Centrum voor Wiskunde en Informatica
REPORT*RAPPORT*

Truth of duration calculus formulae in timed frames

C.A. Middelburg

Software Engineering (SEN)

SEN-R9812 August, 1998

Report SEN-R9812
ISSN 1386-369X

CWI
P.O. Box 94079
1090 GB Amsterdam
The Netherlands

Truth of Duration Calculus Formulae in Timed Frames

C.A. Middelburg
CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
and
Department of Philosophy, Utrecht University
P.O. Box 80126, 3508 TC Utrecht, The Netherlands
Email: cam@cwi.nl

ABSTRACT

Duration calculus is a logical formalism designed for expressing and refining real-time requirements for systems. Timed frames are essentially transition systems meant for modeling the time-dependent behaviour of programs. We investigate the interpretation of duration calculus formulae in timed frames. We elaborate this topic from different angles and show that they agree with each other. The resulting interpretation is expected to make it generally easier to establish semantic links between duration calculus and formalisms aimed at programming. Such semantic links are prerequisites for a solid underpinning of approaches to system development that cover requirement capture through coding using both duration calculus and some formalism(s) aimed at programming.

1991 Mathematics Subject Classification: 68Q55, 68Q60
1991 Computing Reviews Classification System: D.2.1, D.2.4, D.3.1, F.3.1, F.3.2
Keywords and Phrases: duration calculus, real-time requirements, timed frames, time-dependent behaviour, verification
Note: The work presented in this paper has been largely carried out while the author was at UNU/IIST (United Nations University, International Institute for Software Technology).
Note: This paper is a revised version of [23].

1. