modelling processes along these lines is perfectly feasible, and would in fact yield the requisite soundness result along with most of the other properties we have been seeking, this interpretation of instability is more severe than need be, and has some unfortunate side-effects, such as voiding fundamental laws like $P \sqcap P = P$ and $SKIP \, ; P = P$.

The primary motivation for seeking an alternative treatment of instability originates in processes such as $P_4 = a \longrightarrow (\bar{b} \overset{1}{\rhd} \bar{c})$, depicted below:



If P_4 were modelled in $\mathcal{M}_{TF}$, c would be refusable for exactly one time unit after an occurrence of a, precisely the same period during which b would be available. To ensure that such a behaviour is appropriately 'reflected' when P_4 is modelled in $\mathcal{M}_{UR}$, we would like to consider that the τ-transition emerging from the unstable state P_4' can be slightly 'delayed'. The problem is that, in $\mathcal{M}_R$, precisely because P_4' is unstable, we cannot know whether neither, either, or both events b and c are refusable from state P_4' or not. And certainly, to account for the possibility of the earlier a occurring 'just before' the communication of *tock*, we would like to be able to delay τ by a corresponding amount, while considering that b is on offer and c is refused. This, of course, is precisely what unstable refusal modelling allows us to do.

We are therefore led to the following proposal: whereas in $\mathcal{M}_{TF}$, as a consequence of maximal progress and dense modelling of time, processes can essentially only be unstable at individual, discrete instants, in $\mathcal{M}_{UR}$ we would like to allow *certain* unstable states to linger briefly (although no longer than until the following *tock*), enabling us in the interim to collect accurate and complete unstable refusal information. Other unstable states, such as those arising in processes P_1, P_2, and P_3, should instantaneously be resolved, without giving rise to any unstable refusal information. More precisely, the only unstable states from which we allow the collection of unstable refusal information should be those, such as P'_4, arising from strictly positive prior delays; such states are then termed *robustly unstable*. All other unstable states are deemed *fleetingly unstable*. Thanks to this softer treatment of instability, fundamental laws such $P \sqcap P = P$, $SKIP \; ; P = SKIP$, $(a \longrightarrow P) \setminus a = P \setminus a$, etc., are all preserved in $\mathcal{M}_{UR}$.[1]

We now introduce some necessary notation. An *event* is an element of $\Sigma^{\checkmark}_{tock}$. A *refusal* is an element of $REF \mathrel{\widehat{=}} \{\bullet\} \cup \mathcal{P}(\Sigma^{\checkmark})$. An *unstable refusal sequence* is a finite (possibly empty) sequence of non-$\bullet$ refusals, i.e., an element of $URS \mathrel{\widehat{=}} (\mathcal{P}(\Sigma^{\checkmark}))^{\star}$; to avoid confusion, such sequences will be enclosed in square brackets (as in $[A, B, C]$, rather than the usual $\langle A, B, C\rangle$), and represented by lowercase Greek letters. A *refusal pair* is an (ordered) pair (α, A) where $\alpha \in URS$ and $A \in REF$; the set of all such is denoted $RPAIR$. A *test* is an alternating finite sequence (written in the usual way) of refusal pairs and events, of length at least 1, beginning and ending with a refusal pair. In other words, tests are generated by the grammar $T := \langle R\rangle \mid T^\frown\langle a, R\rangle$, where a and R respectively stand for events and refusal pairs. The set of all tests is denoted $TEST_U$.

If $u = \langle(\alpha_0, A_0), a_1, (\alpha_2, A_2), \ldots, a_k, (\alpha_k, A_k)\rangle \in TEST_U$, let $\mathsf{trace}(u) \mathrel{\widehat{=}} \langle a_1, a_2, \ldots, a_k\rangle$ denote the test u stripped of its refusals.

We define the infix relation $\sim$ on URS to be the smallest equivalence relation such that $[] \sim [\emptyset]$ and $\forall(\alpha, \beta \in URS, A \subseteq \Sigma^{\checkmark}) \bullet \alpha^\frown[A]^\frown\beta \sim \alpha^\frown\alpha'^\frown[A]^\frown\alpha''^\frown\beta$, where $\forall [A'] \text{ in } \alpha'^\frown\alpha'' \bullet A' \subseteq A$. In other words, two unstable refusal sequences are deemed equivalent if we can infer from either that the other could also have been observed. Note that $\sim$ preserves the $^\frown$ operation.

Next, for two unstable refusal sequences $\alpha = [A_1, A_2, \ldots, A_k]$ and $\beta = [B_1, B_2, \ldots, B_{k'}]$, we set $\alpha \prec \beta$ if there exists unstable refusal sequences $\gamma = [C_1, C_2, \ldots, C_{k''}]$ and $\delta = [D_1, D_2, \ldots, D_{k'''}]$ such that $\alpha \sim \gamma$, $\beta \sim \delta$, $k'' \leqslant k'''$, and $\forall(1 \leqslant i \leqslant k'') \bullet C_i \subseteq D_i$.

We then extend $\prec$ to $RPAIR$ as follows: given two refusal pairs $(\alpha, A) = ([A_1, A_2, \ldots, A_k], A)$ and $(\beta, B) = ([B_1, B_2, \ldots, B_{k'}], B)$, we set $(\alpha, A) \prec (\beta, B)$ if either $A = B = \bullet$ and $\alpha \prec \beta$, or if $B \neq \bullet$ and $A \subseteq^* B$ and $\alpha \prec \beta^\frown[B]$.

Lastly, we extend $\prec$ to $TEST_U$ by setting $\langle R_0, a_1, R_1, \ldots, a_k, R_k\rangle \prec \langle R'_0, a'_1, R'_1, \ldots, a'_{k'}, R_{k'}\rangle$

[1]Of course, we do expect $\mathcal{M}_{UR}$ to distinguish more processes than $\mathcal{M}_R$, as a result of the additional information it provides, but it is reassuring to know that we are able to achieve our goals without levelling the basic algebraic landscape which $\mathcal{M}_R$ and $\mathcal{M}_{TF}$ live in.

if $k \leqslant k'$ and $R_0 \prec R_0'$ and $\forall (1 \leqslant i \leqslant k) \centerdot a_i = a_i' \wedge R_i \prec R_i'$.

Note that this makes $\prec$ a preorder rather than a partial order relation, owing to the rôle played by $\sim$.

Definition 7.1 *The* discrete-time unstable refusal testing model $\mathcal{M}_{UR}$ *is the set of all* $P \subseteq TEST_U$ *satisfying the following axioms, where* $u, v \in TEST_U$, $A \in REF$, $\alpha \in URS$, $a \in \Sigma^{\checkmark}_{tock}$, $k \in \mathbb{N}$ *and for all* $1 \leqslant i \leqslant k$, $A_i \in REF$, $\alpha_i \in URS$, *and* $a_i \in \Sigma^{\checkmark}$*—in particular,* $a_i \neq tock$.

UR1 $\langle([], \bullet)\rangle \in P$

UR2 $(u \in P \wedge v \prec u) \Rightarrow v \in P$

UR3 $\hat{u}^\frown\langle(\alpha, \bullet)\rangle \in P \Rightarrow \hat{u}^\frown\langle(\alpha, \emptyset)\rangle \in P$

UR4 $(A \neq \bullet \wedge \hat{u}^\frown\langle(\alpha, A)\rangle^\frown\check{v} \in P \wedge \hat{u}^\frown\langle(\alpha, A), a, ([], \bullet)\rangle \notin P) \Rightarrow$
$\hat{u}^\frown\langle(\alpha, A \cup \{a\})\rangle^\frown\check{v} \in P$

UR5 $(A \neq \bullet \wedge \hat{u}^\frown\langle(\alpha, A), a, ([], \bullet)\rangle \in P) \Rightarrow$
$\hat{u}^\frown\langle(\alpha, A), tock, ([], \bullet), a, ([], \bullet)\rangle \in P$

UR6 $\hat{u}^\frown\langle(\alpha, A), a\rangle^\frown v \in P \Rightarrow a \notin^* A$

UR7 $\forall k \in \mathbb{N} \centerdot \exists n \in \mathbb{N} \centerdot (u \in P \wedge \sharp(\mathsf{trace}(u) \upharpoonright tock) \leqslant k) \Rightarrow$
$(\sharp\mathsf{trace}(u) \leqslant n \wedge (\hat{u}^\frown\langle(\alpha, \bullet)\rangle \in P \Rightarrow \exists \alpha' \centerdot \alpha \sim \alpha' \wedge \sharp\alpha' \leqslant n))$

UR8 $u^\frown\langle\checkmark\rangle^\frown v \in P \Rightarrow \mathsf{trace}(v) \leq \langle tock\rangle^\infty$

UR9 $\langle(\alpha_0, A_0), a_1, (\alpha_1, A_1), \ldots, a_k, (\alpha_k, A_k)\rangle \in P \Rightarrow$
$\exists (A_i' \in REF)_{i=1}^k \centerdot \langle(\alpha_0, A_0'), a_1, (\alpha_1, A_1'), \ldots, a_k, (\alpha_k, A_k')\rangle \in P \wedge$
$\forall (1 \leqslant i \leqslant k) \centerdot A_i \subseteq^* A_i' \wedge \alpha_i \prec [|A_i|]$.

We give short intuitive explanations of each of these axioms; as expected, the first eight are very closely related to the corresponding axioms of $\mathcal{M}_R$ and, to a looser extent, $\mathcal{M}_{TF}$.

UR1: A basic non-emptiness condition specifying the least observation of any process. Corresponds to Axioms *R1* and *TF1*.

UR2: Postulates the downward-closed nature of observations. Note that this axiom entails that any stable refusal could also have been recorded as an unstable one. Corresponds to Axioms *R2* and *TF2*.

UR3: States that all processes eventually stabilise, and moreover, that it is always possible to reach stability before the next event (in particular *tock*) occurs. One interpretation of this axiom is that divergences are excluded. This axiom corresponds to Axiom *R3*, but has no counterpart in $\mathcal{M}_{TF}$.

UR4: Tells us how observations can be extended. Essentially, any event that could not have been performed must have been refusable. Corresponds to Axioms *R4* and *TF3*.

UR5: States that events which can stably occur at the end of some test cannot be prohibited from occurring after a further *tock*. Corresponds to Axiom *R5* of $\mathcal{M}_R$. $\mathcal{M}_{TF}$, as discussed where *R5* was introduced, exhibits a somewhat similar property via Axiom *TF3*.

UR6: Stipulates that if an event is found to be stably refused, then it cannot immediately thereafter be performed. Corresponds to Axiom *R6* of $\mathcal{M}_R$. $\mathcal{M}_{TF}$, being a dense model, cannot entertain such a concept.

UR7: Finite variability, corresponding to Axioms *R7* and *TF4*. However, Axiom *UR7* improves upon *R7* in that it prescribes a bound on the number of effectively distinct unstable refusals which may be recorded within a single unstable refusal sequence. In $\mathcal{M}_{TF}$, this property is a consequence of Axiom *TF3*.

UR8: Stipulates that termination in effect ends the progress of a process. Corresponds to Axioms *R8* and *TF5*.

UR9: Asserts that, prior to the occurrence of the first *tock*, any unstable refusal information is redundant. This reflects the principle that robust stability can only be achieved as a result of strictly positive prior delays; it also justifies our claim that all 'fundamental laws', such as $P \sqcap P = P$, involving the resolution of some choices prior to the first occurrence of a *tock*, continue to hold in $\mathcal{M}_{UR}$. Naturally, Axiom *UR9* has no counterpart in either $\mathcal{M}_R$ or $\mathcal{M}_{TF}$, since the former does not have a notion of unstable refusal, and the latter draws no distinction between unstable and stable refusals.

Note that, as in $\mathcal{M}_R$, *tock* may not, by definition, belong to (unstable or stable) refusals.[2] It therefore again follows that $\mathcal{M}_{UR}$ processes are free of timestops and signals, as the following proposition indicates.

[2]An interesting alternative could have been to allow *tock*s to belong to *unstable* refusals. This would have yielded yet more information about processes since an observed refusal of *tock* would imply instability. On the other hand, given that our avowed goal in designing $\mathcal{M}_{UR}$ was to obtain a satisfactory timed failure soundness result—something which we achieve in the form of Theorem 8.1—it makes sense to stick with the more abstract implementation.

Proposition 7.1 *Let $P \in \mathcal{M}_{UR}$ be a process, and let $w \in P$ be a test of P. For any $k \geqslant 0$, we have $w^\frown\langle tock, ([], \emptyset)\rangle^k \in P$.*

The proof is entirely similar to that of Proposition 5.1, and is therefore omitted.

If $u = \langle(\alpha_0, A_0), a_1, (\alpha_2, A_2), \ldots, a_k, (\alpha_k, A_k)\rangle$ is a test, let $\mathsf{refusals}(u) \mathrel{\widehat{=}} \bigcup_{i=0}^{k}\{|A_i| \cup \bigcup \alpha_i\}$. In other words, $\mathsf{refusals}(u)$ consists of the set of all events which are at some point either stably or unstably refused in u.

Define an auxiliary function $(\cdot) \underset{B}{\parallel} (\cdot) : TEST_U \times TEST_U \longrightarrow \mathcal{P}(TEST_U)$. Here $B \subseteq \Sigma^{\checkmark}$ and $a, c \in \Sigma_{tock}$. Let us write $B_{\mathrm{t}} = B \cup \{tock\}$. We proceed inductively on both u and u' (in defining $u \underset{B}{\parallel} u'$), using R and S to stand for refusal pairs.

$$
\begin{array}{rcll}
\begin{array}{r}\langle([A_1, A_2, \ldots, A_k], A)\rangle \\ \langle([C_1, C_2, \ldots, C_{k'}], C)\rangle\end{array} \underset{B}{\parallel} & \widehat{=} & \{\langle([D_1, D_2, \ldots, D_{k'}], D)\rangle \mid & \\
&& \quad (\forall(1 \leqslant i \leqslant k)\bullet & \\
&& \quad D_i \subseteq (B \cap A_i) \cup & \\
&& \quad (B \cap C_i) \cup (A_i \cap C_i)) \wedge & \\
&& \quad (\forall(k+1 \leqslant j \leqslant k')\bullet & \\
&& \quad D_j \subseteq (B \cap |A|) \cup & \\
&& \quad (B \cap C_j) \cup (|A| \cap C_j)) \wedge & \\
&& \quad D \subseteq^* (B \cap^* A) \cup^* & \\
&& \quad (B \cap^* C) \cup^* (A \cap^* C)\} & \text{if } 0 \leqslant k \leqslant k' \\
& \widehat{=} & \left\{\begin{array}{l}\langle([C_1, C_2, \ldots, C_{k'}], C)\rangle \\ \langle([A_1, A_2, \ldots, A_k], A)\rangle\end{array}\right. \underset{B}{\parallel} & \text{if } 0 \leqslant k' < k \\
\langle R, a\rangle^\frown u \underset{B}{\parallel} \langle S\rangle & \widehat{=} & \langle R\rangle \underset{B}{\parallel} \langle S\rangle & \text{if } a \in B_{\mathrm{t}} \\
& \widehat{=} & (\langle R\rangle \underset{B}{\parallel} \langle S\rangle)\{\langle a\rangle\}(u \underset{B}{\parallel} \langle S\rangle) & \text{if } a \notin B_{\mathrm{t}} \\
\langle R, \checkmark\rangle^\frown u \underset{B}{\parallel} \langle S\rangle & \widehat{=} & \langle R\rangle \underset{B}{\parallel} \langle S\rangle & \text{if } \checkmark \in B_{\mathrm{t}} \\
& \widehat{=} & (\langle R\rangle \underset{B}{\parallel} \langle S\rangle)\{\langle\checkmark\rangle^\frown u\} & \text{if } \checkmark \notin B_{\mathrm{t}} \\
\langle R\rangle \underset{B}{\parallel} \langle S, c\rangle^\frown u & \widehat{=} & \langle S, c\rangle^\frown u \underset{B}{\parallel} \langle R\rangle & \\
\langle R\rangle \underset{B}{\parallel} \langle S, \checkmark\rangle^\frown u & \widehat{=} & \langle S, \checkmark\rangle^\frown u \underset{B}{\parallel} \langle R\rangle &
\end{array}
$$

$$
\begin{array}{rcll}
\langle R,a\rangle^\frown u_1 \underset{B}{\parallel} \langle S,c\rangle^\frown u_2 & \mathrel{\widehat{=}} & ((\langle R\rangle \underset{B}{\parallel} \langle S\rangle)\{\langle a\rangle\}(u_1 \underset{B}{\parallel} u_2) & \text{if } a=c\in B_{\mathrm{t}} \\
 & \mathrel{\widehat{=}} & \langle R\rangle \underset{B}{\parallel} \langle S\rangle & \text{if } a,c\in B_{\mathrm{t}} \wedge a\neq c \\
 & \mathrel{\widehat{=}} & ((\langle R\rangle \underset{B}{\parallel} \langle S\rangle)\{\langle c\rangle\} & \\
 & & \quad(\langle R,a\rangle^\frown u_1 \underset{B}{\parallel} u_2) & \text{if } a\in B_{\mathrm{t}} \wedge c\notin B_{\mathrm{t}} \\
 & \mathrel{\widehat{=}} & ((\langle R\rangle \underset{B}{\parallel} \langle S\rangle)\{\langle a\rangle\} & \\
 & & \quad(u_1 \underset{B}{\parallel} \langle S,c\rangle^\frown u_2) & \text{if } a\notin B_{\mathrm{t}} \wedge c\in B_{\mathrm{t}} \\
 & \mathrel{\widehat{=}} & ((\langle R\rangle \underset{B}{\parallel} \langle S\rangle)\{\langle a\rangle\} & \\
 & & \quad(u_1 \underset{B}{\parallel} \langle S,c\rangle^\frown u_2)\cup & \\
 & & ((\langle R\rangle \underset{B}{\parallel} \langle S\rangle)\{\langle c\rangle\} & \\
 & & \quad(\langle R,a\rangle^\frown u_1 \underset{B}{\parallel} u_2) & \text{if } a,c\notin B_{\mathrm{t}} \\
\langle R,\checkmark\rangle^\frown u_1 \underset{B}{\parallel} \langle S,\checkmark\rangle^\frown u_2 & \mathrel{\widehat{=}} & ((\langle R\rangle \underset{B}{\parallel} \langle S\rangle)\{\langle \checkmark\rangle^\frown u_1\} & \\
\langle R,\checkmark\rangle^\frown u_1 \underset{B}{\parallel} \langle S,c\rangle^\frown u_2 & \mathrel{\widehat{=}} & \langle R\rangle \underset{B}{\parallel} \langle S\rangle & \text{if } \checkmark\in B_{\mathrm{t}} \wedge c\in B_{\mathrm{t}} \\
 & \mathrel{\widehat{=}} & ((\langle R\rangle \underset{B}{\parallel} \langle S\rangle)\{\langle c\rangle\} & \\
 & & \quad(\langle R,\checkmark\rangle^\frown u_1 \underset{B}{\parallel} u_2) & \text{if } \checkmark\in B_{\mathrm{t}} \wedge c\notin B_{\mathrm{t}} \\
 & \mathrel{\widehat{=}} & ((\langle R\rangle \underset{B}{\parallel} \langle S\rangle)\{\langle \checkmark\rangle^\frown u_1\} & \text{if } \checkmark\notin B_{\mathrm{t}} \wedge c\in B_{\mathrm{t}} \\
 & \mathrel{\widehat{=}} & ((\langle R\rangle \underset{B}{\parallel} \langle S\rangle)\{\langle \checkmark\rangle^\frown u_1\}\cup & \\
 & & ((\langle R\rangle \underset{B}{\parallel} \langle S\rangle)\{\langle c\rangle\} & \\
 & & \quad(\langle R,\checkmark\rangle^\frown u_1 \underset{B}{\parallel} u_2) & \text{if } \checkmark\notin B_{\mathrm{t}} \wedge c\notin B_{\mathrm{t}} \\
\langle R,a\rangle^\frown u_1 \underset{B}{\parallel} \langle S,\checkmark\rangle^\frown u_2 & \mathrel{\widehat{=}} & \langle S,\checkmark\rangle^\frown u_2 \underset{B}{\parallel} \langle R,a\rangle^\frown u_1. &
\end{array}
$$

Likewise, we define the auxiliary function $(\cdot)\,|||\,(\cdot) : \mathit{TEST}_U \times \mathit{TEST}_U \longrightarrow \mathcal{P}(\mathit{TEST}_U)$. Here $a,c\in\Sigma$. We proceed in a manner similar to the above:

$$
\begin{array}{rcll}
\begin{array}{r}\langle([A_1, A_2, \ldots, A_k], A)\rangle \\ \langle([C_1, C_2, \ldots, C_{k'}], C)\rangle\end{array} ||| & \mathrel{\widehat=} & \{\langle([D_1, D_2, \ldots, D_{k'}], D)\rangle \mid & \\
& & \quad (\forall(1 \leqslant i \leqslant k)\bullet & \\
& & \quad D_i \subseteq A_i \cap C_i) \wedge & \\
& & \quad (\forall(k+1 \leqslant j \leqslant k')\bullet & \\
& & \quad D_j \subseteq |A| \cap C_j) \wedge & \\
& & \quad D \subseteq^* A \cap^* C\} & \text{if } 0 \leqslant k \leqslant k' \\
& \mathrel{\widehat=} & \left\{ \begin{array}{l}\langle([C_1, C_2, \ldots, C_{k'}], C)\rangle \\ \langle([A_1, A_2, \ldots, A_k], A)\rangle\end{array} ||| \right. & \text{if } 0 \leqslant k' < k \\
\langle R, a\rangle^\frown u \; ||| \; \langle S\rangle & \mathrel{\widehat=} & (\langle R\rangle \; ||| \; \langle S\rangle)\{\langle a\rangle\}(u \; ||| \; \langle S\rangle) & \\
\langle R, \checkmark\rangle^\frown u \; ||| \; \langle S\rangle & \mathrel{\widehat=} & (\langle R\rangle \; ||| \; \langle S\rangle)\{\langle\checkmark\rangle^\frown u\} & \\
\langle R, tock\rangle^\frown u \; ||| \; \langle S\rangle & \mathrel{\widehat=} & \langle R\rangle \; ||| \; \langle S\rangle & \\
\langle R\rangle \; ||| \; \langle S, c\rangle^\frown u & \mathrel{\widehat=} & \langle S, c\rangle^\frown u \; ||| \; \langle R\rangle & \\
\langle R\rangle \; ||| \; \langle S, \checkmark\rangle^\frown u & \mathrel{\widehat=} & \langle S, \checkmark\rangle^\frown u \; ||| \; \langle R\rangle & \\
\langle R\rangle \; ||| \; \langle S, tock\rangle^\frown u & \mathrel{\widehat=} & \langle S, tock\rangle^\frown u \; ||| \; \langle R\rangle & \\
\langle R, a\rangle^\frown u_1 \; ||| \; \langle S, c\rangle^\frown u_2 & \mathrel{\widehat=} & (\langle R\rangle \; ||| \; \langle S\rangle)\{\langle a\rangle\} & \\
& & \quad (u_1 \; ||| \; \langle S, c\rangle^\frown u_2) \cup & \\
& & (\langle R\rangle \; ||| \; \langle S\rangle)\{\langle c\rangle\} & \\
& & \quad (\langle R, a\rangle^\frown u_1 \; ||| \; u_2) & \\
\langle R, a\rangle^\frown u_1 \; ||| \; \langle S, \checkmark\rangle^\frown u_2 & \mathrel{\widehat=} & (\langle R\rangle \; ||| \; \langle S\rangle)\{\langle a\rangle\} & \\
& & \quad (u_1 \; ||| \; \langle S, \checkmark\rangle^\frown u_2) \cup & \\
& & (\langle R\rangle \; ||| \; \langle S\rangle)\{\langle\checkmark\rangle^\frown u_2\} & \\
\langle R, a\rangle^\frown u_1 \; ||| \; \langle S, tock\rangle^\frown u_2 & \mathrel{\widehat=} & (\langle R\rangle \; ||| \; \langle S\rangle)\{\langle a\rangle\} & \\
& & \quad (u_1 \; ||| \; \langle S, tock\rangle^\frown u_2) & \\
\langle R, \checkmark\rangle^\frown u_1 \; ||| \; \langle S, \checkmark\rangle^\frown u_2 & \mathrel{\widehat=} & (\langle R\rangle \; ||| \; \langle S\rangle)\{\langle\checkmark\rangle^\frown u_1\} & \\
\langle R, \checkmark\rangle^\frown u_1 \; ||| \; \langle S, tock\rangle^\frown u_2 & \mathrel{\widehat=} & (\langle R\rangle \; ||| \; \langle S\rangle)\{\langle\checkmark\rangle^\frown u_1\} & \\
\langle R, tock\rangle^\frown u_1 \; ||| \; \langle S, tock\rangle^\frown u_2 & \mathrel{\widehat=} & (\langle R\rangle \; ||| \; \langle S\rangle)\{\langle tock\rangle\}(u_1 \; ||| \; u_2) & \\
\langle R, \checkmark\rangle^\frown u_1 \; ||| \; \langle S, c\rangle^\frown u_2 & \mathrel{\widehat=} & \langle S, c\rangle^\frown u_2 \; ||| \; \langle R, \checkmark\rangle^\frown u_1 & \\
\langle R, tock\rangle^\frown u_1 \; ||| \; \langle S, c\rangle^\frown u_2 & \mathrel{\widehat=} & \langle S, c\rangle^\frown u_2 \; ||| \; \langle R, tock\rangle^\frown u_1 & \\
\langle R, tock\rangle^\frown u_1 \; ||| \; \langle S, \checkmark\rangle^\frown u_2 & \mathrel{\widehat=} & \langle S, \checkmark\rangle^\frown u_2 \; ||| \; \langle R, tock\rangle^\frown u_1. &
\end{array}
$$

We now aim to define a hiding operator on tests; to this end we first define it on unstable refusal sequences:

$$\begin{aligned}
[] \setminus A &\mathrel{\widehat{=}} [] \\
([B]^\frown\beta) \setminus A &\mathrel{\widehat{=}} [B]^\frown(\beta \setminus A) \quad \text{if } A \subseteq B \\
&\mathrel{\widehat{=}} [\emptyset]^\frown(\beta \setminus A) \quad \text{if } A \not\subseteq B.
\end{aligned}$$

Next, we define the following glueing operator, which pastes an unstable refusal sequence to the left of a test:

$$\begin{aligned}
&\beta \mathrel{\mathsf{glue}} \langle(\alpha_0, A_0), a_1, (\alpha_1, A_1), \ldots, a_k, (\alpha_k, A_k)\rangle \mathrel{\widehat{=}} \\
&\quad \langle(\beta^\frown\alpha_0, A_0), a_1, (\alpha_1, A_1), \ldots, a_k, (\alpha_k, A_k)\rangle.
\end{aligned}$$

Finally,

$$\begin{aligned}
\langle(\beta, B)\rangle \setminus A &\mathrel{\widehat{=}} \langle(\beta \setminus A, B)\rangle && \text{if } A \subseteq B \\
&\mathrel{\widehat{=}} \langle(\beta \setminus A, \bullet)\rangle && \text{if } A \not\subseteq B \\
(\langle(\beta, B), a\rangle^\frown u) \setminus A &\mathrel{\widehat{=}} (\beta \setminus A) \mathrel{\mathsf{glue}} (u \setminus A) && \text{if } a \in A \\
&\mathrel{\widehat{=}} (\langle(\beta, B)\rangle \setminus A)^\frown\langle a\rangle^\frown(u \setminus A) && \text{if } a \notin A.
\end{aligned}$$

Let $A \subseteq \Sigma^{\checkmark}$. A test u is *A-urgent* if, whenever $\langle(\alpha, B), tock\rangle$ in u, then $A \subseteq B$. If $A = \{a\}$ is a singleton, we write a-urgent instead of $\{a\}$-urgent.

We define a function $\mathsf{RefCl} : TEST_U \longrightarrow \mathcal{P}(TEST_U)$, as follows. For u a test, we let $v \in \mathsf{RefCl}(u)$ if $v \prec u \wedge \mathsf{trace}(v) = \mathsf{trace}(u)$. We extend the definition of RefCl to sets of tests by setting, for $P \subseteq TEST_U$, $\mathsf{RefCl}(P) \mathrel{\widehat{=}} \bigcup\{\mathsf{RefCl}(u) \mid u \in P\}$.

If $f : \Sigma \longrightarrow \Sigma$ is a renaming function and $\beta = [B_1, B_2, \ldots, B_l] \in URS$, we let $f(\beta) \mathrel{\widehat{=}} [f(B_1), f(B_2), \ldots, f(B_l)]$, where $f(\checkmark)$ is defined to be $\checkmark$. Next, for $(\beta, B) \in RPAIR$, we let $f(\beta, B) \mathrel{\widehat{=}} (f(\beta), f(B))$, where $f(\bullet) \mathrel{\widehat{=}} \bullet$. Lastly, if $u = \langle R_0, a_1, R_1, \ldots, a_k, R_k\rangle \in TEST_U$, we define $f(u) \mathrel{\widehat{=}} \langle f(R_0), f(a_1), f(R_1), \ldots, f(a_k), f(R_k)\rangle$, where $f(tock) \mathrel{\widehat{=}} tock$.

Semantic bindings are functions $\rho : VAR \longrightarrow \mathcal{M}_{UR}$. We have the usual substitution operations on bindings, such as $\rho[X := P]$ (with $X \in VAR$, $P \in \mathcal{M}_{UR}$), etc.

For $F : \mathcal{M}_{UR} \longrightarrow \mathcal{M}_{UR}$, we let $\mathsf{fix}(F)$ denote the unique fixed point of F. Should such a thing not exist, we let $\mathsf{fix}(F)$ denote an arbitrary, but fixed, element of $\mathcal{M}_{UR}$.

We now define the function $\mathcal{R}_U[\![\cdot]\!]$ inductively over the structure of Timed CSP terms.

$$\begin{array}{rcl}
\mathcal{R}_U[\![STOP]\!]\rho & \mathrel{\widehat{=}} & \{u \mid \mathsf{trace}(u) \leq \langle tock\rangle^\infty\} \\
\mathcal{R}_U[\![SKIP]\!]\rho & \mathrel{\widehat{=}} & \{u \mid \mathsf{trace}(u) \leq \langle tock\rangle^\infty \wedge \checkmark \notin \mathsf{refusals}(u)\} \cup \\
 & & \{u ^\frown \langle \checkmark \rangle ^\frown v \mid \mathsf{trace}(u) \leq \langle tock\rangle^\infty \wedge \\
 & & \quad \checkmark \notin \mathsf{refusals}(u) \wedge \mathsf{trace}(v) \leq \langle tock\rangle^\infty\} \\
\mathcal{R}_U[\![WAIT\ 0]\!]\rho & \mathrel{\widehat{=}} & \mathcal{R}_U[\![SKIP]\!]\rho \\
\mathcal{R}_U[\![WAIT\ n]\!]\rho & \mathrel{\widehat{=}} & \{u \mid 0 \leqslant k < n \wedge \mathsf{trace}(u) = \langle tock\rangle^k\} \cup \\
(n \geqslant 1) & & \{\hat{u} ^\frown \langle (\alpha, A)\rangle ^\frown \check{v} \mid \mathsf{trace}(u) = \langle tock\rangle^n \wedge \\
 & & \quad \mathsf{trace}(v) \leq \langle tock\rangle^\infty \wedge \checkmark \notin^* A \wedge \\
 & & \quad \checkmark \notin \mathsf{refusals}(v)\} \cup \\
 & & \{\hat{u} ^\frown \langle (\alpha, A)\rangle ^\frown \check{v} ^\frown \langle \checkmark \rangle ^\frown w \mid \mathsf{trace}(u) = \langle tock\rangle^n \wedge \\
 & & \quad \mathsf{trace}(v) \leq \langle tock\rangle^\infty \wedge \checkmark \notin^* A \wedge \checkmark \notin \mathsf{refusals}(v) \wedge \\
 & & \quad \mathsf{trace}(w) \leq \langle tock\rangle^\infty\} \\
\mathcal{R}_U[\![P_1 \overset{0}{\rhd} P_2]\!]\rho & \mathrel{\widehat{=}} & \{\langle ([], \bullet)\rangle ^\frown \check{u} \mid \langle tock\rangle \not\leq \mathsf{trace}(u) \wedge u \in \mathcal{R}_U[\![P_1]\!]\rho\} \cup \\
 & & \mathcal{R}_U[\![P_2]\!]\rho \\
\mathcal{R}_U[\![P_1 \overset{n}{\rhd} P_2]\!]\rho & \mathrel{\widehat{=}} & \{u \mid \langle tock\rangle^n \not\leq \mathsf{trace}(u) \wedge u \in \mathcal{R}_U[\![P_1]\!]\rho\} \cup \\
(n \geqslant 1) & & \{\hat{u} ^\frown \langle (\alpha, \bullet)\rangle ^\frown \check{v} \mid \mathsf{trace}(u) = \langle tock\rangle^n \wedge \\
 & & \quad \langle tock\rangle \not\leq \mathsf{trace}(v) \wedge \hat{u} ^\frown \langle (\alpha, \bullet)\rangle ^\frown \check{v} \in \mathcal{R}_U[\![P_1]\!]\rho\} \cup \\
 & & \{\hat{u} ^\frown \langle (\alpha ^\frown \beta, A)\rangle ^\frown \check{v} \mid \mathsf{trace}(u) = \langle tock\rangle^n \wedge \\
 & & \quad \hat{u} ^\frown \langle (\alpha, \bullet)\rangle \in \mathcal{R}_U[\![P_1]\!]\rho \wedge \langle (\beta, A)\rangle ^\frown \check{v} \in \mathcal{R}_U[\![P_2]\!]\rho\} \\
\mathcal{R}_U[\![a \longrightarrow P]\!]\rho & \mathrel{\widehat{=}} & \{u \mid \mathsf{trace}(u) \leq \langle tock\rangle^\infty \wedge a \notin \mathsf{refusals}(u)\} \cup \\
 & & \{u ^\frown \langle a\rangle ^\frown v \mid \mathsf{trace}(u) \leq \langle tock\rangle^\infty \wedge \\
 & & \quad a \notin \mathsf{refusals}(u) \wedge v \in \mathcal{R}_U[\![P]\!]\rho\} \\
\mathcal{R}_U[\![a : A \longrightarrow P(a)]\!]\rho & \mathrel{\widehat{=}} & \{u \mid \mathsf{trace}(u) \leq \langle tock\rangle^\infty \wedge A \cap \mathsf{refusals}(u) = \emptyset\} \cup \\
 & & \{u ^\frown \langle a\rangle ^\frown v \mid a \in A \wedge \mathsf{trace}(u) \leq \langle tock\rangle^\infty \wedge \\
 & & \quad A \cap \mathsf{refusals}(u) = \emptyset \wedge v \in \mathcal{R}_U[\![P(a)]\!]\rho\} \\
\mathcal{R}_U[\![P_1 \mathbin{\Box} P_2]\!]\rho & \mathrel{\widehat{=}} & \{u \mid \mathsf{trace}(u) \leq \langle tock\rangle^\infty \wedge \\
 & & \quad u \in \mathcal{R}_U[\![P_1]\!]\rho \cap \mathcal{R}_U[\![P_2]\!]\rho\} \cup \\
 & & \{u ^\frown \langle a\rangle ^\frown v \mid \mathsf{trace}(u) \leq \langle tock\rangle^\infty \wedge a \neq tock \wedge \\
 & & \quad u ^\frown \langle a\rangle ^\frown v \in \mathcal{R}_U[\![P_1]\!]\rho \cup \mathcal{R}_U[\![P_2]\!]\rho \wedge \\
 & & \quad u \in \mathcal{R}_U[\![P_1]\!]\rho \cap \mathcal{R}_U[\![P_2]\!]\rho\} \\
\mathcal{R}_U[\![P_1 \sqcap P_2]\!]\rho & \mathrel{\widehat{=}} & \mathcal{R}_U[\![P_1]\!]\rho \cup \mathcal{R}_U[\![P_2]\!]\rho \\
\mathcal{R}_U[\![P_1 \underset{B}{\parallel} P_2]\!]\rho & \mathrel{\widehat{=}} & \{u \mid \exists u_1, u_2 \bullet u \in u_1 \underset{B}{\parallel} u_2 \wedge \\
 & & \quad u_1 \in \mathcal{R}_U[\![P_1]\!]\rho \wedge u_2 \in \mathcal{R}_U[\![P_2]\!]\rho\}
\end{array}$$

$$\begin{array}{rcl}
\mathcal{R}_U[\![P_1 \mid\mid\mid P_2]\!]\rho & \widehat{=} & \{u \mid \exists\, u_1, u_2 \centerdot u \in u_1 \mid\mid\mid u_2 \wedge \\
& & \quad u_1 \in \mathcal{R}_U[\![P_1]\!]\rho \wedge u_2 \in \mathcal{R}_U[\![P_2]\!]\rho\} \\
\mathcal{R}_U[\![P_1 \,;\, P_2]\!]\rho & \widehat{=} & \mathsf{RefCl}(\{u \setminus \checkmark \mid \mathsf{trace}(u) \upharpoonright \checkmark = \langle\rangle \wedge \\
& & \quad u \text{ is } \checkmark\text{-urgent} \wedge u \in \mathcal{R}_U[\![P_1]\!]\rho\}) \cup \\
& & \mathsf{RefCl}(\{\widehat{(u_1 \setminus \checkmark)}{}^\frown u_2 \mid \mathsf{trace}(u_1) \upharpoonright \checkmark = \langle\rangle \wedge \\
& & \quad u_1 \text{ is } \checkmark\text{-urgent} \wedge u_1{}^\frown\langle \checkmark, ([], \bullet)\rangle \in \mathcal{R}_U[\![P_1]\!]\rho \wedge \\
& & \quad u_2 \in \mathcal{R}_U[\![P_2]\!]\rho\}) \\
\mathcal{R}_U[\![P \setminus A]\!]\rho & \widehat{=} & \mathsf{RefCl}(\{u \setminus A \mid u \text{ is } A\text{-urgent} \wedge u \in \mathcal{R}_U[\![P]\!]\rho\}) \\
\mathcal{R}_U[\![f^{-1}(P)]\!]\rho & \widehat{=} & \{u \mid f(u) \in \mathcal{R}_U[\![P]\!]\rho\} \\
\mathcal{R}_U[\![f(P)]\!]\rho & \widehat{=} & \{\langle R_0, f(a_1), R_1, \ldots, f(a_k), R_k\rangle \mid k \geqslant 0 \wedge \\
& & \quad \langle f^{-1}(R_0), a_1, f^{-1}(R_1), \ldots, a_k, f^{-1}(R_k)\rangle \in \\
& & \quad \mathcal{R}_U[\![P]\!]\rho\} \\
\mathcal{R}_U[\![X]\!]\rho & \widehat{=} & \rho(X) \\
\mathcal{R}_U[\![\mu\, X \centerdot P]\!]\rho & \widehat{=} & \mathsf{fix}(\lambda\, x.\mathcal{R}_U[\![P]\!](\rho[X := x])).
\end{array}$$

Proposition 7.2 *For any term $P \in$ **TCSP**, and any semantic binding ρ, $\mathcal{R}_U[\![P]\!]\rho \in \mathcal{M}_{UR}$. Moreover, if P is a $\overline{\textbf{TCSP}}$ program, then $\mathcal{R}_U[\![P]\!]\rho = \mathcal{R}_U[\![P]\!]\rho'$ for any semantic binding ρ'.*

Proof (Sketch.) The proof of the first part, by structural induction on P, proceeds in almost identical manner as in the case of $\mathcal{M}_R$, since the Axioms *UR1–UR8* are very similar to Axioms *R1–R8*. Axiom *UR9*, on the other hand, poses no difficulties whatsoever. The only part worth examining is the case of unstable refusals in Axiom *UR7* (finite variability), in the context of parallel composition and hiding. The result follows from the fact that both these operators preserve the $\sim$ relation when specialised to unstable refusal sequences. ■

The proof of the next proposition can be adapted almost verbatim from the corresponding results in $\mathcal{M}_R$; a complete ultrametric d is defined on $\mathcal{M}_{UR}$, **TCSP** terms correspond to non-expanding functions, and terms which are time-guarded correspond to contractions. We omit the details.

Proposition 7.3 *Recursions have unique fixed points in $\mathcal{M}_{UR}$.*

Let us make the connection between $\mathcal{M}_{UR}$ and $\mathcal{M}_R$ more explicit. We need to define a projection mapping $\Pi : \mathit{TEST}_U \longrightarrow \mathit{TEST}$ by dropping all unstable refusals, as follows: $\Pi(\langle(\alpha_0, A_0), a_1, (\alpha_1, A_1), \ldots, a_k, (\alpha_k, A_k)\rangle) \mathrel{\widehat{=}} \langle A_0, a_1, A_1, \ldots, a_k, A_k\rangle$. Π extends naturally to sets of tests.

Proposition 7.4 *For any $P \in \mathcal{M}_{UR}$, $\Pi(P) \in \mathcal{M}_R$. Moreover, Π is non-expanding (hence continuous). Lastly, for $P \in \overline{\mathbf{TCSP}}$, $\Pi(\mathcal{R}_U[\![P]\!]) = \mathcal{R}[\![P]\!]$.*

Proof (Sketch.) The first part follows trivially from Axioms *UR1–UR8*. The second part is clear, whereas the third part is easily shown by structural induction. ■

As expected, however, $\mathcal{M}_{UR}$ distinguishes strictly more programs than $\mathcal{M}_R$.

Proposition 7.5 *There exist programs $P, Q \in \overline{\mathbf{TCSP}}$ such that $\mathcal{R}_U[\![P]\!] \neq \mathcal{R}_U[\![Q]\!]$ but $\mathcal{R}[\![P]\!] = \mathcal{R}[\![Q]\!]$.*

Proof Consider $P = (\bar{a} \;\Box\; \bar{b}) \overset{1}{\rhd} STOP$ and $Q = (\bar{a} \overset{1}{\rhd} STOP) \;\Box\; (\bar{b} \overset{1}{\rhd} STOP)$. A simple inspection reveals that $\mathcal{R}[\![P]\!] = \mathcal{R}[\![Q]\!]$. But if $u = \langle([], \bullet), tock, ([\{b\}], \bullet), a, ([], \bullet)\rangle$, it is easy to see that $u \in \mathcal{R}_U[\![Q]\!]$ yet $u \notin \mathcal{R}_U[\![P]\!]$. ■

7.2 Operational Semantics

We now present an operational semantics and show it to be congruent to the denotational semantics of the previous section. Once again, the format is very similar to that of Section 5.2.

In order to capture operationally the twin notions of fleeting and robust stability, we need to introduce two different kinds of silent events: τ will stand for 'fleeting' silent events, i.e., those which cannot be delayed at all, whereas τ^* will stand for 'robust' silent events, allowed to remain on offer until just before the following *tock*. The availability of either event makes a state unstable, fleetingly in the case of τ, robustly in the case of τ^*. We use 'variables' ? and ¿ to uniformly denote the presence or absence of the superscript $*$ on select objects appearing in a given equation. Thus $\tau^?$, for instance, can stand for either τ or τ^*, the choice being uniform over the equation in which it appears.

We also need to define fleeting and robust versions of the nodes $WAIT\ 0$ and $P_1 \overset{0}{\rhd} P_2$. Thus $WAIT\ 0$ and $P_1 \overset{0}{\rhd} P_2$ will stand for the fleetingly stable versions of these nodes, identified with the programs by the same syntax, whereas $WAIT^*\ 0$ and $P_1 \overset{0}{\rhd}{}^* P_2$ will denote their robustly stable counterparts. Naturally, we will have, for example, $WAIT\ 0 \xrightarrow{\tau} SKIP$ and $WAIT^*\ 0 \xrightarrow{\tau^*} SKIP$, which we express more concisely as $WAIT^?\ 0 \xrightarrow{\tau^?} SKIP$. In order to ease the syntax, we will also carry in general the nodes $WAIT\ n$, $WAIT^*\ n$, $P_1 \overset{n}{\rhd} P_2$, and $P_1 \overset{n}{\rhd}{}^* P_2$.

Thus our set $NODE_{UR}$ of *(open) nodes* is generated by Timed CSP syntax together with the above additions, and without any well-timedness requirements. $\overline{NODE}_{UR}$ naturally represents the subset of *(closed) nodes.* As usual, these sets are abbreviated $NODE$ and $\overline{NODE}$ if no confusion is likely. The remainder of our conventions on Timed CSP, as listed in Section 2.1, apply. We again insist that our inference rules only apply to closed nodes.

Other notational conventions are very similar to those of Section 5.2: a and b represent visible non-*tock* event, i.e., member of $\Sigma^{\checkmark}$. μ can be a visible non-*tock* event or a silent one ($\mu \in \Sigma^{\checkmark} \cup \{\tau, \tau^*\}$), and x can be a μ or a *tock*. $P \xrightarrow{x} P'$ means that the node P can perform an immediate *x-transition*, and become P' (communicating x in the process if x is a visible event). $P \overset{x}{\not\longrightarrow}$ means that P cannot possibly do an x.

The transition rules are as follows:

$$\frac{}{STOP \xrightarrow{tock} STOP} \tag{7.1}$$

$$\frac{}{SKIP \xrightarrow{tock} SKIP} \tag{7.2}$$

$$\frac{}{SKIP \xrightarrow{\checkmark} STOP} \tag{7.3}$$

$$\frac{}{WAIT^?\ n \xrightarrow{tock} WAIT^*\ (n-1)}\ [\, n \geqslant 1\,] \tag{7.4}$$

$$\frac{}{WAIT^{?}\ 0 \xrightarrow{\tau^{?}} SKIP} \tag{7.5}$$

$$\frac{P_1 \xrightarrow{tock} P_1'}{P_1 \overset{n}{\rhd}^{?}\ P_2 \xrightarrow{tock} P_1' \overset{n-1}{\rhd}^{*}\ P_2}\ [\,n \geqslant 1\,] \tag{7.6}$$

$$\frac{}{P_1 \overset{0}{\rhd}^{?}\ P_2 \xrightarrow{\tau^{?}} P_2} \tag{7.7}$$

$$\frac{P_1 \xrightarrow{\tau^{?}} P_1'}{P_1 \overset{n}{\rhd}^{¿}\ P_2 \xrightarrow{\tau^{?}} P_1' \overset{n}{\rhd}^{¿}\ P_2} \tag{7.8}$$

$$\frac{P_1 \xrightarrow{a} P_1'}{P_1 \overset{n}{\rhd}^{?}\ P_2 \xrightarrow{a} P_1'} \tag{7.9}$$

$$\frac{}{(a \longrightarrow P) \xrightarrow{tock} (a \longrightarrow P)} \tag{7.10}$$

$$\frac{}{(a \longrightarrow P) \xrightarrow{a} P} \tag{7.11}$$

$$\frac{}{(a : A \longrightarrow P(a)) \xrightarrow{tock} (a : A \longrightarrow P(a))} \tag{7.12}$$

$$\frac{}{(a : A \longrightarrow P(a)) \xrightarrow{b} P(b)}\ [\,b \in A\,] \tag{7.13}$$

$$\frac{P_1 \xrightarrow{tock} P_1' \quad P_2 \xrightarrow{tock} P_2'}{P_1 \Box P_2 \xrightarrow{tock} P_1' \Box P_2'} \tag{7.14}$$

$$\frac{P_1 \xrightarrow{\tau?} P_1'}{P_1 \mathbin{\Box} P_2 \xrightarrow{\tau?} P_1' \mathbin{\Box} P_2} \qquad \frac{P_2 \xrightarrow{\tau?} P_2'}{P_1 \mathbin{\Box} P_2 \xrightarrow{\tau?} P_1 \mathbin{\Box} P_2'} \tag{7.15}$$

$$\frac{P_1 \xrightarrow{a} P_1'}{P_1 \mathbin{\Box} P_2 \xrightarrow{a} P_1'} \qquad \frac{P_2 \xrightarrow{a} P_2'}{P_1 \mathbin{\Box} P_2 \xrightarrow{a} P_2'} \tag{7.16}$$

$$\frac{}{P_1 \sqcap P_2 \xrightarrow{\tau} P_1} \qquad \frac{}{P_1 \sqcap P_2 \xrightarrow{\tau} P_2} \tag{7.17}$$

$$\frac{P_1 \xrightarrow{tock} P_1' \quad P_2 \xrightarrow{tock} P_2'}{P_1 \underset{B}{\parallel} P_2 \xrightarrow{tock} P_1' \underset{B}{\parallel} P_2'} \tag{7.18}$$

$$\frac{P_1 \xrightarrow{\mu} P_1'}{P_1 \underset{B}{\parallel} P_2 \xrightarrow{\mu} P_1' \underset{B}{\parallel} P_2} \; [\, \mu \notin B, \mu \neq \checkmark \,] \tag{7.19a}$$

$$\frac{P_2 \xrightarrow{\mu} P_2'}{P_1 \underset{B}{\parallel} P_2 \xrightarrow{\mu} P_1 \underset{B}{\parallel} P_2'} \; [\, \mu \notin B, \mu \neq \checkmark \,] \tag{7.19b}$$

$$\frac{P_1 \xrightarrow{a} P_1' \quad P_2 \xrightarrow{a} P_2'}{P_1 \underset{B}{\parallel} P_2 \xrightarrow{a} P_1' \underset{B}{\parallel} P_2'} \; [\, a \in B \,] \tag{7.20}$$

$$\frac{P_1 \xrightarrow{\checkmark} P_1'}{P_1 \underset{B}{\parallel} P_2 \xrightarrow{\checkmark} P_1'} \; [\, \checkmark \notin B \,] \qquad \frac{P_2 \xrightarrow{\checkmark} P_2'}{P_1 \underset{B}{\parallel} P_2 \xrightarrow{\checkmark} P_2'} \; [\, \checkmark \notin B \,] \tag{7.21}$$

$$\frac{P_1 \xrightarrow{tock} P_1' \quad P_2 \xrightarrow{tock} P_2'}{P_1 \mathbin{|||} P_2 \xrightarrow{tock} P_1' \mathbin{|||} P_2'} \tag{7.22}$$

$$\frac{P_1 \xrightarrow{\mu} P_1'}{P_1 \mathbin{|||} P_2 \xrightarrow{\mu} P_1' \mathbin{|||} P_2} \; [\, \mu \neq \checkmark \,] \tag{7.23a}$$

$$\frac{P_2 \xrightarrow{\mu} P_2'}{P_1 \mathbin{|||} P_2 \xrightarrow{\mu} P_1 \mathbin{|||} P_2'}\;[\,\mu \neq \checkmark\,] \tag{7.23b}$$

$$\frac{P_1 \xrightarrow{\checkmark} P_1'}{P_1 \mathbin{|||} P_2 \xrightarrow{\checkmark} P_1'} \qquad \frac{P_2 \xrightarrow{\checkmark} P_2'}{P_1 \mathbin{|||} P_2 \xrightarrow{\checkmark} P_2'} \tag{7.24}$$

$$\frac{P_1 \xrightarrow{tock} P_1' \quad P_1 \not\xrightarrow{\checkmark}}{P_1 \,;\, P_2 \xrightarrow{tock} P_1' \,;\, P_2} \tag{7.25}$$

$$\frac{P_1 \xrightarrow{\checkmark} P_1'}{P_1 \,;\, P_2 \xrightarrow{\tau} P_2} \tag{7.26}$$

$$\frac{P_1 \xrightarrow{\mu} P_1'}{P_1 \,;\, P_2 \xrightarrow{\mu} P_1' \,;\, P_2}\;[\,\mu \neq \checkmark\,] \tag{7.27}$$

$$\frac{P \xrightarrow{tock} P' \quad \forall\, a \in A \bullet P \not\xrightarrow{a}}{P \setminus A \xrightarrow{tock} P' \setminus A} \tag{7.28}$$

$$\frac{P \xrightarrow{a} P'}{P \setminus A \xrightarrow{\tau} P' \setminus A}\;[\,a \in A\,] \tag{7.29}$$

$$\frac{P \xrightarrow{\mu} P'}{P \setminus A \xrightarrow{\mu} P' \setminus A}\;[\,\mu \notin A\,] \tag{7.30}$$

$$\frac{P \xrightarrow{tock} P'}{f^{-1}(P) \xrightarrow{tock} f^{-1}(P')} \tag{7.31}$$

$$\frac{P \xrightarrow{\mu} P'}{f^{-1}(P) \xrightarrow{\mu} f^{-1}(P')}\;[\,\mu \in \{\tau, \tau^*, \checkmark\}\,] \tag{7.32}$$

$$\frac{P \xrightarrow{f(a)} P'}{f^{-1}(P) \xrightarrow{a} f^{-1}(P')} \tag{7.33}$$

$$\frac{P \xrightarrow{tock} P'}{f(P) \xrightarrow{tock} f(P')} \tag{7.34}$$

$$\frac{P \xrightarrow{\mu} P'}{f(P) \xrightarrow{\mu} f(P')}\ [\,\mu \in \{\tau, \tau^*, \checkmark\}\,] \tag{7.35}$$

$$\frac{P \xrightarrow{a} P'}{f(P) \xrightarrow{f(a)} f(P')} \tag{7.36}$$

$$\frac{}{\mu\, X \mathbin{\bullet} P \xrightarrow{\tau} P[(\mu\, X \mathbin{\bullet} P)/X].} \tag{7.37}$$

The remark made in Section 3.2 concerning negative premisses (appearing in Rules 7.25 and 7.28) applies here as well.

Note once again the one-to-one correspondence between these rules and those associated with the operational semantics for $\mathcal{M}_R$ and $\mathcal{M}_{TF}$.

The operational semantics enjoys a number of properties, which we list after the following definitions.

If P is a closed node, we define $\mathsf{init}^{\tau}_{UR}(P)$ ($\mathsf{init}^{\tau}(P)$ for short) to be the set of visible non-*tock* and silent events that P can immediately perform: $\mathsf{init}^{\tau}_{UR}(P) \mathrel{\widehat=} \{\mu \in \Sigma^{\checkmark} \cup \{\tau, \tau^*\} \mid P \xrightarrow{\mu}\}$. We also write $\mathsf{init}_{UR}(P)$ to represent the set of non-*tock* visible events that P can immediately perform: $\mathsf{init}_{UR}(P) \mathrel{\widehat=} \mathsf{init}^{\tau}_{UR}(P) \cap \Sigma^{\checkmark}$.

For P a closed node, we define an *execution* of P to be a sequence $e = P_0 \xrightarrow{x_1} P_1 \xrightarrow{x_2} \ldots \xrightarrow{x_n} P_n$ (with $n \geqslant 0$), where $P_0 \equiv P$, the P_i's are nodes, and each transition $P_i \xrightarrow{x_{i+1}} P_{i+1}$ in e is validly allowed by the operational inference Rules 7.1–7.37. Here we have each $x_i \in \Sigma^{\checkmark}_{tock} \cup \{\tau, \tau^*\}$. The set of executions of P is written $\mathsf{exec}_{UR}(P)$, or $\mathsf{exec}(P)$ for short when no confusion is likely.

For P a closed node, the P-rooted graph, or *labelled transition system*, incorporating all of P's possible executions is denoted $\mathsf{LTS}_{\mathit{UR}}(P)$, or $\mathsf{LTS}(P)$ if no confusion is likely.

If $tr = \langle x_1, x_2, \ldots, x_n \rangle$ is a $\tau^?$*-trace* (i.e., $tr \in (\Sigma^{\checkmark}_{tock} \cup \{\tau, \tau^*\})^{\star}$), we write $P \overset{tr}{\Longrightarrow} P'$ to mean that there is some execution $P_0 \xrightarrow{x_1} P_1 \xrightarrow{x_2} \ldots \xrightarrow{x_n} P_n$ of P, with $P \equiv P_0$ and $P' \equiv P_n$, which communicates the $\tau^?$-trace tr. $P \overset{tr}{\Longrightarrow}$ means that there is some node P' such that $P \overset{tr}{\Longrightarrow} P'$, and $P \overset{\langle\rangle}{\Longrightarrow} P$ simply represents the trivial execution $P \in \mathsf{exec}(P)$.

Structural induction proofs proceed here exactly as they did in Section 5.2. In particular, the bisimulation techniques which we have developed carry over seamlessly. We therefore now list a number of results, the proofs of which are left to the conscientious reader.

The following propositions are universally quantified over P, P', P'', which represent closed nodes; we are also using n, m, k, k' to represent natural numbers, etc.

Proposition 7.6 *Time determinacy:*

$$(P \xrightarrow{tock} P' \wedge P \xrightarrow{tock} P'') \Rightarrow P' \equiv P''.$$

Of course, this result extends to an arbitrary number of *tock*s.

Proposition 7.7 *Persistency—the set of possible initial visible events remains constant under the occurrence of tocks:*

$$P \xrightarrow{tock} P' \Rightarrow \mathsf{init}(P) = \mathsf{init}(P').$$

Proposition 7.8 *Maximal progress, or τ-urgency:*

$$P \xrightarrow{\tau^?} \Rightarrow P \overset{tock}{\not\longrightarrow}.$$

Corollary 7.9

$$(P \overset{\langle tock\rangle^n}{\Longrightarrow} P' \xrightarrow{\tau^?} \wedge P \overset{\langle tock\rangle^m}{\Longrightarrow} P'' \xrightarrow{\tau^?}) \Rightarrow n = m.$$

Proposition 7.10 *A node P can always perform a sequence of tocks up to the point of the next $\tau^?$ action, or up to any point if no $\tau^?$ action lies ahead:*

$$\forall\, k \geqslant 1 \,\centerdot\, (\nexists\, P' \,\centerdot\, P \overset{\langle tock\rangle^k}{\Longrightarrow} P') \Rightarrow \exists\, k' < k, P'' \,\centerdot\, P \overset{\langle tock\rangle^{k'}}{\Longrightarrow} P'' \xrightarrow{\tau^?}).$$

In fact, $\tau^? = \tau^*$ in the above unless $k' = 0$, as the next proposition shows.

Proposition 7.11 *A state immediately reachable via a tock-transition cannot be* fleetingly *unstable:*

$$P \xrightarrow{tock} P' \Rightarrow P' \stackrel{\tau}{\not\longrightarrow}.$$

Proposition 7.12 *Finite variability—a* program $P \in \overline{\textbf{TCSP}}$ *cannot perform unboundedly many actions within a finite number of tocks:*

$$\forall\, k \geqslant 0 \,\bullet\, \exists\, n = n(P,k) \,\bullet\, \forall\, tr \in (\Sigma^{\checkmark}_{tock} \cup \{\tau, \tau^*\})^{\star} \,\bullet$$
$$(P \overset{tr}{\Longrightarrow} \wedge\, \sharp(tr \upharpoonright tock) \leqslant k) \Rightarrow \sharp tr \leqslant n.$$

Note also that this formulation of finite variability implies Axiom *UR7*, via Theorem 7.13 below.

For $A \in REF$, we write P **ref** A if $A = \bullet \vee (P \stackrel{\tau^?}{\not\longrightarrow} \wedge\, |A| \cap \mathsf{init}(P) = \emptyset)$. We overload this notation by writing, for $\alpha \in URS$, P **ref** α if $\bigcup\{A \mid [A] \text{ in } \alpha\} = \emptyset \vee (P \stackrel{\tau}{\not\longrightarrow} \wedge \bigcup\{A \mid [A] \text{ in } \alpha\} \cap \mathsf{init}(P) = \emptyset)$. Finally, for $(\alpha, A) \in RPAIR$, we write P **ref** (α, A) if P **ref** $\alpha \wedge P$ **ref** A.

Any execution of a program P gives rise to a set of tests in a manner consistent with our understanding of instability. In detail, letting e be an execution of P and u a test, we define inductively the relation $e\ \mathbf{test}_U\ u$ by induction on e:

$$\begin{array}{rcl}
P\ \mathbf{test}_U\ \langle(\alpha, A)\rangle & \Leftrightarrow & P\ \mathbf{ref}\ (\alpha, A) \\
(P \xrightarrow{\tau})^\frown e'\ \mathbf{test}_U\ u & \Leftrightarrow & e'\ \mathbf{test}_U\ u \\
(P \xrightarrow{\tau^*})^\frown e'\ \mathbf{test}_U\ \alpha\ \mathsf{glue}\ u & \Leftrightarrow & P\ \mathbf{ref}\ \alpha \wedge e'\ \mathbf{test}_U\ u \\
(P \xrightarrow{x})^\frown e'\ \mathbf{test}_U\ \langle(\alpha, A), x\rangle^\frown u' & \Leftrightarrow & x \neq \tau^? \wedge P\ \mathbf{ref}\ (\alpha, A) \wedge e'\ \mathbf{test}_U\ u'.
\end{array}$$

We can now define the function Φ_{UR}, which extracts the denotational representation of a program from its set of executions.

Definition 7.2 *For* $P \in \overline{NODE}_R$, *we set*

$$\Phi_{UR}(P) \quad \widehat{=} \quad \{u \mid \exists\, e \in \mathsf{exec}_{UR}(P) \,\bullet\, e\ \mathbf{test}_U\ u\}.$$

The chief congruence result now reads:

Theorem 7.13 *For any* $\overline{\textbf{TCSP}}$ *program* P, *we have*

$$\Phi_{UR}(P) = \mathcal{R}_U[\![P]\!].$$

The proof, while requiring a little more bookkeeping, follows the same pattern as that of Theorem 5.14, and is therefore omitted.

7.3 Refinement, Specification, Verification

Let us conclude this chapter by saying a few words about specification satisfiability and verification. Much of our discussion on this topic in the case of $\mathcal{M}_R$ could be lifted directly to $\mathcal{M}_{UR}$, and it therefore seems unnecessary to reproduce it here in great detail.

The nondeterministic refinement order can be defined on $\mathcal{P}(TEST_U)$ in the expected manner:

Definition 7.3 *For $P, Q \subseteq TEST_U$ (and in particular for $P, Q \in \mathcal{M}_{UR}$), we let $P \sqsubseteq_{UR} Q$ if $P \supseteq Q$. For $P, Q \in \overline{\mathbf{TCSP}}$, we write $P \sqsubseteq_{UR} Q$ to mean $\mathcal{R}_U[\![P]\!] \sqsubseteq_{UR} \mathcal{R}_U[\![Q]\!]$, and $P =_{UR} Q$ to mean $\mathcal{R}_U[\![P]\!] = \mathcal{R}_U[\![Q]\!]$.*

We may drop the subscripts and write simply $P \sqsubseteq Q$, $P = Q$ whenever the context is clear.

This partial order naturally satisfies

$$P \sqsubseteq Q \Leftrightarrow P \sqcap Q = P.$$

Specifications, of course, are assertions about processes, and *behavioural* specifications, such as process refinement, are specifications that are universally quantified in the behaviours of the process. Behavioural specifications can be identified with sets of tests, with a process meeting the specification if it is a subset of it. Note that if a behavioural specification does not concern unstable refusals, Proposition 7.4 implies that it can simply be verified in $\mathcal{M}_R$.

Although we have not fully verified this, we strongly suspect, and will presume, that $\mathcal{M}_{UR}$ can be embedded into a domain in the same manner as $\mathcal{M}_R$, by dropping Axiom *UR7*. Consequently, a large class of behavioural specifications should be expressible as refinements; moreover, these should be model checkable via the operational semantics using the algorithmic techniques which we describe in Appendix B.

Chapter 8

Timed Analysis (II)

We revisit some of the analysis carried out earlier in Chapter 6, and show that $\mathcal{M}_{UR}$ does circumvent, as claimed, certain of the problems we encountered earlier. We concentrate exclusively on timed failure soundness, since completeness, as well as the verification of specifications closed under inverse digitisation, were shown to be adequately handled within $\mathcal{M}_R$.

The main result of this chapter, Theorem 8.1, is interesting and valuable in that it confirms that refining timing granularity does lead to increasingly and arbitrarily precise approximations of true continuous-time process behaviour. This result does however come at a significant expense in modelling and algorithmic complexity and is likely to be less useful than the other verification techniques described in Chapter 6.

8.1 Timed Denotational Expansion

We first define a timed denotational expansion function $\mathsf{Exp}_k : \mathcal{M}_{UR} \longrightarrow \mathcal{P}(TF)$ which extends Chapter 6's version of Exp_k in a manner consistent with unstable refusal information.

The construction goes as follows. Let $P \in \mathcal{M}_{UR}$ and let the test $u = \langle(\alpha_0, A_0), a_1, (\alpha_1, A_1), a_2, \ldots, a_n, (\alpha_n, A_n)\rangle$ belong to $\mathcal{R}_U[\![P]\!]$. Let us write each α_i as $\alpha_i = [A_i^1, A_i^2, \ldots, A_i^{k_i}]$. We first define $\mathsf{preExp}_k(u)$ to be the set of timed failures $(s, \aleph) \in TF$ satisfying the following:

$$\begin{aligned}
&\exists(t_i^j \in \mathbb{R}^+ \mid 0 \leqslant i \leqslant n \wedge 0 \leqslant j \leqslant k_i), t_{n+1}^0 \in \mathbb{R}^+ \bullet \\
&\quad t_0^0 = 0 \wedge t_{n+1}^0 = (1 + \sharp(\langle a_1, a_2, \ldots, a_i \rangle \upharpoonright \mathit{tock}))/k \wedge \\
&\quad (\forall(i,j,l,m) \bullet t_i^j \leqslant t_{i+l}^m \Leftrightarrow l > 0 \vee (l = 0 \wedge m \geqslant j)) \wedge t_n^{k_n} < t_{n+1}^0 \wedge \\
&\quad s = \langle (t_1, a_1), (t_2, a_2), \ldots, (t_n, a_n) \rangle \setminus \mathit{tock} \wedge \\
&\quad (\forall(1 \leqslant i \leqslant n) \bullet \\
&\quad\quad a_i = \mathit{tock} \Rightarrow (t_i^0 = \sharp(\langle a_1, a_2, \ldots, a_i \rangle \upharpoonright \mathit{tock})/k \wedge t_{i-1}^{k_{i-1}} \neq t_i^0)) \wedge \\
&\quad \aleph \subseteq \bigcup_{i=0}^{n} \{ \bigcup_{j=0}^{k_i - 1} \{[t_i^j, t_i^{j+1}) \times A_i^{j+1}\} \cup [t_i^{k_i}, t_{i+1}^0) \times |A_i| \}.
\end{aligned}$$

We can now give

Definition 8.1 *For any $P \in \mathcal{M}_{UR}$ and $k \geqslant 1$, we define the* timed denotational expansion $\mathsf{Exp}_k(P)$ *of P to be*

$$\mathsf{Exp}_k(P) \quad \widehat{=} \quad \bigcup \{\mathsf{preExp}_k(u) \mid u \in P\}.$$

(We will usually omit the subscript k when it is equal to 1.)

Note that the null refusal $\bullet$ is still treated as the empty refusal set.

8.2 Timed Failure Soundness

The timing inaccuracies pertaining to *traces* which we observed in Section 6.2 naturally persist in the current setting. However, the problems having to do with under-refusing events—which had ruled out any timed failure soundness result—have now vanished, thanks to unstable refusal information. As an illustration, consider once more the program

$$P = a \xrightarrow{1} \bar{b}.$$

We had observed that on the timed trace $(0.7, a)$, the timed denotational expansion of $\mathcal{R}[\![P]\!]$ was incapable of refusing b past time 1. However, the situation is different with the timed denotational expansion of $\mathcal{R}_U[\![P]\!]$: after the first *tock*, a τ^*-transition is on offer, while b is robustly refused. $\mathsf{Exp}(\mathcal{R}_U[\![P]\!])$ will therefore have the behaviours $(\langle (0.7, a) \rangle, [0, t))$, for all $t < 2$. (Note, however, that t is not allowed to equal 2, since the τ^*-transition must occur strictly before the second *tock*.) These behaviours certainly cover the timed failure $(\langle(0.7,a)\rangle,[0,1.7)) \in \mathcal{F}_T[\![P]\!]$. And the corresponding test in $\mathcal{R}_U[\![P]\!]$ is simply $\langle([],\{b\}),a,([],\{b\}),tock,([\{b\}],\bullet)\rangle$.

This situation is captured in general in Theorem 8.1, which we present after the following preliminaries.

Recall how the operational semantics for $\mathcal{M}_{UR}$ is derived from the operational semantics for $\mathcal{M}_R$ in such a way as to allow us to 'keep track' of whether nodes are in fleeting or robust states of instability (or, of course, in neither). This information is then encapsulated, respectively, in whether a node can perform τ-transitions or τ^*-transitions (or neither). Naturally, the operational semantics for $\mathcal{M}_{TF}$ can be altered in exactly the same way, by adopting precisely the same conventions as to the communications of $\tau^?$ events as the operational semantics for $\mathcal{M}_{UR}$. Let us consider such an altered operational semantics for the remainder of this section.

One can then establish an analogue of Proposition 7.11, to the effect that any state immediately reachable via an evolution cannot be fleetingly unstable: for $P,P' \in \overline{NODE}_{TF}$ and $t \geqslant 0$,

$$P \overset{t}{\leadsto} P' \Rightarrow P' \overset{\tau}{\not\longrightarrow}.$$

The proof is a simple structural induction on P.

We have commented earlier how operational inference Rules 7.1–7.37 are in one-to-one correspondence with Rules 3.1–3.37. This naturally leads to a one-to-one correspondence between $\mathcal{M}_{UR}$-executions and normal $\mathcal{M}_{TF}$-executions. More precisely, this bijection takes a normal execution $e = P_0 \xmapsto{z_1} P_1 \xmapsto{z_2} \ldots \xmapsto{z_n} P_n \in \mathsf{exec}_{TF}(P)$ to the execution $e' = P_0 \xrightarrow{x_1} P_1 \xrightarrow{x_2} \ldots \xrightarrow{x_n} P_n \in \mathsf{exec}_{UR}(P)$, such that, for all $1 \leqslant i \leqslant n$, $x_i = tock$ when $\xmapsto{z_i} = \overset{1}{\leadsto}$, and $x_i = \mu$ when $\xmapsto{z_i} = \xrightarrow{\mu}$ (where $\mu \in \Sigma^{\checkmark} \cup \{\tau,\tau^*\}$).

Of course, we have, for any $P \in \mathcal{M}_{UR}$ and $k \geqslant 1$, $\mathsf{Exp}(P) = k\mathsf{Exp}_k(P)$.

We can now state:

Theorem 8.1 *For any $P \in \overline{\mathbf{TCSP}}$ and $k \geqslant 1$,*

$$\mathsf{Exp}_k(\mathcal{R}_U[\![kP]\!]) \sqsubseteq_{TF} \mathcal{F}_T[\![P]\!].$$

Proof (Sketch.) As usual, it suffices to prove the result for $k = 1$.

Let P be fixed, and pick $(s,\aleph) \in \mathcal{F}_T[\![P]\!]$. There is some execution $e = P_0 \xmapsto{z_1} P_1 \xmapsto{z_2} \ldots \xmapsto{z_n} P_n \in \mathsf{exec}_{TF}(P)$ such that e **fail** $(s,\aleph)$. Let $e' = [e]_1$

be the lower digitisation of e, let e'' be the normalisation of e', and let $o \in \mathsf{exec}_{UR}(P)$ be the canonical execution corresponding to e''. Let $u \in \mathcal{R}_U[\![P]\!]$ be a $\prec$-maximal test such that $o\ \mathbf{test}_U\ u$.

We have already seen, in Theorem 6.5, that $s \in \mathsf{TTraces}(\mathsf{preExp}(u))$. We would like to show that in fact $(s, \aleph) \in \mathsf{preExp}(u)$. We proceed as in the timed trace case, and consider an arbitrary element $(t, a) \in \aleph$. We reason that there must be $P_k \overset{t'}{\rightsquigarrow} P_{k+1}$ in e such that $\mathsf{dur}(e_k) \leqslant t$, $\mathsf{dur}(e_k) + t' > t$, and $P_k \overset{a}{\not\longrightarrow}$. (Here $e_k = P_0 \overset{z_1}{\longmapsto} P_1 \overset{z_2}{\longmapsto} \ldots \overset{z_k}{\longmapsto} P_k \leq e$ represents the prefix of e up to node P_k.) Under digitisation, the evolution $P_k \overset{t'}{\rightsquigarrow} P_{k+1}$ becomes $P'_k \overset{l}{\rightsquigarrow} P'_{k+1}$ in e', with $l \in \mathbb{N}$. Without loss of generality, let us assume that $l = 0$ or $l = 1$, the second of which could be achieved via normalisation in case $l > 1$.

If $l = 0$, then $\mathsf{dur}(e'_k) = [t]_1 = \mathsf{dur}(e'_{k+1})$. Since $P_k \overset{a}{\not\longrightarrow}$ by assumption, $P'_k \overset{a}{\not\longrightarrow}$, by the properties of digitisation. But $\mathsf{init}(P'_k) = \mathsf{init}(P'_{k+1})$ by persistency, and thus $P'_{k+1} \overset{a}{\not\longrightarrow}$. On the other hand, our earlier discussion lets us conclude that P'_{k+1} cannot be fleetingly unstable, i.e., $P'_{k+1} \overset{\tau}{\not\longrightarrow}$. It follows that a must figure in the unstable refusal immediately following P'_{k+1}, and therefore that a is refusable from $\mathsf{dur}(e'_{k+1}) = [t]_1$ to any time strictly less than $\mathsf{dur}(e'_{k+1}) + 1 = [t]_1 + 1$. Naturally, this interval comprises t, which entails that (t, a) is able to appear in the refusals of $\mathsf{preExp}(u)$.

If $l = 1$, then either $[t]_1 = \mathsf{dur}(P'_{k+1})$, in which case we repeat the above argument, or $[t]_1 = \mathsf{dur}(P'_k)$, in which case a is stably refused after P'_k for up to one time unit. The result follows.

It is clear that this procedure can be carried out uniformly and concurrently for all $(t, a) \in \aleph$, as well as for the trace s. We conclude that $(s, \aleph) \in \mathsf{preExp}(u) \subseteq \mathsf{Exp}(P)$, as required. ■

Theorem 8.1 is the timed failure soundness result which we had sought. It informs us that if some undesirable timed failure behaviour is ruled out by the denotational expansion of $\mathcal{R}_U[\![P]\!]$, then it is certainly not a possible behaviour of $\mathcal{F}_T[\![P]\!]$. Together with quasi-completeness (Theorem 6.13, which can easily be lifted to $\mathcal{M}_{UR}$), it allows us to 'zero in' on $\mathcal{F}_T[\![P]\!]$ by letting, if need be, k increase. Eventual applications to specification verification would follow the same pattern as in Chapter 6.

Chapter 9

Discussion, Comparisons, Future Work

The object of this thesis was to study the relationship between continuous-time and discrete-time models for timed process algebras, and to exploit this relationship towards applications, in particular model checking. Our main accomplishments are twofold: firstly, we have produced a stand-alone CSP-based discrete-time denotational model, equipped with a congruent operational semantics, and compatible with the model checker FDR. Secondly, we have linked this model to the standard continuous-time denotational and operational models for Timed CSP, and have shown how this connection could be exploited to reduce continuous-time verification problems to discrete-time, model checkable verification problems.

In greater detail, our work can be summarised as follows.

We introduced Timed CSP as our prototypical process algebra, together with congruent continuous-time denotational and operational semantics, and established an important result, the digitisation lemma, which later allowed us to develop a powerful model checking technique with wide-ranging applicability.

We then discussed the principal issues pertaining to discrete-time modelling, and explained how the main difficulties could be overcome in certain models for untimed CSP.

We proceeded to carefully construct such a model, the discrete-time refusal testing model $\mathcal{M}_R$, offering congruent denotational and operational semantics which incorporated the properties previously identified. Refinement, specifications and verification were discussed, and a model checking algorithm—presented in Appendix B—was described. We also remarked that in the majority of the verification cases one was likely to meet in practice, the model checker FDR could be used directly.

We followed up with a chapter in which the exact links between this model and the continuous-time model $\mathcal{M}_{TF}$ were studied, with once again an underlying emphasis towards verification. We discovered that the relationship between these continuous-time and discrete-time models for Timed CSP was a very tight one, enabling us in a large number of instances to reduce verification problems in $\mathcal{M}_{TF}$ to corresponding model checkable problems in $\mathcal{M}_R$. More precisely, we built upon and later extended a result of Henzinger, Manna, and Pnueli's to reduce specification satisfiability problems from continuous-time to discrete-time, provided the specifications in question are closed under inverse digitisation. Since many specifications have that property, this provided us with an efficient model checking algorithm, mentioned above, implementable on FDR in most cases. These results, which constitute the most important contribution of our work, were then illustrated in a simple railway level crossing case study. We also identified and discussed the limitations of our discretisation approach.

A further chapter was devoted to constructing an extension of $\mathcal{M}_R$, the discrete-time unstable refusal testing model $\mathcal{M}_{UR}$, to overcome certain shortcomings of $\mathcal{M}_R$ relating to unstable refusal information. Our treatment paralleled that of $\mathcal{M}_R$, and we were able to show that $\mathcal{M}_{UR}$ captured continuous-time behaviour more accurately and smoothly than $\mathcal{M}_R$. However it remains to be shown how substantial the eventual practical benefits of this observation will prove to be.

We elected to focus on Timed CSP for a number of reasons. Firstly, Timed CSP comes endowed with continuous-time denotational and operational semantics, has been the focus of a considerable body of previous work, and is a natural extension of (untimed) CSP. The latter, of course, has been even more widely studied and is equipped with denotational, operational, and algebraic semantics, as well automated model checking tools. The combination of Timed and untimed CSP therefore seemed likely to be a fertile terrain in which to conduct our investigations.

Secondly, Timed CSP possesses a large number of the properties and characteristics usually found in other timed process algebras. The robustness of the methods used in our analysis makes it very plausible that our results should apply to a wide range of similar process algebras, as well as to a number of proposed extensions and variants of Timed CSP, discussed below.

Lastly, denotational CSP-based models typically have an internal specification mechanism rooted in *refinement*. This allowed us to avoid having to introduce an independent formal specification formalism, such as temporal logic, and concentrate instead on the task at hand. Handling specifications via refinement has the additional appeal of being much closer to the top-down approach usually adopted in system design and implementation; refinement-based formal methods are consequently more likely to prove useful throughout a project's production cycle. Temporal-logic-based formalisms, on the other hand, can be more flexible (enabling one, for example, to express assertions about internal state variables), and can be studied independently of any model, making them more portable. For an in-depth account of the use of temporal logics in Timed and untimed CSP, we refer the reader to [Jac92].

A number of continuous-time process algebras other than Timed CSP appear in the literature. These include the algebras of [Wan91, Che92], which are timed extensions of CCS [Mil89], and predicated upon an operational semantics which uses strong bisimulation for process equivalence. In [BB91], Baeten and Bergstra combine a similar approach together with a congruent complete algebraic semantics, while Gerth and Boucher [GB87] complement their operational semantics with a timed-failures-based denotational semantics similar to that of $\mathcal{M}_{TF}$. A more abstract denotational model, subsuming timed failures, is offered in [Jef91a], once again together with a congruent operational semantics.

Discrete-time process algebras have also been widely studied. In [MT90, NS94, CLM97], we find process algebras rooted in operational semantics and strong bisimulation, whereas the process algebra of [HR91] uses instead an operational semantics in which testing equivalence defines the notion of equality between processes. Jeffrey [Jef91b] proposes a process algebra incorporating infinite nondeterminism and Zeno processes, rooted in a failures-based denotational semantics, and equipped with a congruent complete algebraic semantics. Naturally, some of the continuous-time process algebras noted above also have discrete-time interpretations.

Although these lists are far from exhaustive, they give an idea of the range of alternative process algebraic formalisms available in the literature. More thorough overviews can be found in [NS91, Sch95, Ver97].

The discrete-time models which we have constructed in this thesis build upon ideas appearing in [RR86, RR87, Ree88, RR99, Phi87, Muk93, Ros97] (on the denotational side), and [Sch95, NS91, NS94] (on the operational side). Denotationally, $\mathcal{M}_R$ and $\mathcal{M}_{UR}$ share a number of characteristics with Jeffrey's model for Discrete timed CSP [Jef91b].

Other formalisms for modelling timed systems include *timed transition systems* [HMP90], *timed graphs* [ACD90, ACD93], *timed automata* [AD94], *hybrid systems* [NOSY93, Hen96, HHWT97], etc. Unlike process algebras, these tend to be intrinsically rooted in state-based models, and are usually closer to the implementation level. One could argue that process algebras, being more abstract, are more versatile and better suited to algebraic and modular (i.e., compositional) reasoning, while retaining state-based interpretations via their operational semantics (together with, if need be, congruence theorems). On the other hand, the price paid for this higher level of abstraction is often a significant overhead in rigorously establishing certain operational properties of processes—cf., for instance, the proofs of the digitisation lemma or finite variability. One is also sometimes faced with unwanted phenomena which are not always easy—if at all possible—to get rid of, such as Zeno behaviours, divergence, point nondeterminism, unbounded nondeterminism, and so on.

One of the earliest attempts to discretise continuous-time models with a view towards verification was the technique of *timewise refinement*, introduced in [Ree88, Ree89] and developed in [Sch89, RRS91, Sch97]. This method is essentially a translation from Timed to untimed CSP which drops all timing information, retaining only (and loosely) the relative order in which events occur. Although of course not suitable for every verification purpose, timewise refinement, when applicable, is a spectacularly economical proof technique, often able to handle problems for which it is not computationally feasible to apply straightforward model checking algorithms.

Alur, Courcoubetis, and Dill's seminal work on *region graphs* [ACD90, ACD93] provided the first algorithm to model check (continuous-time) timed graphs with respect to satisfiability of formulas expressed in the temporal logic TCTL. Timed graphs consist of a finite set of *nodes* together with a finite set of real-valued *clocks*, giving rise to an uncountable number of *states*. However, Alur *et al.* showed that it was possible to construct an equivalence relation on the state space, partitioning it into a *finite* number of *regions*, in such a way that no TCTL formula can distinguish states belonging to the same region. This led naturally to a model checking algorithm. The technique is however computationally expensive—deciding satisfiability of TCTL formulas by timed graphs can be shown to be PSPACE-complete—and much subsequent research has concentrated on finding ways to curb the resulting complexity [HNSY94, LL95, SS95, HKV96, HK97].

Jackson [Jac92] showed how to apply the region graphs algorithm in the timed failures model for a restricted subset of Timed CSP. The idea essentially consists in curtailing Timed CSP syntax, yielding the sub-process-algebra FTCSP, all of whose programs can then be translated into equivalent timed graphs. (Going in the other direction, the constructions of [CG95] indicate how to translate timed automata into Timed CSP—the latter is thus at least as expressive as the former.)

According to Cook's thesis, the problem of deciding satisfiability of TCTL formulas by timed graphs, being PSPACE-complete, is 'intractable'. However, in many practical cases the actual complexity turns out to be manageable, for two reasons. The first is that the systems in which we tend to be interested often exhibit symmetry and regularity, which significantly reduces the complexity. The second reason is that, even though the equivalence relation associated with region graphs cannot be made any coarser if arbitrary TCTL formulas are considered, in many cases (such as reachability analysis, or specifications that are closed under inverse digitisation), verifying the specification in question does not necessarily require the full discriminative power of regions, consequently reducing the complexity. This has been exploited, among others, in [BSV95], leading to a general technique of iterative refinement which could easily also be employed in our setting.

We will not give a precise and comprehensive account of how the work carried out in this thesis compares with region graphs, as such an account would have to be quite lengthy. However, a rough overview can be offered as follows. Each parallel component of a compound system has a clock associated to it. Regions are then basically constructed by comparing respectively the integral and fractional parts of these clocks, where for fractional parts the allowable comparison predicates are *equality with zero*, *equality*, and *less than*. In our setup, on the other hand, our use of the *tock* event faithfully captures the integral part of a *single* clock, and furthermore any subsequent information concerning the clock's fractional part can only be expressed in terms of a single predicate, *less than or equal to*. It follows that the equivalence relation which this induces on the state space is much coarser than that of region graphs, with the twin corollaries that the analysis offered within our framework should be computationally more efficient than full-fledged region graphs algorithms, but may also occasionally fail to produce an answer in situations where a region graphs investigation would be successful. This being said, we recall that, by refining the time granularity of the discrete-time interpretations of programs, increasing precision can be achieved. We shall return to this question shortly.

In [HMP92], Henzinger *et al.* address the important question of which dense-time properties can be investigated with integral-time methods, and how. They carry out their work in the realm of timed transition systems, together with the temporal logic MTL. We have used one of their main results—Theorem 6.8, dealing with timed traces—which we later extended in Theorem 6.15 to incorporate timed failures. We then built on these results to produce the central Theorems 6.10 and 6.16 and Corollaries 6.11 and 6.17, allowing us to reduce the verification of any specification closed under inverse digitisation to our discrete-time setting.

Much research has gone into temporal logics and their relation to model checking. Excellent and thorough accounts can be found in [HMP90, AH93, AH94, Hen98]. In [HK97], Henzinger and Kupferman show that model checking TCTL formulas can be carried out on any tool which allows CTL model checking. As they point out, since such tools are typically much more sophisticated (not to mention widespread) than TCTL model checkers, this reduction amounts to more than just an interesting theoretical result. Likewise, an advantage of the work which we have carried out is that efficient and competitive model checking tools such as FDR can (in most cases) directly be employed to verify specifications on Timed CSP processes.

An interesting distinctive approach to using temporal logics towards verification has been the development of TLA and TLA$^+$, together with the model checker TLC [YML99]. Rather than expressing system and specification using two different formalisms (such as timed graphs and temporal logic), the idea here is to translate the system itself into the temporal logic TLA$^+$, the very language used to express specifications. The satisfiability check is then carried out entirely within a single temporal logic framework. (This approach is in some sense dual to ours, where system and specification are both expressed as processes, with satisfiability corresponding to refinement.)

Symbolic model checking and related techniques have attracted much attention in recent years. Examples include [BCM$^+$92], using *binary decision diagrams*, [BLP$^+$99], introducing *clock difference diagrams*, a timed analogue of binary decision diagrams, [HNSY94, NOSY93], focussing on hybrid systems, and [HKV96, AHR98], focussing on dense-time timed transition systems. Other formal methods include *proof systems* [Sch89, Dav91, Sch94], *property preserving abstractions* [LGS$^+$95], and applications of *data-independence* [Laz99]. Whether and how such techniques can be adapted to our framework are very interesting and relevant research questions.

Let us now consider in turn a number of avenues along which our framework could be extended, and discuss other opportunities for future research.

An intrinsic limitation of the original formulation of CSP and Timed CSP has to do with the need for the environment to cooperate on all events. Although this difficulty can usually be circumvented in practice, as we argue below, it does lead to unnatural situations. As pointed out by Davies [Dav91], not all events are appropriately modelled as synchronisations: a telephone may ring, for instance, even if no-one is prepared to actively participate in listening to it.[1] However it is impossible in Timed CSP to describe a system in which, for example, the event *SprinklerOn* is guaranteed to occur at most $5s$ after the event *FireAlarm* is observed; in general it is even impossible to guarantee that events occur as soon as they become available.

Current solutions to this problem fall in three categories. The first is to bypass the need for environmental cooperation by hiding the event in question, as we did in our railway level crossing example (Section 6.7) to ensure that cars drove off the dangerous crossing after exactly $10s$. Unfortunately, such events, being hidden, can then by definition no longer be observed. A second, more sophisticated solution is to postulate an *assume-guarantee* framework, which lets us draw appropriate conclusions, or guarantees, from environmental assumptions; this is discussed, among others, in [KR91, Dav91], and applied in [Sch94]. From a model checking perspective, a possible implementation could consist in ascribing a certain priority to such events to ensure that time cannot pass while they are on offer. The third and most radical option is to redesign the semantic models to incorporate *broadcast concurrency*, or *signals*, as is done in [Dav91]. We expect that any of these approaches would successfully mesh with the relatively robust constructions and techniques described in this thesis.

Another point worth addressing is the extent to which we can push the paradigm of specification-as-refinement in Timed CSP, and how it compares with temporal logic, particularly in terms of expressiveness. This difficult issue is obviously very important, given the nature of our denotational models as well as the mode of operation of FDR, if we are serious about applying our efforts to real-world problems. As we have seen, the main difficulties stem from the fact that Zeno behaviours and unrestricted unbounded nondeterminism sit uneasily with our denotational semantics (particularly in continuous time). We have argued that it is not Zeno behaviours *per se* that are problematic, but the fact that they potentially lead to divergence under hiding. Of course, the fact that our models are predicated upon ultrametric spaces and invoke the contraction mapping theorem to compute fixed points

[1]Then again... This echoes the age-old metaphysical query, "if a tree falls in a forest with no-one nearby, does it make a noise?"

means that some definite work will be required to integrate Zeno features into the denotational semantics. A starting point seems to lie in the fact that our discrete-time models have alternative cpo-based constructions.

An interesting question is whether our discrete-time model $\mathcal{M}_R$ is canonical in any way; i.e., whether it is the most abstract model with respect to some outside criterion, such as some testing equivalence, or the verification of specifications that are closed under inverse digitisation. As far as the second criterion is concerned, Proposition 6.18 hints strongly that the answer is negative. Steve Schneider has suggested a discrete-time model similar to $\mathcal{M}_R$ but with refusals recorded only prior to *tock* events. It seems that it should be possible to build such a model and employ it as we have $\mathcal{M}_R$ to verify exactly specifications closed under inverse digitisation. Moreover, we believe that such a model would in fact be fully abstract with respect to this criterion. Since this model would clearly be coarser than $\mathcal{M}_R$, it is natural to believe that its corresponding model checking algorithm should be more efficient than $\mathcal{M}_R$'s as well.

On the other hand, we have seen that $\mathcal{M}_R$ can be used to verify exactly certain specifications that are *not* closed under inverse digitisation. Can we characterise these specifications? Can we characterise specifications, or perhaps processes, that are closed under inverse digitisation?[2]

Another interesting research direction is to look for *less* abstract models than $\mathcal{M}_R$, together with correspondingly larger natural classes of specifications that such models could verify exactly. Bill Roscoe in particular has suggested a model similar ro $\mathcal{M}_R$ but with the *tock* event split into two 'simultaneous' sub-events, representing respectively the beginning and the end of a standard *tock*. In this way, it would be possible to determine whether other events occur strictly before, at the same time as, or strictly after *tock*. It seems clear that such a model would induce a finer equivalence than $\mathcal{M}_R$ on the continuous-time state space of processes, although it would clearly not correspond to the still finer equivalence of region graphs. Whether the latter equivalence can be captured in a discretisation framework is a very interesting question, the answer to which we believe to be negative. (By "discretisation" we essentially mean discrete-time modelling in which time is represented via a *single* fictitious digital clock.) In support of our negative conjecture, note that, according to Proposition 6.22, there are specifications which fail to become closed under inverse digitisation at *any* level of timing

[2]In [HMP92], Henzinger *et al.* offer sufficient criteria for a specification to be closed under inverse digitisation, but there is no suggestion that these criteria are in fact necessary.

granularity. This also suggests that it is impossible to obtain a discrete-time model in which *every* refinement-based Timed CSP specification can be verified exactly. We believe all these questions to be very interesting and well worthy of further research.

A difficult problem seems to be the development of an algebraic semantics for Timed CSP. Propositions 6.19 and 6.20 imply that $\mathcal{M}_{TF}$ and $\mathcal{M}_R$ would actually have to have different such semantics. Unfortunately, defining a suitable notion of normal form seems complicated since any normal form would necessarily have to incorporate parallel operators, in view of Proposition 6.23.[3] An interesting question is whether there is a good inherent notion of minimal degree of parallelism for timed processes (perhaps related to minimal sets of clocks in corresponding timed automata—see [ACH94, KR96, CG95]). As a related note, we point out that all the "laws of Timed CSP" listed in [RR99] can easily be shown to hold in each of $\mathcal{M}_{TF}$, $\mathcal{M}_R$, and $\mathcal{M}_{UR}$ (with $\delta = 0$).

Full abstraction has been discussed earlier with respect to criteria such as the verification of specifications that are closed under inverse digitisation. A dual concern is that of the density of the denotational image of the set $\overline{\mathbf{TCSP}}$ of programs in the denotational model (whether $\mathcal{M}_{TF}$ or $\mathcal{M}_R$). To our knowledge, very little work has been carried out in this direction in the past (in the case of $\mathcal{M}_{TF}$), which suggests that it is not easy to identify an adequately constraining set of axioms. For example, a process such as $P = \mathit{WAIT}\ 0.5$ could not, in our syntax, be reached as the limit of a sequence of $\overline{\mathbf{TCSP}}$ programs (with respect to the standard ultrametric on $\mathcal{M}_{TF}$). Note however that P satisfies Axioms *TF1–TF5*, i.e., P is a *bona fide* element of $\mathcal{M}_{TF}$. A solution could be to ban P by imposing the digitisation lemma as a sixth axiom. Would there then still be $\mathcal{M}_{TF}$ processes which could not be approximated by $\overline{\mathbf{TCSP}}$ programs? And if so, can one find a sufficiently constraining set of axioms that would achieve density of $\overline{\mathbf{TCSP}}$? These questions, of course, can also be asked of $\mathcal{M}_R$ and $\mathcal{M}_{UR}$. We remark, while on the topic, that there is no particular reason to prefer to enunciate properties such as Axioms *R1–R8* as axioms rather than establish them as propositions, other than to derive a density result of the type described above. In fact, as we saw in the proof of the congruence theorem for $\mathcal{M}_R$, there is probably less work to do if one has as few axioms to start with as possible.

A rather deep and difficult problem in Timed CSP concerns the study of nondeterminism, particularly in the continuous-time case (see [RR99]).

[3]Schneider's 'head standard form' [Sch92] incorporates unbounded nondeterministic choice as well as a generalised prefix operator, neither of which are allowed in our syntax.

It is interesting to ask whether these questions could be clarified via the links between the continuous-time and discrete-time models. For instance, is determinism an invariant whether a process is interpreted in $\mathcal{M}_{TF}$ or $\mathcal{M}_R$? The answers to such questions potentially have important applications to the study of non-interference in computer security [Ros95]. This is a research programme which we are already involved in.

Throughout this work, we have exclusively been concerned with the timed failures model $\mathcal{M}_{TF}$. One possible extension would be to move over to the more sophisticated timed failures-stability model $\mathcal{M}_{TFS}$ [RR99], able to handle 'global' stability (invariance with respect to the passage of time). The bedrock of our constructions and analyses has lain in having a congruent operational semantics complementing our denotational continuous-time model, together with an appropriate digitisation lemma. The first of these (in the case of $\mathcal{M}_{TFS}$) is achieved in Appendix C, and we are confident that the second should easily follow.

A further extension could be the inclusion of *fairness*. A traditional solution to this problem is achieved via *infinite traces* models (see, e.g., [Ros97]), which for obvious reasons are not an ideal framework for model checking. A less subtle, but more forcible approach consists in prioritising events, for example by lowering the priority of certain events if they are seen to occur 'a lot'. As we have seen, achieving priority in our models, which allowed us to implement maximal progress, is not difficult. The framework could easily be extended and authoritarian renditions of fairness subsequently enacted.

Another possibility could be the addition of *functional programming* capabilities to the syntax, as is done for instance in [Laz99] in untimed CSP.

A much more ambitious, if speculative, extension would see the incorporation of *probability* to the framework. A good starting point might be [Low95].

We conclude by pointing out that, however interesting the theoretical results and speculations we have presented here are, it would also certainly be desirable to apply them to a number of case studies and find out exactly how closely theory matches practice.

Appendix A

Mathematical Proofs

A.1 Proofs for Chapter 3

Lemma A.1 (3.11) *Let $P \in \overline{\mathbf{TCSP}}$, and let $e = P_0 \xmapsto{z_1} P_1 \xmapsto{z_2} \ldots \xmapsto{z_n} P_n \in \mathsf{exec}_{TF}(P)$. For any $0 \leqslant \varepsilon \leqslant 1$, there exists an execution $[e]_\varepsilon = P_0' \xmapsto{z_1'} P_1' \xmapsto{z_2'} \ldots \xmapsto{z_n'} P_n' \in \mathsf{exec}_{TF}(P)$ with the following properties:*

1. *The transitions and evolutions of e and $[e]_\varepsilon$ are in natural one-to-one correspondence. More precisely, whenever $P_i \xmapsto{z_{i+1}} P_{i+1}$ in e is a transition, then so is $P_i' \xmapsto{z_{i+1}'} P_{i+1}'$ in $[e]_\varepsilon$, and moreover $z_{i+1}' = z_{i+1}$. On the other hand, whenever $P_i \xmapsto{z_{i+1}} P_{i+1}$ in e is an evolution, then so is $P_i' \xmapsto{z_{i+1}'} P_{i+1}'$ in $[e]_\varepsilon$, with $|z_{i+1} - z_{i+1}'| < 1$.*

2. *All evolutions in $[e]_\varepsilon$ have integral duration.*

3. *$P_0' \equiv P_0 \equiv P$; in addition, $P_i' \in \overline{\mathbf{TCSP}}$ and $\mathsf{init}_{TF}(P_i') = \mathsf{init}_{TF}(P_i)$ for all $0 \leqslant i \leqslant n$.*

4. *For any prefix $e(k) = P_0 \xmapsto{z_1} P_1 \xmapsto{z_2} \ldots \xmapsto{z_k} P_k$ of e, the corresponding prefix $[e]_\varepsilon(k)$ of $[e]_\varepsilon$ is such that $\mathsf{dur}([e]_\varepsilon(k)) = [\mathsf{dur}(e(k))]_\varepsilon$.*

We present the proof after a number of preliminaries.

We first define an operational analogue to the notion of semantic binding. A *syntactic binding* is a function $\eta : \mathit{VAR} \longrightarrow \overline{\mathit{NODE}}_{TF}$. Any open node P together with a syntactic binding η give rise to a closed node $P\eta$ via substitution in the obvious way. We also have standard operations on syntactic bindings, such as $\eta[X := P]$ (for P a closed node), defined in exactly the same manner as in the denotational case.

Let $\{\mathrm{R}_k \mid k \in \mathbb{N}\}$ be a family of relations between closed nodes. This set is an *indexed bisimulation* if, for any $k \geqslant 1$, any $P, P', Q, Q'' \in \overline{NODE}_{TF}$, any $\mu \in \Sigma^{\checkmark} \cup \{\tau\}$, and any $t \geqslant 0$, the following four conditions are met:

$$(P \,\mathrm{R}_k\, Q \wedge P \overset{t}{\rightsquigarrow} P') \Rightarrow \exists\, Q' \centerdot Q \overset{t}{\rightsquigarrow} Q' \wedge P' \,\mathrm{R}_{k-1}\, Q'$$

$$(P \,\mathrm{R}_k\, Q \wedge P \xrightarrow{\mu} P') \Rightarrow \exists\, Q' \centerdot Q \xrightarrow{\mu} Q' \wedge P' \,\mathrm{R}_{k-1}\, Q'$$

$$(P \,\mathrm{R}_k\, Q \wedge Q \overset{t}{\rightsquigarrow} Q'') \Rightarrow \exists\, P'' \centerdot P \overset{t}{\rightsquigarrow} P'' \wedge P'' \,\mathrm{R}_{k-1}\, Q''$$

$$(P \,\mathrm{R}_k\, Q \wedge Q \xrightarrow{\mu} Q'') \Rightarrow \exists\, P'' \centerdot P \xrightarrow{\mu} P'' \wedge P'' \,\mathrm{R}_{k-1}\, Q''.$$

Given an indexed bisimulation $\{\mathrm{R}_k\}$ and two *open* nodes P and Q, we formally write $P \,\mathrm{R}_k\, Q$ if, for any syntactic binding η, $P\eta \,\mathrm{R}_k\, Q\eta$. Nonetheless, unless specified otherwise or clear from the context, indexed bisimulations shall be understood to relate closed rather than open nodes.

Let us now define a family $\{\sim_k \mid k \in \mathbb{N}\}$ as follows: for any $i \geqslant 0$,

$$\sim_i \;\;\widehat{=}\;\; \bigcup\{\mathrm{R} \mid \exists\, \{\mathrm{S}_k \mid k \in \mathbb{N}\} \text{ an indexed bisimulation} \centerdot \mathrm{R} = \mathrm{S}_i\}.$$

We then have the following string of lemmas.

Lemma A.2

1. *$\{\sim_k\}$ is an indexed bisimulation, the strongest such.*
2. *Each $\sim_k$ is an equivalence relation.*
3. *Any nodes P and P' are $\sim_0$-related.*
4. *For any $k, k' \geqslant 0$, if $k \leqslant k'$ then $\sim_{k'} \subseteq \sim_k$.*

Proof

1. Routine inspection.
2. If $\equiv_k$ denotes, for any k, the syntactic identity relation on $\overline{NODE}$, then $\{\equiv_k\}$ is easily seen to be an indexed bisimulation; reflexivity ensues. Symmetry and transitivity follow from the observation that, if $\{\mathrm{R}_k\}$ and $\{\mathrm{R}'_k\}$ are indexed bisimulations, then so are $\{\mathrm{R}_k^{-1}\}$ and $\{\mathrm{R}_k \circ \mathrm{R}'_k\}$.

3. If we let $\mathrm{R}_0 = \overline{NODE} \times \overline{NODE}$, and let, for any $k \geqslant 1$, $\mathrm{R}_k = \emptyset$, we easily find that $\{\mathrm{R}_k\}$ is an indexed bisimulation. The result follows by maximality of $\{\sim_k\}$.

4. For any $k \geqslant 0$, let $\mathrm{R}_k = \bigcup\{\sim_i \mid i \geqslant k\}$. One easily verifies that $\{\mathrm{R}_k\}$ is an indexed bisimulation, and thus that $\mathrm{R}_k \subseteq \sim_k$ for any k; the result follows.

■

If η and η' are syntactic bindings and $k \in \mathbb{N}$, let us write $\eta \sim_k \eta'$ if, for all $X \in \mathit{VAR}$, $\eta(X) \sim_k \eta'(X)$. We then have:

Lemma A.3 *Let $P \in \mathit{NODE}$ and $\eta \sim_k \eta'$ for some syntactic bindings η, η', and some fixed $k \geqslant 0$. Then $P\eta \sim_k P\eta'$.*

Proof (Sketch.) We proceed by induction on k. For a given k, we define a family $\{\approx_n\}$ as follows: for $n \leqslant k$, $\approx_n = \{(R\rho, R\rho') \mid R \in \mathit{NODE} \wedge \rho \sim_n \rho'\}$. And for $n > k$, $\approx_n = \emptyset$. We remark that the definition of $\approx_i$, for any fixed i, is uniform over all $k \geqslant i$.

We aim to show that, for any particular value of k, $\{\approx_n\}$ is an indexed bisimulation. By the previous lemma, this will entail that any pair of nodes that are $\approx_i$-related are in fact $\sim_i$-related. The result will then follow by choosing $i = k$, $P = R$, $\rho = \eta$, and $\rho' = \eta'$.

The base case $k = 0$ is trivial: $\{\approx_n\}$ is clearly—in fact, vacuously—an indexed bisimulation.

For the induction step $(k + 1)$, we proceed by structural induction on R. We need to show that, whenever $R\rho \approx_n R\rho'$, $R\rho$ and $R\rho'$ satisfy the conditions for $\{\approx_n\}$ to be an indexed bisimulation.

The structural induction cases are fairly easy. We tackle parallel composition and recursion in illustration:

Let $R = R_1 \parallel R_2$. We assume that $\rho \sim_{k+1} \rho'$ (for greater values of n than $k+1$ the result vacuously holds, and for lesser values the result follows by the induction hypothesis on k) and thus that $R\rho \approx_{k+1} R\rho'$. Suppose that $R\rho \xrightarrow{a} Q$, for some $a \in \Sigma^{\checkmark}$. Since $R\rho \equiv (R_1 \parallel R_2)\rho \equiv R_1\rho \parallel R_2\rho$, we find, according to the operational rules of inference, that we must have $R_1\rho \xrightarrow{a} Q_1$, $R_2\rho \xrightarrow{a} Q_2$, and $Q \equiv Q_1 \parallel Q_2$.

By the structural induction hypothesis, since $R_1\rho \xrightarrow{a} Q_1$, it must be the case that $R_1\rho' \xrightarrow{a} Q_1'$, with $Q_1 \approx_k Q_1'$. We can now invoke the induction hypothesis on k to conclude that $Q_1 \sim_k Q_1'$. Likewise, since $R_2\rho \xrightarrow{a} Q_2$, it must be the case that $R_2\rho' \xrightarrow{a} Q_2'$, with $Q_2 \sim_k Q_2'$.

Write $Q' \equiv Q_1' \parallel Q_2'$. It is clear that $R\rho' \xrightarrow{a} Q'$.

Now define syntactic bindings ν and ν' as follows: ν and ν' agree on all variables other than X and Y, and $\nu(X) = Q_1$, $\nu(Y) = Q_2$, $\nu'(X) = Q_1'$, and $\nu'(Y) = Q_2'$. Clearly, $\nu \sim_k \nu'$. We have

$$Q \equiv Q_1 \parallel Q_2 \equiv (X \parallel Y)\nu \approx_k (X \parallel Y)\nu' \equiv Q_1' \parallel Q_2' \equiv Q'$$

as required.

The other cases (τ-transitions, evolutions, and symmetry) are entirely similar.

We now consider the case of recursion (in our structural induction).

Note that $(\mu X \centerdot R)\rho \xrightarrow{\tau} R[\mu X \centerdot R/X]\rho$ and likewise $(\mu X \centerdot R)\rho' \xrightarrow{\tau} R[\mu X \centerdot R/X]\rho'$ (and these are the only allowable behaviours). But if $\rho \sim_{k+1} \rho'$ (as we are assuming), then $\rho \sim_k \rho'$ (as per Lemma A.2), and we are done: $R[\mu X \centerdot R/X]\rho \approx_k R[\mu X \centerdot R/X]\rho'$, as required. ■

The following result will not directly be needed elsewhere in this work, but has been included because of its independent central importance in operational semantics, and because its proof is now only a small step away from the results already contained in this section.

Proposition A.4 *For any $k \geqslant 0$, $\sim_k$ is a congruence over Timed CSP syntax. In other words, all Timed CSP operators are preserved by $\sim_k$.*

Proof (Sketch.) One proceeds by case analysis, considering each Timed CSP operator in turn. The result is trivial in the case of nullary operators. For binary operators, and unary operators other than recursion, the argument is reasonably straightforward; we consider the case of external choice as an illustration. We then conclude by tackling recursion.

Let $P \equiv P_1 \Box P_2$ and $Q \equiv Q_1 \Box Q_2$, where for simplicity we are only considering closed nodes. Suppose that $P_1 \sim_k Q_1$ and $P_2 \sim_k Q_2$. We claim that $P \sim_k Q$ follows. To see this, define, for any $n \geqslant 0$,

$$\approx_n \quad = \quad \sim_n \cup \{(R_1 \Box R_2, S_1 \Box S_2) \mid R_1 \sim_n S_1 \wedge R_2 \sim_n S_2\}.$$

It now clearly suffices to show that $\{\approx_n\}$ is an indexed bisimulation. This is a simple inspection. For instance, suppose that $R \approx_n S$ for some $n \geqslant 1$.

If this is already an instance of $\sim_n$, we are done. Otherwise, we must have $R \equiv R_1 \mathbin{\Box} R_2$ and $S \equiv S_1 \mathbin{\Box} S_2$, with $R_1 \sim_n S_1$ and $R_2 \sim_n S_2$. Now consider the various transitions and evolutions available to R. For example, suppose that $(R_1 \mathbin{\Box} R_2) \overset{t}{\rightsquigarrow} R'$, for some some $t \geqslant 0$. Consulting the rules of inference, we conclude that it must have been the case that $R_1 \overset{t}{\rightsquigarrow} R_1'$, $R_2 \overset{t}{\rightsquigarrow} R_2'$, and $R' \equiv R_1' \mathbin{\Box} R_2'$. Since $R_1 \sim_n S_1$, we must have $S_1 \overset{t}{\rightsquigarrow} S_1'$ for some node S_1' with $R_1' \sim_{n-1} S_1'$. Likewise, there must be some node S_2' such that $S_2 \overset{t}{\rightsquigarrow} S_2'$ and $R_2' \sim_{n-1} S_2'$. Writing $S' \equiv S_1' \mathbin{\Box} S_2'$, it then follows that $(S_1 \mathbin{\Box} S_2) \overset{t}{\rightsquigarrow} S'$ and that $R' \approx_{n-1} S'$. Other cases are equally easy; $\{\approx_n\}$ is therefore a time-bisimulation, as required.

The recursion operator is handled by induction on k. Specifically, we show that, for any $P, Q \in \mathit{NODE}$, $P \sim_k Q$ implies that $\mu X \bullet P \sim_k \mu X \bullet Q$.

The base case ($k = 0$) is trivial. For the induction step, we consider open nodes P and Q such that $P \sim_{k+1} Q$. Since this entails that $P \sim_k Q$ (as per Lemma A.2), we can invoke the induction hypothesis to conclude that $\mu X \bullet P \sim_k \mu X \bullet Q$. We must show that in fact $\mu X \bullet P \sim_{k+1} \mu X \bullet Q$.

Let η be some fixed syntactic binding. We have $(\mu X \bullet P)\eta \xrightarrow{\tau} P[\mu X \bullet P/X]\eta \equiv P\eta[X := (\mu X \bullet P)\eta]$. Likewise, $(\mu X \bullet Q)\eta \xrightarrow{\tau} Q\eta[X := (\mu X \bullet Q)\eta]$. So we need to show that $P\eta[X := (\mu X \bullet P)\eta] \sim_k Q\eta[X := (\mu X \bullet Q)\eta]$. But $P\eta[X := (\mu X \bullet P)\eta] \sim_k P\eta[X := (\mu X \bullet Q)\eta]$ by Lemma A.3, since we know that $\mu X \bullet P \sim_k \mu X \bullet Q$. And by definition, $P\eta[X := (\mu X \bullet Q)\eta] \sim_k Q\eta[X := (\mu X \bullet Q)\eta]$, since $P \sim_k Q$. The result follows by transitivity of $\sim_k$. ■

Proposition A.4 easily entails that *strong bisimilarity* (cf., for example, [Mil89]) is a congruence for Timed CSP. This remains true even in non-finitely branching operational semantics, although in that case one must use ordinals in indexed bisimulations rather than simply positive integers. We end our digression here and now return to our main thread.

Let $P \in \mathit{NODE}$, and let η be a syntactic binding. We define a function $F = F_{P,X,\eta} : \mathit{NODE} \longrightarrow \overline{\mathit{NODE}}$, as follows: $F(Q) \mathrel{\widehat{=}} (\mathit{SKIP} \mathbin{;} P)[Q/X]\eta$. In essence, $F_{P,X,\eta}$ mimics the unwinding of recursion in P over the variable X, under the binding η, and then substitutes its argument for all free instances of X in P. This leads to the following:

Lemma A.5 *Let $P \in \mathit{NODE}$ and let η be a syntactic binding. For any $k \in \mathbb{N}$, $F^k(\mathit{STOP}) \sim_k (\mu X \bullet P)\eta$, where $F = F_{P,X,\eta}$ is as above.*

Proof We proceed by induction on k. The base case $k = 0$ is trivial since, according to Lemma A.2, all programs are $\sim_0$-related. For the induction step, we first make the simple observations that, for any nodes R and R', if $R \sim_k R'$ then $(\mathit{SKIP}\ ;\ R) \sim_{k+1} (\mathit{SKIP}\ ;\ R')$, as an immediate calculation of the possible executions easily reveals.

So assume that the result holds for k. We have the following sequence of deductions, select justifications of which follow.

$$\begin{aligned}
F^{k+1}(\mathit{STOP}) &\equiv F(F^k(\mathit{STOP})) \\
&\equiv (\mathit{SKIP}\ ;\ P)[F^k(\mathit{STOP})/X]\eta \\
&\equiv \mathit{SKIP}\ ;\ (P[F^k(\mathit{STOP})/X]\eta) \\
&\equiv \mathit{SKIP}\ ;\ (P(\eta[X := F^k(\mathit{STOP})])) \\
&\sim_{k+1} \mathit{SKIP}\ ;\ (P(\eta[X := (\mu X \centerdot P)\eta])) && (1) \\
&\equiv \mathit{SKIP}\ ;\ (P[(\mu X \centerdot P)\eta/X]\eta) \\
&\sim_{k+1} (\mu X \centerdot P)\eta. && (2)
\end{aligned}$$

This concludes the induction step. (1) follows from our earlier observation together with Lemma A.3, using the fact that $F^k(\mathit{STOP}) \sim_k (\mu X \centerdot P)\eta$ by the induction hypothesis. (2) rests on the observation that $\mathit{SKIP}\ ;\ (P[(\mu X \centerdot P)\eta/X]\eta) \sim_n (\mu X \centerdot P)\eta$ for *any* $n \geqslant 0$, as a straightforward calculation will attest. ■

Lemma A.6 *Let $P, Q \in \overline{\mathit{NODE}}$, $k \geqslant 1$, and suppose that $P \sim_k Q$. Then, for any execution $e = P_0 \xmapsto{z_1} P_1 \xmapsto{z_2} \ldots \xmapsto{z_n} P_n$ of P with $n < k$, there is a corresponding execution $u = Q_0 \xmapsto{z_1} Q_1 \xmapsto{z_2} \ldots \xmapsto{z_n} Q_n$ of Q, with identical transitions and evolutions, and such that, for all $0 \leqslant i \leqslant n$, $\mathsf{init}(P_i) = \mathsf{init}(Q_i)$.*

The executions e and u are said to be *bisimilar.*

Proof (Sketch.) This is a routine induction on n; the inductive statement is strengthened to require that $P_i \sim_{k-i} Q_i$, for all i. ■

We can now give the proof of Lemma A.1.

Proof Let us agree that programs satisfying the conclusion of the lemma be termed *digitisable.* Recall also that, given an execution e and a pivot ε as in the lemma statement, the execution $[e]_\varepsilon$ is called the ε-digitisation of e.

We proceed by structural induction on P, as follows: for any syntactic binding η such that $\eta(X)$ is a digitisable $\overline{\textbf{TCSP}}$ program for all $X \in \mathit{VAR}$, we claim that $P\eta$ is digitisable as well. Here $P \in \mathbf{TCSP}$, and in particular only contains integral delays. The principal cases are presented below.

case $WAIT\ n$**:** The result follows easily from the fact that n is an integer.

case $a \longrightarrow P$**:** Let e be an execution of $(a \longrightarrow P)\eta \equiv a \longrightarrow P\eta$, and let ε be fixed. If e is either trivial or begins immediately with an a-transition, the result follows by the induction hypothesis. On the other hand, a string of initial evolutions give rise to no greater difficulties, thanks to Rule 3.10, than a single initial evolution; thus let us assume for simplicity that $e = (a \longrightarrow P\eta) \overset{t}{\rightsquigarrow} (a \longrightarrow P\eta) \xrightarrow{a} e'$, where $e' \in \mathsf{exec}(P\eta)$.

Writing again $\dot{t}$ to designate the fractional part of t, we now apply the induction hypothesis to P, with syntactic binding η, execution e', and pivot $\varepsilon' = \varepsilon - \dot{t} + [\dot{t}]_\varepsilon$, to get the digitisation $[e']_{\varepsilon'}$ of e'. It is then easy to verify that the execution

$$e_\varepsilon = (a \longrightarrow P)\eta \overset{[t]_\varepsilon}{\rightsquigarrow} (a \longrightarrow P)\eta \xrightarrow{a} {}^\frown[e']_{\varepsilon'}$$

is itself a valid ε-digitisation of e, i.e., $e_\varepsilon = [e]_\varepsilon$.

Let us omit explicit mention of the syntactic binding η in the next three cases to alleviate the notation.

case $P_1 \,\Box\, P_2$**:** An execution of $P_1 \,\Box\, P_2$ is eventually only an execution of P_1 or of P_2, at which point the induction hypothesis finishes up the job. What must therefore be examined is what happens in the interim, namely prior to the commitment to one or the other process in the external choice construct. Consulting the relevant operational rules reveals that only evolutions and τ-transitions are permitted during that period. But then we can again apply the induction hypothesis to both processes to turn such evolutions into integral-duration ones, ensuring in doing so that the relevant τ-transitions occur at the required integral times.

case $P_1 \underset{B}{\|} P_2$**:** The crux of the argument in this case is similar to that of the previous one, and rests on the following observation: if P_1 and P_2 can synchronise on event a at time t, and both can also independently communicate a at time $\lfloor t \rfloor$ (or $\lceil t \rceil$), then both can also synchronise on a at time $\lfloor t \rfloor$ (or $\lceil t \rceil$).

case $P \setminus A$**:** The only source of difficulty here stems from the fact that, while a process which only offers visible events can perfectly well choose instead to evolve, a process offering a hidden event must perform *something* (rather than evolve) on the spot. Since hidden events of $P \setminus A$ potentially have A-transitions of P for origin, the worry is that while these A-transitions may have been delayable in P, as urgent hidden events in $P \setminus A$ they are required to occur as soon as possible.

The key observation to make is that, thanks to Rule 3.28, any execution of $P \setminus A$ derives from an execution of P in which evolutions took place solely when no A-transitions were available; and conversely, any such execution of P gives rise to a valid execution of $P \setminus A$.

So let e be an execution of $P \setminus A$, deriving directly from such a suitable execution u of P, so that every A-transition in u has been turned into a τ-transition in e (and with e and u otherwise transition- and evolution-wise identical). Apply the induction hypothesis to P and u, yielding digitisation u'. Since by definition the matching nodes of u and u' have the same events available to them, we conclude that evolutions in u' also only take place while no A-transition is enabled. u' thus validly gives rise, through hiding, to an execution of $P \setminus A$, one which, by construction, is a digitisation of e.

case X**:** This case follows immediately from our assumptions on η: $X\eta \equiv \eta(X)$ is required to be digitisable.

case $\mu X \bullet P$**:** Let $F = F_{P,X,\eta}$ be the recursion-unwinding-mimicking function defined earlier: $F(Q) = (\mathit{SKIP} \ ; \ P)[Q/X]\eta$. We first show that $F^k(\mathit{STOP})$ is a digitisable program for all $k \in \mathbb{N}$. This simple induction on k has a trivial base case of $k = 0$ since $F^0(\mathit{STOP}) \equiv \mathit{STOP}$. For the induction step, assuming the result holds for k, we write

$$\begin{aligned}
F^{k+1}(\mathit{STOP}) &\equiv F(F^k(\mathit{STOP})) \\
&\equiv (\mathit{SKIP} \ ; \ P)[F^k(\mathit{STOP})/X]\eta \\
&\equiv (\mathit{SKIP} \ ; \ P)\eta[X := F^k(\mathit{STOP})] \\
&\equiv \mathit{SKIP} \ ; \ (P\eta[X := F^k(\mathit{STOP})]).
\end{aligned}$$

By the induction hypothesis on k and our assumptions on η, $\eta[X := F^k(\mathit{STOP})]$ is itself a syntactic binding taking variables to digitisable programs. We can thus invoke the induction hypothesis on P to conclude that $P\eta[X := F^k(\mathit{STOP})]$ is digitisable as well, and thus clearly that so is $F^{k+1}(\mathit{STOP})$, as required. This completes the induction on k.

Let us now consider an execution e of $(\mu X \bullet P)$ containing n transitions and evolutions, and let pivot ε be fixed. By Lemma A.5, we have that $F^{n+1}(STOP) \sim_{n+1} (\mu X \bullet P)\eta$. We can thus invoke Lemma A.6 to extract an execution $u \in F^{n+1}(STOP)$ bisimilar to e. But since $F^{n+1}(STOP)$ is digitisable, we can find a digitisation $u' \in \mathsf{exec}(F^{n+1}(STOP))$ of u. Invoking Lemma A.6 again, there exists an execution e' of $(\mu X \bullet P)\eta$ which is bisimilar to u'. It is easy to see that e' is a digitisation of e, as required.

This completes the induction step, and hence the proof, since it is obvious that syntactic bindings are irrelevant insofar as the set of executions of *programs* are concerned. ■

A.2 Proofs for Chapter 5

Proposition A.7 (5.2) *For any term $P \in$ **TCSP**, and any semantic binding ρ, $\mathcal{R}[\![P]\!]\rho \in \mathcal{M}_R$.*

Proof The proof is a structural induction over terms. One must verify in turn that each term, under an arbitrary semantic binding ρ, satisfies the axioms of $\mathcal{M}_R$. Cases are in general tedious but straightforward, so let us only consider a few illustrating examples (the most interesting of which being that of the hiding operator).

case $P_1 \Box P_2$**:** The induction hypothesis states that $\mathcal{R}[\![P_1]\!]\rho, \mathcal{R}[\![P_2]\!]\rho \in \mathcal{M}_R$. Therefore, by *R1*, $\langle\bullet\rangle \in \mathcal{R}[\![P_1]\!]\rho \cap \mathcal{R}[\![P_2]\!]\rho$, and clearly $\mathsf{trace}(\langle\bullet\rangle) = \langle\rangle \leq \langle tock\rangle^\infty$. Thus $\langle\bullet\rangle \in \mathcal{R}[\![P_1 \Box P_2]\!]\rho$, establishing *R1*.

R2, *R3*, *R5*, *R6*, *R7*, and *R8* also follow immediately from the induction hypothesis. To establish *R4*, consider a refusal $A \neq \bullet$, and assume that $\hat{u}^\frown\langle A\rangle^\frown\check{v} \in \mathcal{R}[\![P_1 \Box P_2]\!]\rho$ and $\hat{u}^\frown\langle A, a, \bullet\rangle \notin \mathcal{R}[\![P_1 \Box P_2]\!]\rho$. We can discount the possibility that $a = tock$ since that would quickly lead us to conclude, via *R4* and the induction hypothesis, that one of $\mathcal{R}[\![P_1]\!]\rho$ or $\mathcal{R}[\![P_2]\!]\rho$ has tests in which *tock* is refused, which in turn would violate the induction hypothesis since, by definition, *tock* cannot appear in the refusal of a test and $\mathcal{M}_R$ exclusively contains sets of tests. Three subcases now arise, of which we only consider the following: $\mathsf{trace}(u) \leq \langle tock\rangle^\infty$ and $\mathsf{trace}(v) \not\leq \langle tock\rangle^\infty$. Moreover, for simplicity let us assume that in fact the first element in $\mathsf{trace}(v)$ is not a *tock*. The first condition implies that $\hat{u}^\frown\langle A\rangle \in \mathcal{R}[\![P_1]\!]\rho \cap \mathcal{R}[\![P_2]\!]\rho$.

Since $\hat{u}^\frown\langle A, a, \bullet\rangle \notin \mathcal{R}[\![P_1 \,\Box\, P_2]\!]\rho$ and $a \neq \mathit{tock}$, $\hat{u}^\frown\langle A, a, \bullet\rangle \notin \mathcal{R}[\![P_1]\!]\rho$ and $\hat{u}^\frown\langle A, a, \bullet\rangle \notin \mathcal{R}[\![P_2]\!]\rho$. Hence, invoking the induction hypothesis and *R4*, $\hat{u}^\frown\langle A \cup \{a\}\rangle \in \mathcal{R}[\![P_1]\!]\rho \cap \mathcal{R}[\![P_2]\!]$. Since $\hat{u}^\frown\langle A\rangle^\frown\check{v} \in \mathcal{R}[\![P_1 \,\Box\, P_2]\!]\rho$ and $\mathsf{trace}(v) \nleq \langle \mathit{tock}\rangle^\infty$, $\hat{u}^\frown\langle A\rangle^\frown\check{v} \in \mathcal{R}[\![P_1]\!]\rho \cup \mathcal{R}[\![P_2]\!]\rho$. By what has been established and the induction hypothesis, this entails that $\hat{u}^\frown\langle A \cup \{a\}\rangle^\frown\check{v} \in \mathcal{R}[\![P_1]\!]\rho \cup \mathcal{R}[\![P_2]\!]\rho$, from which we easily conclude, collecting the various facts and assumptions above, that $\hat{u}^\frown\langle A \cup \{a\}\rangle^\frown\check{v} \in \mathcal{R}[\![P_1 \,\Box\, P_2]\!]\rho$. This establishes *R4*.

Therefore $\mathcal{R}[\![P_1 \,\Box\, P_2]\!]\rho \in \mathcal{M}_R$.

case $P \setminus A$**:** It is easy to see that *R1*, *R5*, *R6*, *R7*, and *R8* follow quickly from the induction hypothesis.

To see that *R2* holds, consider an arbitrary test $v \in \mathcal{R}[\![P \setminus A]\!]\rho$. If v' is a test such that $v' \prec v \wedge \mathsf{trace}(v') = \mathsf{trace}(v)$, then $v' \in \mathcal{R}[\![P \setminus A]\!]\rho$ by application of RefCl. Otherwise (still with $v' \prec v$) we must have $\mathsf{trace}(v') \leq \mathsf{trace}(v)$. Since $v \in \mathcal{R}[\![P \setminus A]\!]\rho$, there exists u an A-urgent test in $\mathcal{R}[\![P]\!]\rho$ such that $v = u \setminus A$. It is clear from the definition of $\setminus$ on tests that there must therefore be a test u' such that $u' \leq u$ and $\mathsf{trace}(v') = \mathsf{trace}(u' \setminus A)$. It is equally clear that u' must be A-urgent since u is, and that $v' \prec u' \setminus A$. By the induction hypothesis and *R2*, $u' \in \mathcal{R}[\![P]\!]\rho$, and thus by the above and another application of RefCl we get that $v' \in \mathcal{R}[\![P \setminus A]\!]\rho$, as required.

For *R3*, let $v \in \mathcal{R}[\![P \setminus A]\!]\rho$ be a test whose last refusal is $\bullet$. Then there exists $u \in \mathcal{R}[\![P]\!]\rho$ an A-urgent test such that $v \prec u \setminus A$—in fact there is clearly no harm in assuming $v = u \setminus A$. Let us denote the last refusal of u by B. By *R3* and the induction hypothesis, we can assume that $B \neq \bullet$. By definition of $\setminus$ on tests we then have that $A \nsubseteq B$ (since the last refusal of $u \setminus A$ is $\bullet$). We now invoke *R4* (repeatedly if needed; strictly speaking this also involves an implicit use of *R2*) to replace B in u by a maximal refusal B', something we can achieve since Σ is finite. If $A \subseteq B'$, then by definition of $\setminus$ the last refusal of $(\hat{u}^\frown\langle B'\rangle) \setminus A$ is B', and clearly $\hat{u}^\frown\langle B'\rangle$ is A-urgent (since u was) and belongs to $\mathcal{R}[\![P]\!]\rho$ by construction. Thus $(\hat{u}^\frown\langle B'\rangle) \setminus A \in \mathcal{R}[\![P \setminus A]\!]\rho$, and only differs from v in its last refusal. An application of RefCl would then yield $\hat{v}^\frown\langle\emptyset\rangle \in \mathcal{R}[\![P \setminus A]\!]\rho$.

We must however also consider the alternative, namely $A \nsubseteq B'$. By the induction hypothesis and the contrapositive of *R4*, recalling that B' is maximal, we conclude that there exists $a \in A$ such that $\hat{u}^\frown\langle B', a, \bullet\rangle \in \mathcal{R}[\![P]\!]\rho$. Note that since $a \neq \mathit{tock}$ by definition (*tock*s cannot be hidden), the test $\hat{u}^\frown\langle B', a, \bullet\rangle$ is A-urgent since u is. Moreover, we clearly have $(\hat{u}^\frown\langle B, a, \bullet\rangle) \setminus A = u \setminus A = v$. We then repeat the above algorithm, namely invoke *R3* and *R4* to find a maximal non-$\bullet$ refusal B'' such that $\hat{u}^\frown\langle B', a, B''\rangle \in \mathcal{R}[\![P]\!]\rho$. If $A \subseteq B''$, as argued above, we we are done. Otherwise we keep repeating the process until the desirable situation is reached. The reason this algorithm must eventually terminate successfully is that, if it did not, it would imply that there is an unbounded sequence of events, all drawn from the set A, which can be performed after u (amongst the tests of $\mathcal{R}[\![P]\!]\rho$). This is in direct contradiction with Axiom *R7*, assumed to hold of $\mathcal{R}[\![P]\!]\rho$ by the induction hypothesis. This establishes that $\mathcal{R}[\![P \setminus A]\!]\rho$ satisfies *R3*.

Lastly, we tackle *R4*. Let $B \neq \bullet$, and suppose that $\hat{v}^\frown\langle B\rangle^\frown\check{w} \in \mathcal{R}[\![P \setminus A]\!]\rho$ and $\hat{v}^\frown\langle B, b, \bullet\rangle \notin \mathcal{R}[\![P \setminus A]\!]\rho$. We want to conclude that $\hat{v}^\frown\langle B \cup \{b\}\rangle^\frown\check{w} \in \mathcal{R}[\![P \setminus A]\!]\rho$. Let us assume that the test $\hat{v}^\frown\langle B\rangle^\frown\check{w}$ is maximal with this property; in particular, it did not arise as an application of RefCl. (It is harmless to make this assumption since we can later 'degrade' it via RefCl if we want, once the desired conclusion has been reached.) Since $B \neq \bullet$, we must have that $A \subseteq B$, by definition of $\setminus$ on tests and maximality of B. Since $\hat{v}^\frown\langle B\rangle^\frown\check{w} \in \mathcal{R}[\![P \setminus A]\!]\rho$, there must be A-urgent tests u, u' such that $v = u \setminus A$, $w = u' \setminus A$, and $\hat{u}^\frown\langle B\rangle^\frown\check{u}' \in \mathcal{R}[\![P]\!]\rho$. Recall that $\hat{v}^\frown\langle B, b, \bullet\rangle \notin \mathcal{R}[\![P \setminus A]\!]\rho$. Therefore we also must have $\hat{u}^\frown\langle B, b, \bullet\rangle \notin \mathcal{R}[\![P]\!]\rho$, since $b \notin A$ as $A \subseteq B$ and $\mathcal{R}[\![P]\!]\rho$ is assumed to obey Axiom *R6*. Invoking *R4*, we conclude that $\hat{u}^\frown\langle B \cup \{b\}\rangle^\frown\check{u}' \in \mathcal{R}[\![P]\!]\rho$. This test is clearly A-urgent, and by construction $(\hat{u}^\frown\langle B \cup \{b\}\rangle^\frown\check{u}') \setminus A = \hat{v}^\frown\langle B \cup \{b\}\rangle^\frown\check{w} \in \mathcal{R}[\![P]\!]\rho$ as required.

We conclude that $\mathcal{R}[\![P \setminus A]\!]\rho \in \mathcal{M}_R$.

case $f(P)$**:** There are no deep-seated difficulties with this operator. As an illustration, we establish Axioms *R4* and *R5*. For the former, assume that $A \neq \bullet$, and suppose that $\hat{u}^\frown\langle A\rangle^\frown\check{v} \in \mathcal{R}[\![f(P)]\!]\rho$ and $\hat{u}^\frown\langle A, a, \bullet\rangle \notin \mathcal{R}[\![f(P)]\!]\rho$. We claim that $\hat{u}^\frown\langle A \cup \{a\}\rangle^\frown\check{v} \in \mathcal{R}[\![f(P)]\!]\rho$. Let us say that the test $\hat{u}^\frown\langle A\rangle^\frown\check{v} \in \mathcal{R}[\![f(P)]\!]\rho$ derives in the obvious manner from the test $\hat{u'}^\frown\langle f^{-1}(A)\rangle^\frown\check{v'} \in \mathcal{R}[\![P]\!]\rho$. Let $C = f^{-1}(a)$. Since $\hat{u}^\frown\langle A, a, \bullet\rangle \notin \mathcal{R}[\![f(P)]\!]\rho$, it must be the case that, for all $c \in C$, $\hat{u'}^\frown\langle f^{-1}(A), c, \bullet\rangle \notin \mathcal{R}[\![P]\!]\rho$. Then, by repeatedly invoking Axiom *R4* (and, implicitly, *R2*), recalling that Σ (and hence C) is finite, we get $\hat{u'}^\frown\langle f^{-1}(A) \cup C\rangle^\frown\check{v'} \in \mathcal{R}[\![P]\!]\rho$. Since $f^{-1}(A) \cup C = f^{-1}(A) \cup f^{-1}(a) = f^{-1}(A \cup \{a\})$, we get that $\hat{u}^\frown\langle A \cup \{a\}\rangle^\frown\check{v} \in \mathcal{R}[\![f(P)]\!]\rho$ as required.

Let us now consider the case of *R5*. Let $\hat{v}^\frown\langle A, f(a), \bullet\rangle \in \mathcal{R}[\![f(P)]\!]\rho$, and assume $A \neq \bullet$. We claim that $\hat{v}^\frown\langle A, \mathit{tock}, \bullet, f(a), \bullet\rangle \in \mathcal{R}[\![f(P)]\!]\rho$.

Our hypothesis implies that $\hat{u}^\frown\langle f^{-1}(A), a, \bullet\rangle \in \mathcal{R}[\![P]\!]\rho$, where u is any test corresponding to v under the semantic application of f. Since $A \neq \bullet$, $f^{-1}(A) \neq \bullet$. By the induction hypothesis, we can invoke *R5* on $\mathcal{R}[\![P]\!]\rho$ to conclude that $\hat{u}^\frown\langle f^{-1}(A), \mathit{tock}, \bullet, a, \bullet\rangle \in \mathcal{R}[\![P]\!]\rho$. It then follows that $\hat{v}^\frown\langle A, \mathit{tock}, \bullet, f(a), \bullet\rangle \in \mathcal{R}[\![f(P)]\!]\rho$, as required.

case $\mu X \centerdot P$**:** Our definition of fix makes checking this case trivial: by the induction hypothesis, $\mathcal{R}[\![P]\!]\rho \in \mathcal{M}_R$ for *any* semantic binding ρ, and therefore $\lambda x.\mathcal{R}[\![P]\!](\rho[X := x])$ is clearly an $\mathcal{M}_R$ selfmap. Applying fix yields an element of $\mathcal{M}_R$, since fix is defined even when its argument does not have a unique fixed point. ■

Theorem A.8 (Banach fixed point theorem) *If $F : M \longrightarrow M$ is a contraction on a non-empty complete metric space, then F has a unique fixed point $a \in M$. Moreover, for any $b \in M$, $a = \lim_{i\to\infty} F^i(b)$.*

This theorem is also known as the *contraction mapping theorem.* A proof can be found in any standard Analysis or Topology text, such as [Mar74].

Lemma A.9 (5.6) *Every term in* **TCSP** *is (i.e., corresponds to) a non-expanding function. Moreover, if $P = P(X, \vec{Y})$ is a term which is time-guarded for X, then P is a contraction in X.*

Proof We proceed, for the first part, by structural induction over the structure of terms. By lemma 5.5 it suffices to show that every Timed CSP operator is non-expanding in each of its arguments. In case of the μX operator, we shall assume the second part, which we will prove independently afterwards.

It is plain that all nullary operators (such as $STOP$) are non-expanding as they have no free variables. The proof that the other operators (apart from recursion) are also non-expanding is straightforward; we shall examine the case of interleaving as a typical representative. Note that hiding, usually the most troublesome operator in cpo-based models, is easily seen to be non-expanding as *tock*s cannot be hidden.

Let $Q \in \mathcal{M}_R$ be fixed, and consider $|||_Q(\cdot) = (\cdot) \mathbin{|||} Q : \mathcal{M}_R \longrightarrow \mathcal{M}_R$. Let $P, P' \in \mathcal{M}_R$ be such that $d(P, P') = 2^{-k}$ for some $k \in \mathbb{N}$. Let $v \in |||_Q(P)$ be such that $\sharp(v \upharpoonright \mathit{tock}) \leqslant k$. By definition of $|||$ on processes there must be $u_1 \in P$, $u_2 \in Q$ such that $v \in u_1 \mathbin{|||} u_2$. It is easy to see, from the definition of $|||$ on tests, that $\sharp(u_1 \upharpoonright \mathit{tock}) \leqslant \sharp(v \upharpoonright \mathit{tock}) \leqslant k$ since u_1 and u_2 synchronise on every *tock* to produce *tock*s in v. (The only exception is if $\checkmark$ happens to be communicated by u_2, which then makes the remainder of u_1 irrelevant. But then we can replace u_1 by a prefix $u_1' \in P$ of duration less than or equal to the duration of u_2, which itself must then be less than or equal to k, with $v \in u_1' \; ||| \; u_2$ still holding.)

Since $d(P, P') \leqslant 2^{-k}$, it follows from the above that $u_1 \in P'$, and thus that $v \in |||_Q(P')$. Since this holds for arbitrary v of duration less than or equal to k, we conclude that $|||_Q(P)(k) \subseteq |||_Q(P')(k)$. Likewise the reverse containment holds, and hence $d(|||_Q(P), |||_Q(P')) \leqslant k$, establishing that $|||$ is non-expanding in its first argument. The case of $|||^Q(\cdot) = Q \; ||| \; (\cdot)$ is entirely symmetric. It therefore follows that $|||$ is non-expanding, as required.

The case of the μX operator is slightly more involved. Recall that our syntax requires us to apply it only to terms $P = P(X, Y, \vec{Z})$ which are time-guarded for X. We shall assume, and then prove afterwards, that every such P corresponds to a contraction in X over $\mathcal{M}_R$. The induction hypothesis, on the other hand, asserts that P is non-expanding in each of its variables. (Note that if X is the only free variable of P, then $\mu X \bullet P$, being a constant, is trivially non-expanding.)

So let $\vec{S} \in \mathcal{M}_R \times \ldots \times \mathcal{M}_R$ be fixed, and let $F_{\vec{S}}(Y) = \mu X \bullet P(X, Y, \vec{S}) : \mathcal{M}_R \longrightarrow \mathcal{M}_R$. Let $Q, Q' \in \mathcal{M}_R$ be such that $d(Q, Q') \leqslant 2^{-k}$. By definition of μX, $F_{\vec{S}}(Q) = \mathsf{fix}(\lambda x . P(x, Q, \vec{S}))$. By our assumption above, this represents an application of fix to a contraction mapping having, by the Banach fixed point theorem (Theorem A.8), a unique fixed point $P_{\mathit{lim}} = P(P_{\mathit{lim}}, Q, \vec{S}) = F_{\vec{S}}(Q)$. In addition, if $P_0 \in \mathcal{M}_R$ is arbitrary, the Banach fixed point theorem asserts that the sequence $\langle P_0, P_1 = P(P_0, Q, \vec{S}), P_2 = P(P_1, Q, \vec{S}), \ldots, P_{i+1} = P(P_i, Q, \vec{S}), \ldots \rangle$ converges to P_{lim}.

Similarly, we let P'_{lim} be the unique fixed point of the contraction mapping P calculated using Q' instead of Q, i.e., $P'_{\mathit{lim}} = F_{\vec{S}}(Q')$. Again, we have a sequence converging to P'_{lim}, where the starting point $P'_0 = P_0$ is the same one as above: $\langle P'_0, P'_1 = P(P'_0, Q', \vec{S}), P'_2 = P(P'_1, Q', \vec{S}), \ldots, P'_{i+1} = P'(P'_i, Q', \vec{S}), \ldots \rangle$. Invoking the induction hypothesis, and recalling that $d(Q, Q') \leqslant 2^{-k}$, it is easy to see that $d(P_i, P'_i) \leqslant 2^{-k}$ for all $i \geqslant 0$; from this one concludes that the respective limits of the two sequences cannot be further apart than 2^{-k} either, which shows that $F_{\vec{S}}(Y) = \mu X \bullet P(X, Y, \vec{S})$ is non-expanding in Y as required. **TCSP** terms therefore indeed correspond to non-expanding functions.

We now establish the second part of the lemma, namely the assertion that, if P is time-guarded for X, then P is a contraction in X. (Here the variable X is arbitrary, but fixed.) We will prove this result by means of a double induction, as follows. Our first induction will be on the number of $\mu\, Z$ operators appearing in P, for any $Z \in \mathit{VAR}$. Within each such step, we will carry out a structural induction over the recursive definition of time-guardedness (Definition 2.2). Our precise inductive statement is: 'If P contains k recursion operators and P is time-guarded for X, then P represents a contraction in X'. We aim to show that, for each $k \geqslant 0$, this statement is in fact true of terms which are time-guarded in X. This second part is accomplished via the structural induction.

The base case $k = 0$ corresponds to terms P which are recursion-free:

- Clearly, STOP, SKIP, and $\mathit{WAIT}\ n$ are contractions in X as they do not have X as a free variable. The case $\mu\, X \bullet P$ is vacuously true as it contains one (rather than zero) recursion operator.

- If $Y \neq X$, then Y is a contraction in X, again because X does not appear as a free variable in it.

- Assume that the induction hypothesis holds for P, and suppose that the antecedent holds (i.e., P contains no recursion operators and is time-guarded for X), otherwise the result follows vacuously. Then $a \longrightarrow P$, $P \setminus A$, $f^{-1}(P)$, and $f(P)$ are clearly contractions in X, since, as shown earlier, the relevant operators are all non-expanding—note that this result was obtained independently of any assumption about recursion—and since, as per lemma 5.5, the composition of a contraction and a non-expanding function produces a contraction.

  The case $\mu\, Y \bullet P$ is vacuously true as the term contains a recursion operator.

- The conclusion also follows in the cases of timeout, external and internal choice, parallel and sequential composition, and interleaving, since those operators were all independently shown to be non-expanding.

- Lastly, assume that the induction hypothesis holds for P, and that P is time-guarded for X and time-active. Assume furthermore that neither P nor Q contains a recursion operator (otherwise the result is vacuously true). We want to conclude that $P\ ;\ Q$ is a contraction in X.

  Since Q is a term not containing a recursion operator, we can invoke the first part of this lemma to conclude that it represents a non-expanding function, as that part, once again, is easily seen to have been established independently of any assumption about recursion. The induction hypothesis and our assumption that P is time-guarded for X imply that P is a contraction in X. Let us write $P = P(X)$ and $Q = Q(X)$, ignoring for simplicity any other free variable (which at any rate would be considered fixed). One can then prove by a simple structural induction over the recursive definition of time-activity that behaviours of duration j in $R \in \mathcal{M}_R$ give rise to behaviours of duration at least $j+1$ in $P(R)$; $Q(R)$: it suffices to show that behaviours of $P(R)$; $Q(R)$ consist of behaviours of $P(R)$ of duration at least 1, followed by behaviours of $Q(R)$, and then recall that P is a contraction and that Q is non-expanding. One then concludes that P ; Q must be a contraction mapping in X with contraction factor at most 1/2. We omit the details.

This concludes the base case.

The induction step proceeds in a nearly identical manner, except that we can now assume that time-guarded terms containing k or fewer recursion operators correspond to contraction mappings. We can then repeat the sort of reasoning involving the Banach fixed point theorem which we carried out earlier to discharge our proof obligations. As an illustration, let us show how to handle the case $\mu Y \bullet P$ in the recursive definition of time-guardedness in X.

Our induction hypothesis states that, whenever P' is time-guarded in X and contains k or fewer recursion operators, then P' represents a contraction in X. We now assume that the term $\mu Y \bullet P$ contains $k+1$ recursion operators and that P is time-guarded in X; we must show that $\mu Y \bullet P$ corresponds to a contraction in X.

Observe that $P = P(Y, X, \vec{Z})$ must contain exactly k recursion operators. Since it is time-guarded in X, we can apply the induction hypothesis and conclude that P is a contraction in X. Moreover, P must be time-guarded in Y since our syntactic rules only allow an application of the operator μY to such terms. Make a change of free variables in order to rewrite $P(Y, X, \vec{Z})$ as $P'(X, W, \vec{Z})$, where $W \in \mathit{VAR}$ is a fresh variable. As P was time-guarded in Y, P' is time-guarded in X, and clearly contains k recursion operator. Applying the induction hypothesis again, we conclude that P' corresponds to a contraction in X, and therefore that P is a contraction in Y. Calculating the fixed point by successive iterations, as we did earlier, enables us to conclude that $\mu Y \bullet P$ is non-expanding. And since P is a contraction in X, the same iteration technique easily yields that $\mu\, Y \bullet P$ is also a contraction in X, which is what we were required to show.

This completes the induction step and the proof of the lemma. ■

Proposition A.10 (5.7) *Let $P = P(X, \vec{Y})$ be a* **TCSP** *term which is time-guarded for X. Then $\mu\, X \bullet P$ is the unique function $F = F(\vec{Y})$ such that $F(\vec{Y}) = P(F(\vec{Y}), \vec{Y})$.*

Proof This follows immediately from the previous lemma and the Banach fixed point theorem. If $G = G(\vec{Y})$ is any other function that is a fixed point of P when substituted for X, we find that, whenever we fix $\vec{Y} = \vec{S}$, we must have $F(\vec{S}) = G(\vec{S})$ by the uniqueness part of the Banach fixed point theorem and the fact that $P_{\vec{S}}(X) : \mathcal{M}_R \longrightarrow \mathcal{M}_R$ is a contraction. Therefore $F = G$ as claimed. ■

Proposition A.11 (5.13) *Finite variability—a* program $P \in \overline{\textbf{TCSP}}$ *cannot perform unboundedly many actions within a finite number of tocks:*

$$\forall\, k \geqslant 0 \bullet \exists\, n = n(P, k) \bullet \forall\, tr \in (\Sigma^{\checkmark}_{tock} \cup \{\tau\})^{\star} \bullet$$
$$(P \overset{tr}{\Longrightarrow} \wedge\ \sharp(tr \upharpoonright tock) \leqslant k) \Rightarrow \sharp tr \leqslant n.$$

We first require a number of preliminaries, definitions and lemmas alike.

The notion of *syntactic binding* which we will use is quite similar to that introduced in the previous section: it consists in functions $\eta : VAR \longrightarrow \overline{\textbf{TCSP}}$. Note however that, since well-timedness is here a crucial property, we cannot merely constrain the range of our bindings to $\overline{NODE}_R$.

We have standard applications of bindings to terms, substitutions, etc.

We now define the notion of *time-bisimulation*, an analogue to indexed bisimulations which keeps tracks of time elapsed (number of *tock*s communicated), rather than number of transitions and evolutions witnessed. Since our treatment is in many respects similar to that of indexed bisimulations, our exposition will be terser; omitted proofs, for instance, should be understood to proceed here more or less as they did earlier.

Let $\{\mathrm{R}^k \mid k \in \mathbb{N}\}$ be a family of relations between programs. This set is a *time-bisimulation* if, for any $k \geqslant 1$, any $P, P', Q, Q'' \in \overline{\textbf{TCSP}}$, and any $\mu \in \Sigma^{\checkmark} \cup \{\tau\}$, the following four conditions are met:

$$(P \mathrm{R}^k Q \wedge P \xrightarrow{tock} P') \Rightarrow \exists Q' \centerdot Q \xrightarrow{tock} Q' \wedge P' \mathrm{R}^{k-1} Q'$$
$$(P \mathrm{R}^k Q \wedge P \xrightarrow{\mu} P') \Rightarrow \exists Q' \centerdot Q \xrightarrow{\mu} Q' \wedge P' \mathrm{R}^k Q'$$
$$(P \mathrm{R}^k Q \wedge Q \xrightarrow{tock} Q'') \Rightarrow \exists P'' \centerdot P \xrightarrow{tock} P'' \wedge P'' \mathrm{R}^{k-1} Q''$$
$$(P \mathrm{R}^k Q \wedge Q \xrightarrow{\mu} Q'') \Rightarrow \exists P'' \centerdot P \xrightarrow{\mu} P'' \wedge P'' \mathrm{R}^k Q''.$$

Given a time-bisimulation $\{\mathrm{R}^k\}$ and two *terms* P and Q, we formally write $P \mathrm{R}^k Q$ if, for any syntactic binding η, $P\eta \mathrm{R}^k Q\eta$.

We define a family $\{\sim^k \mid k \in \mathbb{N}\}$ as follows: for any $i \geqslant 0$,

$$\sim^i \;\widehat{=}\; \bigcup\{\mathrm{R} \mid \exists \{\mathrm{S}^k \mid k \in \mathbb{N}\} \text{ a time-bisimulation} \centerdot \mathrm{R} = \mathrm{S}^i\}.$$

We now have:

Lemma A.12

1. *$\{\sim^k\}$ is a time-bisimulation, the strongest such.*
2. *Each $\sim^k$ is an equivalence relation.*
3. *Any nodes P and P' are $\sim^0$-related.*
4. *For any $k, k' \geqslant 0$, if $k \leqslant k'$ then $\sim^{k'} \subseteq \sim^k$.*

Lemma A.13 *For any $k \geqslant 0$, $\sim^k$ is preserved by all Timed CSP operators other than recursion—in other words, $\sim^k$ is a congruence with respect to these operators.*[1]

We omit the straightforward proof, which proceeds in a manner very similar to that of Proposition A.4.

Let $\{\mathrm{R}^k \mid k \in \mathbb{N}\}$ be a family of relations between nodes and, for any k, write $\widetilde{\mathrm{R}}^k$ to denote the composition $\sim^k \circ \mathrm{R}^k \circ \sim^k$. The set $\{\mathrm{R}^k\}$ is a *time-bisimulation up to* $\sim$ if, for any $k \geqslant 1$, any $P, P', Q, Q'' \in \overline{\mathbf{TCSP}}$, and any $\mu \in \Sigma^{\checkmark} \cup \{\tau\}$, the following four conditions are met:

[1]In fact, it is possible to show that, as with indexed bisimulations, each $\sim^k$ is a congruence with respect to *all* of the operators of Timed CSP, but we will not require this more complicated result here. A proof is given in [Oua00].

$$(P \,\mathrm{R}^k\, Q \wedge P \xrightarrow{tock} P') \Rightarrow \exists\, Q' \centerdot Q \xrightarrow{tock} Q' \wedge P' \,\widetilde{\mathrm{R}}^{k-1}\, Q'$$
$$(P \,\mathrm{R}^k\, Q \wedge P \xrightarrow{\mu} P') \Rightarrow \exists\, Q' \centerdot Q \xrightarrow{\mu} Q' \wedge P' \,\widetilde{\mathrm{R}}^{k}\, Q'$$
$$(P \,\mathrm{R}^k\, Q \wedge Q \xrightarrow{tock} Q'') \Rightarrow \exists\, P'' \centerdot P \xrightarrow{tock} P'' \wedge P'' \,\widetilde{\mathrm{R}}^{k-1}\, Q''$$
$$(P \,\mathrm{R}^k\, Q \wedge Q \xrightarrow{\mu} Q'') \Rightarrow \exists\, P'' \centerdot P \xrightarrow{\mu} P'' \wedge P'' \,\widetilde{\mathrm{R}}^{k}\, Q''.$$

Lemma A.14 *If* $\{\mathrm{R}^k \mid k \in \mathbb{N}\}$ *is a time-bisimulation up to* $\sim$*, then, for each* $k \geqslant 0$*,* $\mathrm{R}^k \subseteq \sim^k$*.*

Proof (Sketch.) One first shows that $\{\widetilde{\mathrm{R}}^k\}$ is a full-fledged time-bisimulation. This is a straightforward diagram chase requiring only the transitivity of each $\sim^k$ (as per Lemma A.12).

Next, one notices that, for any k, $\mathrm{R}^k \subseteq \widetilde{\mathrm{R}}^k$, since each $\sim^k$ is reflexive (Lemma A.12 again).

The result then follows by combining both these observations and invoking the maximality of $\{\sim^k\}$. ■

If η and η' are syntactic bindings and $k \geqslant 0$, we write $\eta \sim^k \eta'$ if, for all $X \in \mathit{VAR}$, $\eta(X) \sim^k \eta'(X)$. We then have:

Lemma A.15 *Let* $P \in \mathbf{TCSP}$ *and* $\eta \sim^k \eta'$ *for some syntactic bindings* η*,* η'*, and some fixed* $k \geqslant 0$*. Then* $P\eta \sim^k P\eta'$*.*

Proof (Sketch.) Define a family of relations $\{\approx^n\}$ as follows. For $n \leqslant k$, let $\approx^n = \approx^k = \{(R\eta, R\eta') \mid R \in \mathbf{TCSP}\}$. For greater values of n, let $\approx^n = \emptyset$.

We need only show that $\{\approx^n\}$ is a time-bisimulation up to $\sim$. The result will follow by taking $R = P$ and invoking Lemma A.14.

We proceed by structural induction on R. Let us tackle the cases of *tock*-transitions on external choice, along with recursion, in illustration; all other cases are at least as easy.

Without loss of generality, we pick $n = k \geqslant 1$ and thus consider the relation $\approx^k$. Let $R = R_1 \,\Box\, R_2$. Our structural induction hypothesis states that $(R_1\eta, R_1\eta')$ and $(R_2\eta, R_2\eta')$ both fulfill the four conditions for $\{\approx^k\}$ to be a time-bisimulation up to $\sim$. We must show the same holds for $(R\eta, R\eta')$. As stated above, we shall limit ourselves to (half of) the case of *tock*-transitions.

So suppose that $R\eta \xrightarrow{tock} Q$. Consulting the relevant inference rules, we find that we must have $R_1\eta \xrightarrow{tock} Q_1$, $R_2\eta \xrightarrow{tock} Q_2$, and $Q \equiv Q_1 \Box Q_2$.

By the induction hypothesis, we then conclude that $R_1\eta' \xrightarrow{tock} Q_1'$, with $Q_1 \mathrel{\tilde{\approx}}^{k-1} Q_1'$, i.e., $Q_1 \sim^{k-1} S_1 \approx^{k-1} S_1' \sim^{k-1} Q_1'$.

Since $S_1 \approx^{k-1} S_1'$, there is by definition some term E_1 such that $S_1 \equiv E_1\eta$ and $S_1' \equiv E_1\eta'$.

Likewise, we find $R_2\eta' \xrightarrow{tock} Q_2'$, with $Q_2 \sim^{k-1} S_2 \equiv E_2\eta \approx^{k-1} E_2\eta' \equiv S_2' \sim^{k-1} Q_2'$.

Let $Q' \equiv Q_1' \Box Q_2'$. We clearly have $R\eta' \xrightarrow{tock} Q'$. Moreover, since $\sim^{k-1}$ preserves the external choice operator (as per Lemma A.13), we have $(Q_1 \Box Q_2) \sim^{k-1} (S_1 \Box S_2)$, and likewise for the primed expression.

Stringing everything together, we get

$$\begin{gathered} Q \equiv (Q_1 \Box Q_2) \sim^{k-1} (S_1 \Box S_2) \equiv (E_1\eta \Box E_2\eta) \equiv (E_1 \Box E_2)\eta \\ \approx^{k-1} \\ (E_1 \Box E_2)\eta' \equiv (E_1\eta' \Box E_2\eta') \equiv (S_1' \Box S_2') \sim^{k-1} (Q_1' \Box Q_2') \equiv Q' \end{gathered}$$

i.e., $Q \mathrel{\tilde{\approx}}^{k-1} Q'$, as required.

We now consider the case of recursion. Note that $(\mu X \bullet R)\eta \xrightarrow{\tau} (R[\mu X \bullet R/X])\eta$ and likewise $(\mu X \bullet R)\eta' \xrightarrow{\tau} (R[\mu X \bullet R/X])\eta'$ (and these are the only allowable behaviours). But $(R[\mu X \bullet R/X])\eta \approx^k (R[\mu X \bullet R/X])\eta'$ by definition, so we are done. ∎

Lemma A.16 *If P is a* **TCSP** *term which is time-guarded for X, and η is a syntactic binding, then for any programs Q and Q', and any $k \geqslant 0$, if $Q \sim^k Q'$, then $P[Q/X]\eta \sim^{k+1} P[Q'/X]\eta$.*

Proof This is a done by structural induction on the definition of time-guardedness. All cases are straightforward save for the last one, which we tackle here.

We must show that the result holds for $P \equiv P_1 \mathbin{;} P_2$ if it holds for P_1 and P_2. Here P_1 is time-guarded for X and time-active, whereas no assumptions are made about P_2.

We are assuming that $Q \sim^k Q'$. From the induction hypothesis, we get that $P_1[Q/X]\eta \sim^{k+1} P_1[Q'/X]\eta$. Moreover, since P_1 is time-active, so are

$P_1[Q/X]\eta$ and $P_1[Q'/X]\eta$. (This is easily shown by structural induction on time-activity.)

By Lemma A.15, we also have that $P_2[Q/X]\eta \equiv P_2\eta[X := Q] \sim^k P_2\eta[X := Q'] \equiv P_2[Q'/X]\eta$, since it is assumed that $Q \sim^k Q'$.

Let us define a function $g : \overline{\mathbf{TCSP}} \longrightarrow \mathbb{N} \cup \{\infty\}$ to give us, for any program R, the smallest number of *tock*s that R must communicate before it is potentially able to terminate successfully:

$$g(R) = \inf\{\sharp(tr \upharpoonright tock) \mid R \stackrel{tr}{\Longrightarrow} \wedge \langle\checkmark\rangle \text{ in } tr\}.$$

We remark that, since $P_1[Q/X]\eta$ is time-active, $g(P_1[Q/X]\eta) \geqslant 1$. (This can be shown via a simple induction on time-activity.)

We now define a family of relations $\{\approx^n \mid n \in \mathbb{N}\}$ between programs, as follows: for any n, let

$$\begin{aligned}\approx^n \ = \ & \sim^n \cup \{((R_1 \mathbin{;} S_1), (R_2 \mathbin{;} S_2)) \mid \exists\, i, j \geqslant 0 \bullet n = i + j \wedge \\ & S_1 \sim^i S_2 \wedge R_1 \sim^{i+j} R_2 \wedge g(R_1) \geqslant j\}.\end{aligned}$$

From our earlier observations, we immediately have that $P[Q/X]\eta \equiv (P_1[Q/X]\eta \mathbin{;} P_2[Q/X]\eta) \approx^{k+1} (P_1[Q'/X]\eta \mathbin{;} P_2[Q'/X]\eta) \equiv P[Q'/X]\eta$. The result follows by observing that $\{\approx^n\}$ is easily verified to be a time-bisimulation. ■

Let $P \in \mathbf{TCSP}$ be time-guarded for X, and let η be a syntactic binding. We define a function $F = F_{P,X,\eta} : \mathbf{TCSP} \longrightarrow \overline{\mathbf{TCSP}}$, as follows: $F(Q) \mathrel{\widehat{=}} (SKIP \mathbin{;} P)[Q/X]\eta$. We have:

Lemma A.17 *Let $P \in \mathbf{TCSP}$ be time-guarded for X, and let η be a syntactic binding. For any $k \geqslant 0$, $F^k(STOP) \sim^k (\mu\, X \bullet P)\eta$, where $F = F_{P,X,\eta}$ is as above.*

Proof We proceed by induction on k. The base case $k = 0$ is trivial since, according to Lemma A.12, all programs are $\sim^0$-related. Assume that the result holds for k. We have the following sequence of deductions, select justifications of which follow.

$$\begin{aligned}
F^{k+1}(STOP) &\equiv F(F^k(STOP)) \\
&\equiv (SKIP\ ;\ P)[F^k(STOP)/X]\eta \\
&\equiv SKIP\ ;\ (P[F^k(STOP)/X]\eta) \\
&\sim^{k+1} SKIP\ ;\ (P[(\mu X \centerdot P)\eta/X]\eta) && (1) \\
&\sim^{k+1} (\mu X \centerdot P)\eta. && (2)
\end{aligned}$$

This concludes the induction step. (1) follows from Lemma A.16, using the fact that $F^k(STOP) \sim^k (\mu X \centerdot P)\eta$ by the induction hypothesis, and the fact that left sequential composition with *SKIP* preserves time-bisimulation (thanks to Lemma A.13). (2) rests on the observation that $SKIP\ ;\ (P[(\mu X \centerdot P)\eta/X]\eta) \sim^n (\mu X \centerdot P)\eta$ for *any* $n \geqslant 0$, as an immediate calculation of the possible executions will attest. ■

Lemma A.18 *Let $P, P' \in \overline{\mathbf{TCSP}}$, $k \geqslant 1$, and suppose that $P \sim^k P'$. Then for any τ-trace tr with $\sharp(tr \upharpoonright tock) < k$ and such that $P \overset{tr}{\Longrightarrow}$, we have $P' \overset{tr}{\Longrightarrow}$.*

Proof The τ-trace tr must originate from some execution $e \in \mathsf{exec}_R(P)$. Since $P \sim^k P'$, this execution can be matched for a duration of k *tock*s by some execution $e' \in \mathsf{exec}_R(P')$. It is plain that e' also gives rise to tr. ■

We are now in a position to give the proof of Proposition A.11.

Proof (Sketch.) We proceed by structural induction on terms. Specifically, let us call a program *finitely variable* if it satisfies the conclusion of Proposition A.11. We show by induction on P that whenever η is a syntactic binding such that $\eta(X)$ is finitely variable for all $X \in VAR$, then $P\eta$ is finitely variable. Here $P \in \mathbf{TCSP}$, and in particular is well-timed.

All cases of the structural induction are straightforward save for recursion, which we handle below.

We are assuming that $P\eta$ is finitely variable whenever η is pointwise finitely variable, and aim to show that $(\mu X \centerdot P)\eta$ is also finitely variable whenever η is pointwise finitely variable. Note that our grammar requires P to be time-guarded for X.

Let $k \geqslant 0$ be given, and let η be fixed. By Lemma A.16, $(\mu X \centerdot P)\eta \sim^{k+1} F^{k+1}(STOP)$, where $F = F_{P,X,\eta}$ is the function satisfying $F(Q) = (SKIP\ ;\ P)[Q/X]\eta$ defined earlier.

We first show that $F^n(STOP)$ is finitely variable for all $n \geqslant 0$. This is a simple induction on n which has a trivial base case $n = 0$. For the induction step, assuming the statement holds for n, we have

$$\begin{aligned} F^{n+1}(STOP) &\equiv F(F^n(STOP)) \\ &\equiv (SKIP \,;\, P)[F^n(STOP)/X]\eta \\ &\equiv SKIP \,;\, (P(\eta[X := F^n(STOP)])). \end{aligned}$$

Now $P(\eta[X := F^n(STOP)])$ is finitely variable by the structural induction hypothesis on P, our assumptions on η, and the induction hypothesis on n. Therefore $SKIP \,;\, (P(\eta[X := F^n(STOP)])) \equiv F^{n+1}(STOP)$ are finitely variable as well, which completes the induction on n.

Let us now consider any τ-trace tr with $\sharp(tr \upharpoonright tock) \leqslant k$ and such that $(\mu X \bullet P)\eta \overset{tr}{\Longrightarrow}$. Since $(\mu X \bullet P)\eta \sim^{k+1} F^{k+1}(STOP)$, we can invoke Lemma A.18 to conclude that $F^{k+1}(STOP) \overset{tr}{\Longrightarrow}$. But since $F^{k+1}(STOP)$ is finitely variable, we must have $\sharp tr \leqslant n$, for some uniform $n = n(k)$. Since tr was an arbitrary τ-trace of $(\mu X \bullet P)\eta$ containing k *tock*s or fewer, we conclude that $(\mu X \bullet P)\eta$ is indeed finitely variable, as required. ■

Theorem A.19 (5.14) *For any* $\overline{\mathbf{TCSP}}$ *program* P*, we have*

$$\Phi_R(P) = \mathcal{R}[\![P]\!].$$

Proof We first need some preliminaries. Define a *syntactic binding* to be a function $\eta : VAR \longrightarrow \overline{\mathbf{TCSP}}$. Syntactic bindings are the operational analogue of semantic bindings.

For $P \in \mathbf{TCSP}$, the application $P\eta \in \overline{\mathbf{TCSP}}$ is obtained via substitution of programs for free variables in the obvious way.

Given a syntactic binding η, define the associated semantic binding $\overline{\eta}$ as follows: for any $X \in VAR$, $\overline{\eta}(X) = \Phi_R(\eta(X))$. (See, however, the caveat below.)

The more general statement we would then like to prove is that, for any $P \in \mathbf{TCSP}$, and any syntactic binding η,

$$\Phi_R(P\eta) = \mathcal{R}[\![P]\!]\overline{\eta}. \qquad \text{(A.1)}$$

It is plain that this would establish the theorem, since programs have no free variables and therefore make bindings irrelevant.

The problem with this approach is that, in spite of our claim two paragraphs ago, there is no *a priori* guarantee that $\overline{\eta}$ is indeed a semantic binding, in the sense that its range be contained in $\mathcal{M}_R$, because we do not yet know that the function Φ_R's range (when considered over domain $\overline{\mathbf{TCSP}}$) is itself contained in $\mathcal{M}_R$. Therefore, we cannot be sure that the right-hand side $\mathcal{R}[\![P]\!]\overline{\eta}$ of Equation (A.1) is even well-defined!

There are two ways to circumvent this difficulty. The most obvious is to prove directly, from the operational semantics and the definition of Φ_R, that, for any $P \in \overline{\mathbf{TCSP}}$, $\Phi_R(P) \in \mathcal{M}_R$. This approach, which is implicitly taken in [Sch95] in the proof of that paper's congruence result, boils down to a somewhat tedious verification of the denotational axioms. Proposition 5.13 (the operational version of finite variability), for instance, readily implies Axiom *R7* (the denotational version of finite variability) for $\Phi_R(P)$; all the other axioms of $\mathcal{M}_R$ must likewise be shown to hold.

An alternative device can be employed, as follows. First, observe that every Timed CSP operator (apart from X and μX) can easily be extended to the whole of $\mathcal{P}(\mathit{TEST})$ (as opposed to $\mathcal{M}_R$). Consequently, semantic bindings can too be defined to assume values in $\mathcal{P}(\mathit{TEST})$ rather than being restricted to $\mathcal{M}_R$. Moreover, $\mathcal{P}(\mathit{TEST})$ is itself easily seen to be a complete ultrametric space under the 'same' metric as $\mathcal{M}_R$. **TCSP** terms again correspond to non-expanding functions, and terms that are time-guarded for X naturally correspond to contractions in X. (These facts are all readily verified by examining the relevant definitions and proofs, and noting that nowhere in the course of dealing with these statements was the fact that we were working in $\mathcal{M}_R$—rather than $\mathcal{P}(\mathit{TEST})$—used in any essential way.) It then follows that recursions have unique fixed points, which is the one crucial fact we will need to establish the theorem. Note that, since $\mathcal{M}_R$ is a subset of $\mathcal{P}(\mathit{TEST})$, the unique fixed points computed in both spaces must agree.

We now show that, for any term $P \in \mathbf{TCSP}$, and any syntactic binding η, Equation (A.1) is satisfied. We proceed by structural induction on P. Most instances are easy if occasionally tedious, and we therefore concentrate on four select cases.

case $P \underset{B}{\|} Q$**:** We must show that each of $\Phi_R((P \underset{B}{\|} Q)\eta)$ and $\mathcal{R}[\![P \underset{B}{\|} Q]\!]\overline{\eta}$ is a subset of the other. Since bindings play no real rôle here, let us for simplicity dispense with η and $\overline{\eta}$ by assuming that P and Q are themselves programs.

Let $u \in \Phi_R(P \underset{B}{\|} Q)$. Then there is some $e \in \mathsf{exec}(P \underset{B}{\|} Q)$ such that e **test** u. Write $e = R_0 \xrightarrow{z_1} R_1 \xrightarrow{z_2} \ldots \xrightarrow{z_n} R_n$. Each R_k must be of the form $P_i \underset{B}{\|} Q_j$, and each z_{k+1} must have been performed either by both of P_i and Q_j (precisely when $z_{k+1} \in B \cup \{tock\}$) or by exactly one of P_i and Q_j.

We can thus decompose the execution e into executions $e_P = P_0 \xrightarrow{x_1} P_1 \xrightarrow{x_2} \ldots \xrightarrow{x_l} P_l \in \mathsf{exec}(P)$ and $e_Q = Q_0 \xrightarrow{y_1} Q_1 \xrightarrow{y_2} \ldots \xrightarrow{y_m} Q_m \in \mathsf{exec}(Q)$. Write e_P **test** u_P and e_Q **test** u_Q, where u_P and u_Q are maximal such tests under the information ordering $\prec$; that is to say, the refusals in these tests are as large as possible.

We now claim that $u \in u_P \underset{B}{\parallel} u_Q$. This is a simple inspection of the definitions. Consider, for instance, $R_k \xrightarrow{z_{k+1}} R_{k+1}$, where $R_k = P_i \underset{B}{\parallel} Q_j$. If $z_{k+1} = \tau$, then this particular transition of e contributes nothing to u, and can be dismissed. Note that the corresponding x_{i+1} or y_{j+1} will itself also be a τ and contribute nothing to u_P or u_Q. Let us therefore first suppose that $z_{k+1} \in B \cup \{tock\}$. If D is a refusal such that R_k **test** $\langle D \rangle$, it easily follows from the fact that $R_k \equiv P_i \underset{B}{\parallel} Q_j$ that there are (maximal) refusals A and C such that P_i **test** $\langle A \rangle$ and Q_j **test** $\langle C \rangle$ and $\langle D \rangle \in \langle A \rangle \underset{B}{\parallel} \langle C \rangle$. And since $z_{k+1} \in B \cup \{tock\}$, we must have $x_{i+1} = y_{j+1} = z_{k+1}$ and $R_{k+1} \equiv P_{i+1} \underset{B}{\parallel} Q_{j+1}$. We therefore see that the contribution $\langle D, z_{k+1} \rangle$ of e in u is accounted for by the contributions $\langle A, z_{k+1} \rangle$ and $\langle C, z_{k+1} \rangle$, respectively of e_P and e_Q to u_P and u_Q.

A similar argument handles the case $z_{k+1} \notin B \cup \{tock, \tau\}$. We conclude that indeed $u \in u_P \underset{B}{\parallel} u_Q$. Since $e_P \in \mathsf{exec}(P)$ and e_P **test** u_P, we have, by definition, that $u_P \in \Phi_R(P)$. We can then invoke the induction hypothesis to conclude that $u_P \in \mathcal{R}[\![P]\!]$. Likewise, $u_Q \in \mathcal{R}[\![Q]\!]$. Naturally, it follows by definition that $u \in \mathcal{R}[\![P \underset{B}{\parallel} Q]\!]$, as required.

To establish the reverse containment, one starts with $u \in \mathcal{R}[\![P \underset{B}{\parallel} Q]\!]$ and then essentially retraces one's steps back into the statement that $u \in \Phi_R(P \underset{B}{\parallel} Q)$. This simple procedure requires little else than careful bookkeeping, and is left to the reader.

case $P \setminus A$**:** Here again let us disregard syntactic and semantic bindings which are at any rate of little importance. Let $v \in \Phi_R(P \setminus A)$. We must then have $e = P_0 \setminus A \xrightarrow{x_1} P_1 \setminus A \xrightarrow{x_2} \ldots \xrightarrow{x_n} P_n \setminus A \in \mathsf{exec}(P \setminus A)$ such that e **test** v. It is an easy matter to show that the execution e 'originates' from some execution $e' = P_0 \xrightarrow{y_1} P_1 \xrightarrow{y_2}$

$\ldots \xrightarrow{y_n} P_n \in \mathsf{exec}(P)$, where $x_i = y_i$ if $y_i \notin A$, and otherwise $x_i = \tau$, for all $1 \leqslant i \leqslant n$. Note, in addition, that, thanks to Rule 5.28 and Proposition 5.10, whenever $P_i \xrightarrow{tock} P_{i+1}$, it must have been the case that $\mathsf{init}^{\tau}(P_i) \cap (A \cup \{\tau\}) = \emptyset$.

Let u be a $\prec$-maximal test with the property that e' **test** u. Note that, by the property of e' quoted above, it must be the case that u is A-urgent. And since $u \in \Phi_R(P)$, we can invoke the induction hypothesis to conclude that $u \in \mathcal{R}[\![P]\!]$. It remains to show that $v \prec u \setminus A$. But this easily follows from the definition of hiding on tests together with the definition of the relation **test**.

The reverse containment is easily obtained by reversing the argument just given.

case X**:** We have that $\Phi_R(X\eta) = \Phi_R(\eta(X)) = \overline{\eta}(X) = \mathcal{R}[\![X]\!]\overline{\eta}$ as required.

case $\mu X \centerdot P$**:** We recall our induction hypothesis which states that, if ρ is *any* syntactic binding, then $\Phi_R(P\rho) = \mathcal{R}[\![P]\!]\overline{\rho}$. We now have

$$\begin{aligned}
\Phi_R((\mu X \centerdot P)\eta) &= \Phi_R(P[\mu X \centerdot P/X]\eta) && (1)\\
&= \Phi_R(P(\eta[X := (\mu X \centerdot P)\eta])) && (2)\\
&= \mathcal{R}[\![P]\!]\overline{\eta[X := (\mu X \centerdot P)\eta]} && (3)\\
&= \mathcal{R}[\![P]\!](\overline{\eta}[X := \Phi_R((\mu X \centerdot P)\eta)]). && (4)
\end{aligned}$$

To see (1), notice first that the only initial operational step allowed $(\mu X \centerdot P)\eta$ is a τ-transition into $P[\mu X \centerdot P/X]\eta$. Since the **test** relation discards τ-transitions from executions, (1) follows. (2) is an immediate consequence of the fact that $P[\mu X \centerdot P/X]\eta \equiv P(\eta[X := (\mu X \centerdot P)\eta])$. (3) is an application of the induction hypothesis, and (4) follows from the fact that $\overline{\eta[X := (\mu X \centerdot P)\eta]}(Y)$ is by definition equal to $\Phi_R(\eta[X := (\mu X \centerdot P)\eta](Y))$, which is $\Phi_R((\mu X \centerdot P)\eta)$ if $Y \equiv X$, and $\Phi_R(\eta(Y)) = \overline{\eta}(Y)$ otherwise.

This string of equalities implies that $\Phi_R((\mu X \centerdot P)\eta)$ is a fixed point of the function $\lambda x.\mathcal{R}[\![P]\!](\overline{\eta}[X := x]) : \mathcal{P}(\mathit{TEST}) \longrightarrow \mathcal{P}(\mathit{TEST})$. Since we know that $\mathcal{R}[\![\mu X \centerdot P]\!]\overline{\eta}$ is the *unique* fixed point of this function, we must conclude that $\Phi_R((\mu X \centerdot P)\eta) = \mathcal{R}[\![\mu X \centerdot P]\!]\overline{\eta}$ as required. ■

Appendix B

Power-Simulations and Model Checking

We define a *power-simulation* relation on the labelled transition systems of programs and show that it corresponds precisely to test refinement in $\mathcal{M}_R$. This provides us with a model checking algorithm for deciding refinement of processes with finite LTS's. The ideas presented here are, in much less sophisticated form, similar to those upon which FDR refinement checks are based—more on this topic can be found in [Ros97].

Let $P \in \overline{NODE}_R$ be fixed. For $u \in TEST$, let $P/\hat{u}$ represent the set of nodes reachable from P upon observing the test $\hat{u}^\frown\langle\bullet\rangle$ (the last refusal of u being discarded). Formally:

$$P/\hat{u} \mathrel{\widehat{=}} \{Q \in NODE_R \mid \exists\, \hat{e}^\frown Q \in \mathsf{exec}_R(P) \,\centerdot\, \hat{e}^\frown Q \ \mathbf{test}\ \hat{u}^\frown\langle\bullet\rangle\}.$$

We use this to build the *power labelled transition system* $\mathsf{PLTS}_R(P)$ of P, as follows. Its set of *power nodes* is $\mathsf{PLTS}_R^N \mathrel{\widehat{=}} \{P/\hat{u} \mid u \in TEST\}$, with an oriented edge (A, a)—where $A \in REF$ and $a \in \Sigma_{tock}^{\checkmark}$—from $P/\hat{u}$ to $P/\hat{v}$ if $P/(\hat{u}^\frown\langle A, a, \bullet\rangle) = P/\hat{v}$; we represent this with the notation $P/\hat{u} \xrightarrow{A,a} P/\hat{v}$.

We remark that $\mathsf{PLTS}_R(P)$ is finite whenever $\mathsf{LTS}_R(P)$ is. Note moreover that $\mathsf{PLTS}_R(P)$ is always *deterministic*: no power node ever has two distinct identically labelled edges emerging from it.

Given a power node $p \in \mathsf{PLTS}_R^N(P)$, we let $\mathsf{refs}(p) \mathrel{\widehat{=}} \{A \in REF \mid \exists\, Q \in p \,\centerdot\, Q\ \mathbf{ref}\ A\}$.

Let $P, Q \in \overline{NODE}_R$. We say that a relation $\mathcal{R}$ on $\mathsf{PLTS}_R^N(P) \times \mathsf{PLTS}_R^N(Q)$ is a *power-simulation* if the following condition holds:

$$\begin{aligned}
&\forall q \in \mathsf{PLTS}^N_R(Q) \bullet \forall p \in \mathsf{PLTS}^N_R(P) \bullet p\,\mathcal{R}\,q \Rightarrow\\
&\quad(\mathsf{refs}(q) \subseteq \mathsf{refs}(p) \wedge\\
&\quad\ \forall (q \xrightarrow{e} q') \in \mathsf{PLTS}_R(Q)\bullet\\
&\qquad\quad \exists\, p' \bullet p \xrightarrow{e} p' \in \mathsf{PLTS}_R(P) \wedge p'\,\mathcal{R}\,q').
\end{aligned}$$

We now have:

Definition B.1 *For $P, Q \in \overline{NODE}_R$, we say that P* power-simulates *Q, written P psim_R Q, if there exists $\mathcal{R}$ a power-simulation on $\mathsf{PLTS}^N_R(P) \times \mathsf{PLTS}^N_R(Q)$ such that $P/\langle\rangle\ \mathcal{R}\ Q/\langle\rangle$.*

Finally, we have the following main result:

Theorem B.1 *For any $P, Q \in \overline{\mathbf{TCSP}}$,*

$$P\ \mathsf{psim}_R\ Q \Leftrightarrow P \sqsubseteq_R Q.$$

Proof (Sketch.) Follows easily from Theorem 5.14 and the definition of power-simulation. ■

Theorem B.1 yields an algorithm to decide the refinement $P \sqsubseteq_R Q$ whenever the labelled transition systems of P and Q are finite. Consider the finite set $\mathsf{PLTS}^N_R(P) \times \mathsf{PLTS}^N_R(Q)$, and let `Check` and `Pend` be two 'set-variables', both intended as subsets of $\mathsf{PLTS}^N_R(P) \times \mathsf{PLTS}^N_R(Q)$, with `Check` initially empty and `Pend` initially containing the single pair $(P/\langle\rangle, Q/\langle\rangle)$. Repeat the following steps until `Pend` is empty:

1. Pick a pair $(p, q) \in$ `Pend`.
2. Check whether $\mathsf{refs}(q) \subseteq \mathsf{refs}(p)$. If this is *not* the case, exit immediately with the answer $P \not\sqsubseteq_R Q$.
3. For each edge $q \xrightarrow{e} q'$ in $\mathsf{PLTS}_R(Q)$, check whether there is an edge $p \xrightarrow{e} p'$ in $\mathsf{PLTS}_R(P)$. If this is *not* the case, exit immediately with the answer $P \not\sqsubseteq_R Q$.
4. Add (p, q) to `Check`.
5. For each edge $q \xrightarrow{e} q'$ in $\mathsf{PLTS}_R(Q)$ and corresponding unique edge $p \xrightarrow{e} p'$ in $\mathsf{PLTS}_R(P)$, add (p', q') to `Pend` unless (p', q') is in `Check`.

It is not difficult to see that this algorithm will always terminate. If the loop is successfully exited (i.e., upon finding `Pend` empty), then `Check` is a power-simulation and thus $P \sqsubseteq_R Q$. On the other hand, if $P \not\sqsubseteq_R Q$, then the loop will necessarily be exited with this answer at some point. The easy verification of these claims is left to the reader.

We conclude by pointing out that refinement in $\mathcal{M}_{UR}$ can be decided, whenever the labelled transition systems of the processes involved are finite, by using similar definitions and a similar algorithm.

Appendix C

Operational Semantics for the Timed Failures-Stability Model

We give an operational semantics for Reed and Roscoe's timed failures-stability model for Timed CSP, denoted here $\mathcal{M}_{TFS}$, with associated semantic map $\mathcal{F}_{TS}[\![\cdot]\!]$. This model, presented in [RR87, Ree88, RR99], improves upon $\mathcal{M}_{TF}$ in that it associates a *stability value* to every timed failure, which value represents the least time by which a process, given its observed behaviour, can be guaranteed to have become *stable*. *Stability* here refers to invariance under the passage of time.

To capture this notion operationally, two different types of evolutions are introduced: *stable* evolutions (implying that the process has already reached stability), as well as *unstable* evolutions (implying that the process is headed towards some eventual internal transition, and is therefore unstable).

The style and conventions of this appendix are very similar to those of Section 3.2. We use as *nodes* the previously defined collection $\overline{NODE}_{TF}$ of Timed CSP programs with fractional delays and no well-timedness requirements. *Open nodes*, likewise, are elements of $NODE_{TF}$. a and b stand for (non-*tock*) visible events, i.e., belong to $\Sigma^{\checkmark}$. $A \subseteq \Sigma$ and $B \subseteq \Sigma^{\checkmark}$. μ can be a visible event or a silent one ($\mu \in \Sigma^{\checkmark} \cup \{\tau\}$). $P \xrightarrow{\mu} P'$ means P can perform an immediate and instantaneous μ*-transition*, and become P' (communicating μ in the process if μ is a visible event). $P \overset{\mu}{\not\longrightarrow}$ means that P cannot possibly do a μ at that particular time. $P \overset{t}{\leadsto}_1 P'$ implies that P is a stable node, and can become P' (also stable) simply by virtue of letting t units of time elapse, where t is a non-negative real number. $P \overset{t}{\leadsto}_2 P'$ states that the unstable node P can become P' by letting t time units elapse. When using the generic notation $\overset{t}{\rightsquigarrow}_x$, it is understood that x stands for either 1 or 2.

The transition rules are as follows:

$$\frac{}{\mathit{STOP} \overset{t}{\rightsquigarrow}_1 \mathit{STOP}} \tag{C.1}$$

$$\frac{}{\mathit{SKIP} \overset{t}{\rightsquigarrow}_1 \mathit{SKIP}} \tag{C.2}$$

$$\frac{}{\mathit{SKIP} \xrightarrow{\checkmark} \mathit{STOP}} \tag{C.3}$$

$$\frac{}{\mathit{WAIT}\ u \overset{t}{\rightsquigarrow}_2 \mathit{WAIT}\ (u-t)}\ [\, t \leqslant u \,] \tag{C.4}$$

$$\frac{}{\mathit{WAIT}\ 0 \xrightarrow{\tau} \mathit{SKIP}} \tag{C.5}$$

$$\frac{P_1 \overset{t}{\rightsquigarrow}_x P_1'}{P_1 \overset{u}{\rhd} P_2 \overset{t}{\rightsquigarrow}_2 P_1' \overset{u-t}{\rhd} P_2}\ [\, t \leqslant u \,] \tag{C.6}$$

$$\frac{}{P_1 \overset{0}{\rhd} P_2 \xrightarrow{\tau} P_2} \tag{C.7}$$

$$\frac{P_1 \xrightarrow{\tau} P_1'}{P_1 \overset{u}{\rhd} P_2 \xrightarrow{\tau} P_1' \overset{u}{\rhd} P_2} \tag{C.8}$$

$$\frac{P_1 \xrightarrow{a} P_1'}{P_1 \overset{u}{\rhd} P_2 \xrightarrow{a} P_1'} \tag{C.9}$$

$$\frac{}{(a \longrightarrow P) \overset{t}{\rightsquigarrow}_1 (a \longrightarrow P)} \tag{C.10}$$

$$\frac{}{(a \longrightarrow P) \xrightarrow{a} P} \tag{C.11}$$

$$\frac{}{(a : A \longrightarrow P(a)) \overset{t}{\rightsquigarrow}_1 (a : A \longrightarrow P(a))} \tag{C.12}$$

$$\frac{}{(a : A \longrightarrow P(a)) \xrightarrow{b} P(b)} \ [\, b \in A \,] \tag{C.13}$$

$$\frac{P_1 \overset{t}{\rightsquigarrow}_x P_1' \quad P_2 \overset{t}{\rightsquigarrow}_y P_2'}{P_1 \mathrel{\Box} P_2 \overset{t}{\rightsquigarrow}_{\max\{x,y\}} P_1' \mathrel{\Box} P_2'} \tag{C.14}$$

$$\frac{P_1 \xrightarrow{\tau} P_1'}{P_1 \mathrel{\Box} P_2 \xrightarrow{\tau} P_1' \mathrel{\Box} P_2} \qquad \frac{P_2 \xrightarrow{\tau} P_2'}{P_1 \mathrel{\Box} P_2 \xrightarrow{\tau} P_1 \mathrel{\Box} P_2'} \tag{C.15}$$

$$\frac{P_1 \xrightarrow{a} P_1'}{P_1 \mathrel{\Box} P_2 \xrightarrow{a} P_1'} \qquad \frac{P_2 \xrightarrow{a} P_2'}{P_1 \mathrel{\Box} P_2 \xrightarrow{a} P_2'} \tag{C.16}$$

$$\frac{}{P_1 \sqcap P_2 \xrightarrow{\tau} P_1} \qquad \frac{}{P_1 \sqcap P_2 \xrightarrow{\tau} P_2} \tag{C.17}$$

$$\frac{P_1 \overset{t}{\rightsquigarrow}_x P_1' \quad P_2 \overset{t}{\rightsquigarrow}_y P_2'}{P_1 \mathbin{\underset{B}{\|}} P_2 \overset{t}{\rightsquigarrow}_{\max\{x,y\}} P_1' \mathbin{\underset{B}{\|}} P_2'} \tag{C.18}$$

$$\frac{P_1 \xrightarrow{\mu} P_1'}{P_1 \mathbin{\underset{B}{\|}} P_2 \xrightarrow{\mu} P_1' \mathbin{\underset{B}{\|}} P_2} \ [\, \mu \notin B, \mu \neq \checkmark \,] \tag{C.19a}$$

$$\frac{P_2 \xrightarrow{\mu} P_2'}{P_1 \underset{B}{\parallel} P_2 \xrightarrow{\mu} P_1 \underset{B}{\parallel} P_2'} \; [\, \mu \notin B, \mu \neq \checkmark \,] \tag{C.19b}$$

$$\frac{P_1 \xrightarrow{a} P_1' \quad P_2 \xrightarrow{a} P_2'}{P_1 \underset{B}{\parallel} P_2 \xrightarrow{a} P_1' \underset{B}{\parallel} P_2'} \; [\, a \in B \,] \tag{C.20}$$

$$\frac{P_1 \xrightarrow{\checkmark} P_1'}{P_1 \underset{B}{\parallel} P_2 \xrightarrow{\checkmark} P_1'} \; [\, \checkmark \notin B \,] \qquad \frac{P_2 \xrightarrow{\checkmark} P_2'}{P_1 \underset{B}{\parallel} P_2 \xrightarrow{\checkmark} P_2'} \; [\, \checkmark \notin B \,] \tag{C.21}$$

$$\frac{P_1 \overset{t}{\rightsquigarrow}_x P_1' \quad P_2 \overset{t}{\rightsquigarrow}_y P_2'}{P_1 \interleave P_2 \overset{t}{\rightsquigarrow}_{\max\{x,y\}} P_1' \interleave P_2'} \tag{C.22}$$

$$\frac{P_1 \xrightarrow{\mu} P_1'}{P_1 \interleave P_2 \xrightarrow{\mu} P_1' \interleave P_2} \; [\, \mu \neq \checkmark \,] \tag{C.23a}$$

$$\frac{P_2 \xrightarrow{\mu} P_2'}{P_1 \interleave P_2 \xrightarrow{\mu} P_1 \interleave P_2'} \; [\, \mu \neq \checkmark \,] \tag{C.23b}$$

$$\frac{P_1 \xrightarrow{\checkmark} P_1'}{P_1 \interleave P_2 \xrightarrow{\checkmark} P_1'} \qquad \frac{P_2 \xrightarrow{\checkmark} P_2'}{P_1 \interleave P_2 \xrightarrow{\checkmark} P_2'} \tag{C.24}$$

$$\frac{P_1 \overset{t}{\rightsquigarrow}_x P_1' \quad P_1 \stackrel{\checkmark}{\not\longrightarrow}}{P_1 \,;\, P_2 \overset{t}{\rightsquigarrow}_x P_1' \,;\, P_2} \tag{C.25}$$

$$\frac{P_1 \xrightarrow{\checkmark} P_1'}{P_1 \,;\, P_2 \xrightarrow{\tau} P_2} \tag{C.26}$$

$$\frac{P_1 \xrightarrow{\mu} P_1'}{P_1 \,;\, P_2 \xrightarrow{\mu} P_1' \,;\, P_2} \; [\, \mu \neq \checkmark \,] \tag{C.27}$$

$$\frac{P \overset{t}{\rightsquigarrow}_x P' \quad \forall\, a \in A \centerdot P \overset{a}{\not\longrightarrow}}{P \setminus A \overset{t}{\rightsquigarrow}_x P' \setminus A} \tag{C.28}$$

$$\frac{P \xrightarrow{a} P'}{P \setminus A \xrightarrow{\tau} P' \setminus A}\ [\, a \in A\,] \tag{C.29}$$

$$\frac{P \xrightarrow{\mu} P'}{P \setminus A \xrightarrow{\mu} P' \setminus A}\ [\, \mu \notin A\,] \tag{C.30}$$

$$\frac{P \overset{t}{\rightsquigarrow}_x P'}{f^{-1}(P) \overset{t}{\rightsquigarrow}_x f^{-1}(P')} \tag{C.31}$$

$$\frac{P \xrightarrow{\mu} P'}{f^{-1}(P) \xrightarrow{\mu} f^{-1}(P')}\ [\, \mu \in \{\tau, \checkmark\}\,] \tag{C.32}$$

$$\frac{P \xrightarrow{f(a)} P'}{f^{-1}(P) \xrightarrow{a} f^{-1}(P')} \tag{C.33}$$

$$\frac{P \overset{t}{\rightsquigarrow}_x P'}{f(P) \overset{t}{\rightsquigarrow}_x f(P')} \tag{C.34}$$

$$\frac{P \xrightarrow{\mu} P'}{f(P) \xrightarrow{\mu} f(P')}\ [\, \mu \in \{\tau, \checkmark\}\,] \tag{C.35}$$

$$\frac{P \xrightarrow{a} P'}{f(P) \xrightarrow{f(a)} f(P')} \tag{C.36}$$

$$\frac{}{\mu\, X \centerdot P \xrightarrow{\tau} P[(\mu\, X \centerdot P)/X].} \tag{C.37}$$

The remark made in Section 3.2 concerning negative premisses (appearing in Rules C.25 and C.28) applies here as well.

We now present, without proof, a number of properties enjoyed by this operational semantics.

If P and Q are open nodes, we write $P \equiv Q$ to indicate that P and Q are syntactically identical.

If P is a closed node, we define $\mathsf{init}_{TFS}(P)$ to be the set of visible events that P can immediately perform: $\mathsf{init}_{TFS}(P) \mathrel{\widehat{=}} \{a \in \Sigma^{\checkmark} \mid P \xrightarrow{a}\}$.

For P a closed node, we define an *execution* of P to be a sequence $e = P_0 \overset{z_1}{\longmapsto} P_1 \overset{z_2}{\longmapsto} \ldots \overset{z_n}{\longmapsto} P_n$ (with $n \geqslant 0$), where $P_0 \equiv P$, the P_i's are nodes, and each subsequence $P_i \overset{z_{i+1}}{\longmapsto} P_{i+1}$ in e is either a transition $P_i \xrightarrow{\mu} P_{i+1}$ (with $z_{i+1} = \mu$), or an evolution $P_i \overset{t}{\leadsto}_x P_{i+1}$ (with $z_{i+1} = t$). In addition, every such transition or evolution must be validly allowed by the operational inference Rules C.1–C.37. The set of executions of P is written $\mathsf{exec}_{TFS}(P)$, or $\mathsf{exec}(P)$ for short. By convention, writing down a transition (or sequence thereof) such as $P \xrightarrow{a} P'$, is equivalent to stating that $P \xrightarrow{a} P' \in \mathsf{exec}(P)$; the same, naturally, goes for evolutions.

Every execution e gives rise to a *timed τ-trace* $\mathsf{abs}(e)$ in the obvious way, by removing nodes and evolutions from the execution, but recording events' time of occurrence in agreement with e's (stable and unstable) evolutions.

The *duration* $\mathsf{dur}(e)$ of an execution e is equal to the sum of its (stable and unstable) evolutions.

Proposition C.1 *A node cannot be both stable and unstable:*

$$(P \overset{t}{\leadsto}_x P' \wedge P \overset{t'}{\leadsto}_y P'') \Rightarrow x = y.$$

Proposition C.2 *A stable node is invariant under the passage of time, and can evolve up to any time:*

$$P \overset{t}{\leadsto}_1 P' \Leftrightarrow (P \equiv P' \wedge \forall\, t' \geqslant 0 \centerdot P \overset{t'}{\leadsto}_1 P).$$

Proposition C.3 *Time determinacy:*

$$(P \overset{t}{\leadsto}_x P' \wedge P \overset{t}{\leadsto}_x P'') \Rightarrow P' \equiv P''.$$

Proposition C.4 *Persistency—the set of possible initial visible events remains constant under evolution:*

$$P \overset{t}{\leadsto}_x P' \Rightarrow \mathsf{init}(P) = \mathsf{init}(P').$$

Proposition C.5 *Time continuity:*

$$P \overset{t+t'}{\rightsquigarrow}_x P' \Leftrightarrow \exists\, P'' \centerdot P \overset{t}{\rightsquigarrow}_x P'' \overset{t'}{\rightsquigarrow}_x P'.$$

Proposition C.6 *Maximal progress, or τ-urgency:*

$$P \xrightarrow{\tau} \Rightarrow \forall\, t > 0 \centerdot \nexists\, P' \centerdot P \overset{t}{\rightsquigarrow}_x P'.$$

Corollary C.7

$$(P \overset{t}{\rightsquigarrow}_x P' \xrightarrow{\tau} \wedge\, P \overset{t'}{\rightsquigarrow}_x P'' \xrightarrow{\tau}) \Rightarrow (t = t' \wedge x = 2).$$

Proposition C.8 *A node P can always evolve up to the time of the next τ action, or up to any time if no τ action lies ahead:*

$$\forall\, t \geqslant 0 \centerdot (\nexists\, P' \centerdot P \overset{t}{\rightsquigarrow}_x P') \Rightarrow (P \xrightarrow{\tau} \vee\, \exists\, t' < t, P'' \centerdot P \overset{t'}{\rightsquigarrow}_2 P'' \xrightarrow{\tau}).$$

Proposition C.9 *An unstable node cannot evolve forever:*

$$P \overset{t}{\rightsquigarrow}_2 P' \Rightarrow \exists\, t', P'' \centerdot P \overset{t'}{\rightsquigarrow}_2 P'' \xrightarrow{\tau}.$$

Proposition C.10 *Finite variability—a* program $P \in \overline{\mathbf{TCSP}}$ *cannot perform unboundedly many action in a finite amount of time:*

$$\forall\, t \geqslant 0 \centerdot \exists\, n = n(P, t) \in \mathbb{N} \centerdot \forall\, e \in \mathsf{exec}(P) \centerdot \mathsf{dur}(e) \leqslant t \Rightarrow \sharp\mathsf{abs}(e) \leqslant n.$$

Unsurprisingly, this operational semantics has tight connections with the other operational semantics presented in this thesis, and in particular with that associated with $\mathcal{M}_{TF}$:

Proposition C.11 *For any program $P \in \overline{\mathbf{TCSP}}$,*

$$\begin{aligned} P \xrightarrow{\mu} P' \in \mathsf{exec}_{TF} \quad &\Leftrightarrow \quad P \xrightarrow{\mu} P' \in \mathsf{exec}_{TFS} \\ P \overset{t}{\rightsquigarrow} P' \in \mathsf{exec}_{TF} \quad &\Leftrightarrow \quad \exists\, x \centerdot P \overset{t}{\rightsquigarrow}_x P' \in \mathsf{exec}_{TFS}. \end{aligned}$$

A set of visible events is *refused* by a node P if P cannot immediately perform a τ-transition and has no valid initial transition labelled with an event from that set. Thus for $A \subseteq \Sigma^{\checkmark}$, we write P **ref** A if $P \overset{\tau}{\not\longrightarrow} \wedge A \cap \mathsf{init}(P) = \emptyset$.

An execution e of a node P is said to *fail* a timed failure-stability triple $(s, \alpha, \aleph)$ (where $\alpha \in \mathbb{R}^+$ represents an approximation from below of the stability value of P on $(s, \aleph)$) if the timed trace s corresponds to the execution e, the nodes of e can always refuse the relevant parts of $\aleph$, and α is the least time following the completion of all transitions in e by which P is not observed to be unstable. We write this relation as $e\ \mathbf{fail}_S\ (s, \alpha, \aleph)$. It is defined inductively on e as follows:

$$\begin{aligned}
P\ \mathbf{fail}_S\ (\langle\rangle, 0, \emptyset) &\Leftrightarrow \mathbf{true}\\
(P \xrightarrow{\tau})^\frown e'\ \mathbf{fail}_S\ (s, \alpha, \aleph) &\Leftrightarrow e'\ \mathbf{fail}_S\ (s, \alpha, \aleph)\\
(P \xrightarrow{a})^\frown e'\ \mathbf{fail}_S\ (\langle(0,a)\rangle^\frown s', \alpha, \aleph) &\Leftrightarrow a \neq \tau \wedge e'\ \mathbf{fail}_S\ (s', \alpha, \aleph)\\
(P \overset{t}{\rightsquigarrow}_2)^\frown e'\ \mathbf{fail}_S\ (s, \alpha, \aleph) &\Leftrightarrow P\ \mathbf{ref}\ \sigma(\aleph \upharpoonright t) \wedge\\
&\qquad e'\ \mathbf{fail}_S\ (s - t, \alpha - t, \aleph - t)\\
(P \overset{t}{\rightsquigarrow}_1)^\frown e'\ \mathbf{fail}_S\ (\langle\rangle, 0, \aleph) &\Leftrightarrow \mathsf{abs}(e') = \langle\rangle \wedge P\ \mathbf{ref}\ \sigma(\aleph) \wedge\\
&\qquad \mathsf{end}(\aleph) \leqslant t + \mathsf{dur}(e')\\
(P \overset{t}{\rightsquigarrow}_1)^\frown e'\ \mathbf{fail}_S\ (s, \alpha, \aleph) &\Leftrightarrow \mathsf{abs}(e') \neq \langle\rangle \wedge P\ \mathbf{ref}\ \sigma(\aleph \upharpoonright t) \wedge\\
&\qquad e'\ \mathbf{fail}_S\ (s - t, \alpha - t, \aleph - t).
\end{aligned}$$

The $\underline{SUP}$ operator on sets of timed failure-stability triples is taken from [RR99]:

$$\underline{SUP}(S) \ \widehat{=}\ \{(s, \alpha, \aleph) \mid (s, \aleph) \in \mathsf{TFailures}(S) \wedge \alpha = \sup\{\beta \mid (s, \beta, \aleph) \in S\}\}$$

where $\mathsf{TFailures}((s, \alpha, \aleph)) \ \widehat{=}\ (s, \aleph)$, etc.

Finally, we define the function Φ_{TFS}, which extracts the timed failure-stability denotational representation of a program from its set of executions.

Definition C.1 *For $P \in \overline{\mathbf{TCSP}}$, we set*

$$\Phi_{TFS}(P) \ \widehat{=}\ \underline{SUP}\{(s, \alpha, \aleph) \mid \exists\, e \in \mathsf{exec}_{TFS}(P) \bullet e\ \mathbf{fail}_S\ (s, \alpha, \aleph)\}.$$

We can now state the chief congruence result:

Theorem C.12 *For any $\overline{\mathbf{TCSP}}$ program P, we have*

$$\Phi_{TFS}(P) = \mathcal{F}_{TS}[\![P]\!].$$

Bibliography

[ACD90] R. Alur, C. Courcoubetis, and D. Dill. Model-checking for real-time systems. In *Proceedings of the Fifth Annual Symposium on Logic in Computer Science (LICS 90)*, pages 414–425. IEEE Computer Society Press, 1990.

[ACD93] R. Alur, C. Courcoubetis, and D. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):2–34, 1993.

[ACH94] R. Alur, C. Courcoubetis, and T. Henzinger. The observational power of clocks. In *Proceedings of the Fifth International Conference on Concurrency Theory (CONCUR 94)*, volume 836, pages 162–177. Springer LNCS, 1994.

[AD94] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[AH93] R. Alur and T. A. Henzinger. Real-time logics: Complexity and expressiveness. *Information and Computation*, 104(1):35–77, 1993.

[AH94] R. Alur and T. A. Henzinger. A really temporal logic. *Journal of the ACM*, 41(1):181–204, 1994.

[AHR98] R. Alur, T. A. Henzinger, and S. K. Rajamani. Symbolic exploration of transition hierarchies. In *Proceedings of the Fourth Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 98)*, volume 1384, pages 330–344. Springer LNCS, 1998.

[AK96] R. Alur and R. P. Kurshan. Timing analysis in COSPAN. In *Proceedings of Hybrid Systems III*, volume 1066, pages 220–231. Springer LNCS, 1996.

[BB91] J. C. M. Baeten and J. A. Bergstra. Real time process algebra. *Formal Aspects of Computing*, 3:142–188, 1991.

[BCM+92] J. Burch, E. Clarke, K. McMillan, D. Dill, and L. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, 1992.

[BLL+96] J. Bengtsson, K. G. Larsen, F. Larsen, P. Pettersson, and W. Yi. UppAal: A tool-suite for automatic verification of real-time systems. In *Proceedings of Hybrid Systems III*, volume 1066, pages 232–243. Springer LNCS, 1996.

[BLP+99] G. Behrmann, K. Larsen, J. Pearson, C. Weise, and W. Yi. Efficient timed reachability analysis using clock difference diagrams. In *Proceedings of the Eleventh International Conference on Computer-Aided Verification (CAV 99)*, volume 1633. Springer LNCS, 1999.

[BSV95] F. Balarin and A. L. Sangiovanni-Vincentelli. An iterative approach to verification of real-time systems. *Formal Methods in System Design*, 6:67–95, 1995.

[CG95] R. Chapman and M. Goldsmith. Translating timer automata to TCSP. Formal Systems Design and Development, Inc., 1995.

[Che92] L. Chen. *Timed Processes: Models, Axioms and Decidability.* PhD thesis, University of Edinburgh, 1992.

[CLM97] R. Cleaveland, G. Lüttgen, and M. Mendler. An algebraic theory of multiple clocks. In *Proceedings of the Eighth International Conference on Concurrency Theory (CONCUR 97)*, volume 1243, pages 166–180. Springer LNCS, 1997.

[Dav91] J. Davies. *Specification and Proof in Real-Time Systems.* PhD thesis, Oxford University, 1991.

[DOTY96] C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool Kronos. In *Proceedings of Hybrid Systems III*, volume 1066, pages 208–219. Springer LNCS, 1996.

[DS95] J. Davies and S. A. Schneider. A brief history of Timed CSP. *Theoretical Computer Science*, 138(2):243–271, 1995.

[Fid93] C. J. Fidge. A formal definition of priority in CSP. *ACM Transactions on Programming Languages and Systems*, 15(4):681–705, 1993.

[GB87] R. Gerth and A. Boucher. A timed failures model for extended communicating processes. In *Proceedings of the Fourteenth International Colloquium on Automata, Languages and Programming (ICALP 87)*, volume 267, pages 95–114. Springer LNCS, 1987.

[Hen96] T. A. Henzinger. The theory of hybrid automata. In *Proceedings of the Eleventh Annual Symposium on Logic in Computer Science (LICS 96)*. IEEE Computer Society Press, 1996.

[Hen98] T. A. Henzinger. It's about time: Real-time logics reviewed. In *Proceedings of the Ninth International Conference on Concurrency Theory (CONCUR 98)*, volume 1466, pages 439–454. Springer LNCS, 1998.

[HHWT97] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: A model checker for hybrid systems. In *Proceedings of the Ninth International Conference on Computer-Aided Verification (CAV 97)*, volume 1254, pages 460–463. Springer LNCS, 1997.

[HJL93] C. L. Heitmeyer, R. D. Jeffords, and B. G. Labaw. A benchmark for comparing different approaches for specifying and verifying real-time systems. In *Proceedings of the Tenth International Workshop on Real-Time Operating Systems and Software*, 1993.

[HK97] T. A. Henzinger and O. Kupferman. From quantity to quality. In *Proceedings of the First International Workshop on Hybrid and Real-time Systems (HART 97)*, volume 1201, pages 48–62. Springer LNCS, 1997.

[HKV96] T. A. Henzinger, O. Kupferman, and M. Y. Vardi. A space-efficient on-the-fly algorithm for real-time model checking. In *Proceedings of the Seventh International Conference on Concurrency Theory (CONCUR 96)*, volume 1119, pages 514–529. Springer LNCS, 1996.

[HMP90] T. A. Henzinger, Z. Manna, and A. Pnueli. Temporal proof methodologies for real-time systems. In *Proceedings of the Eighteenth Annual Symposium on Principles of Programming Languages (POPL 90)*, pages 353–366. ACM Press, 1990.

[HMP92] T. A. Henzinger, Z. Manna, and A. Pnueli. What good are digital clocks? In *Proceedings of the Nineteenth International Colloquium on Automata, Languages, and Programming (ICALP 92)*, volume 623, pages 545–558. Springer LNCS, 1992.

[HNSY94] T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice-Hall International, London, 1985.

[HR91] M. Hennessy and T. Regan. A temporal process algebra. In *Proceedings of the Third International Conference on Formal Description Techniques for Distributed Systems and Communications Protocols (FORTE 90)*, pages 33–48. North-Holland, 1991.

[Jac92] D. M. Jackson. *Logical Verification of Reactive Software Systems.* PhD thesis, Oxford University, 1992.

[Jef91a] A. Jeffrey. Abstract timed observation and process algebra. In *Proceedings of the Second International Conference on Concurrency Theory (CONCUR 91)*, volume 527, pages 332–345. Springer LNCS, 1991.

[Jef91b] A. Jeffrey. Discrete timed CSP. Programming Methodology Group Memo 78, Department of Computer Sciences, Chalmers University, 1991.

[KR91] A. Kay and J. N. Reed. A rely and guarantee method for Timed CSP. *IEEE Transactions for Software Engineering*, 1991.

[KR96] R.W. Knight and G. M. Reed. Minimal clock numbers for untimed bisimulation congruence. Unpublished, 1996.

[Lan90] R. Langerak. A testing theory for LOTOS using deadlock detection. In *Proceedings of the Tenth International Symposium on Protocol Specification, Testing and Verification (PSTV 90).* Elsevier, 1990.

[Laz99] R. S. Lazić. *A semantic study of data-independence with applications to Model Checking.* PhD thesis, Oxford University, 1999.

[LGS$^+$95] C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani, and S. Bensalem. Property preserving abstractions for the verification of concurrent systems. *Formal Methods in System Design*, 6:11–44, 1995.

[LL95] F. Laroussinie and K. G. Larsen. Compositional model checking of real time systems. In *Proceedings of the Sixth International Conference on Concurrency Theory (CONCUR 95)*, volume 962, pages 27–41. Springer LNCS, 1995.

[Low95] G. Lowe. Probabilistic and prioritized models of Timed CSP. *Theoretical Computer Science*, 138(2):315–352, 1995.

[Mar74] J. E. Marsden. *Elementary Classical Analysis.* W. H. Freeman and Company, New York, 1974.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice-Hall International, London, 1989.

[MRS95] M. W. Mislove, A. W. Roscoe, and S. A. Schneider. Fixed points without completeness. *Theoretical Computer Science*, 138(2):273–314, 1995.

[MT90] F. Moller and C. Tofts. A temporal calculus of communicating systems. In *Proceedings of the First International Conference on Concurrency Theory (CONCUR 90)*, volume 458, pages 401–415. Springer LNCS, 1990.

[Muk93] A. Mukkaram. *A Refusal Testing Model for CSP.* PhD thesis, Oxford University, 1993.

[NOSY93] X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. An approach to the description and analysis of hybrid systems. In *Hybrid Systems (R. L. Grossman, A. Nerode, A. P. Raun, and H. Rischel, Eds.)*, volume 736, pages 149–178. Springer LNCS, 1993.

[NS91] X. Nicollin and J. Sifakis. An overview and synthesis on timed process algebras. In *Proceedings of the Third International Conference on Computer-Aided Verification (CAV 91)*, volume 575, pages 376–397. Springer LNCS, 1991.

[NS94] X. Nicollin and J. Sifakis. The algebra of timed processes, ATP: Theory and application. *Information and Computation*, 114:131–178, 1994.

[OR99] J. Ouaknine and G. M. Reed. Model-checking temporal behaviour in CSP. In *Proceedings of the 1999 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 99)*. CSREA Press, 1999.

[Oua97] J. Ouaknine. Connections between CSP and Timed CSP. Dissertation for Transfer to D.Phil. Status, Oxford University, 1997.

[Oua98a] J. Ouaknine. Model-checking real-time concurrent systems. In *Abstracts for the 1998 SUN Student Conference*. Oxford University Computing Laboratory, 1998.

[Oua98b] J. Ouaknine. Towards automated verification of Timed CSP, July 1998. Presentation, *DERA workshop on Advances in Formal Analysis of Critical Systems*, Royal Holloway, University of London.

[Oua99] J. Ouaknine. A framework for model-checking Timed CSP. In *Proceedings of the Colloquium on Applicable Modelling, Verification and Analysis Techniques for Real-Time Systems*. The Institution of Electrical Engineers, 1999.

[Oua00] J. Ouaknine. Strong (time-)bisimilarity is a congruence for (Timed) CSP. Unpublished, 2000.

[Phi87] I. Phillips. Refusal testing. *Theoretical Computer Science*, 50(3):241–284, 1987.

[QS81] J. P. Queille and J. Sifakis. Specification and verification of concurrent systems in cesar. In *Proceedings of the Fifth International Symposium on Programming*, volume 137, pages 337–351. Springer LNCS, 1981.

[Ree88] G. M. Reed. *A Mathematical Theory for Real-Time Distributed Computing*. PhD thesis, Oxford University, 1988.

[Ree89] G. M. Reed. A hierarchy of models for real-time distributed computing. In *Proceedings of the Fourth Conference on Mathematical Foundations of Programming Semantics (MFPS 89)*, volume 442, pages 80–128. Springer LNCS, 1989.

[Ros95] A. W. Roscoe. CSP and determinism in security modelling. In *Proceedings of the 1995 IEEE Symposium on Security and Privacy*, pages 114–127, 1995.

[Ros97] A. W. Roscoe. *The Theory and Practice of Concurrency.* Prentice-Hall International, London, 1997.

[RR86] G. M. Reed and A. W. Roscoe. A timed model for communicating sequential processes. In *Proceedings of the Thirteenth International Colloquium on Automata, Languages, and Programming (ICALP 86)*, pages 314–323. Springer LNCS, 1986. *Theoretical Computer Science*, 58:249–261.

[RR87] G. M. Reed and A. W. Roscoe. Metric spaces as models for real-time concurrency. In *Proceedings of the Second Conference on Mathematical Foundations of Programming Semantics (MFPS 87)*, volume 298, pages 331–343. Springer LNCS, 1987.

[RR99] G. M. Reed and A. W. Roscoe. The timed failures-stability model for CSP. *Theoretical Computer Science*, 211:85–127, 1999.

[RRS91] G. M. Reed, A. W. Roscoe, and S. A. Schneider. CSP and timewise refinement. In *Proceedings of the Fourth BCS-FACS Refinement Workshop*, Cambridge, 1991. Springer WIC.

[Sch89] S. A. Schneider. *Correctness and Communication in Real-Time Systems.* PhD thesis, Oxford University, 1989.

[Sch92] S. A. Schneider. Unbounded nondeterminism for real-time processes. Unpublished, 1992.

[Sch94] S. A. Schneider. Using CSP with Z in the mine pump case study. Unpublished, 1994.

[Sch95] S. A. Schneider. An operational semantics for Timed CSP. *Information and Computation*, 116:193–213, 1995.

[Sch97] S. A. Schneider. Timewise refinement for communicating processes. *Science of Computer Programming*, 28:43–90, 1997.

[Sch00] S. A. Schneider. *Concurrent and Real Time Systems: the CSP approach.* John Wiley, 2000.

[SS95] O. V. Sokolsky and S. A. Smolka. Local model checking for real time systems. In *Proceedings of the Seventh International Conference on Computer-Aided Verification (CAV 95)*, volume 939, pages 211–224. Springer LNCS, 1995.

[Ver97] J. J. Vereijken. *Discrete-Time Process Algebra.* PhD thesis, Eindhoven University of Technology, 1997.

[Wan91] Y. Wang. *A Calculus of Real-Time Systems.* PhD thesis, Chalmers University of Technology, 1991.

[YML99] Y. Yu, P. Manolios, and L. Lamport. Model checking TLA$^+$ specifications. In *Proceedings of the Tenth Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME 99)*, volume 1703, pages 54–66. Springer LNCS, 1999.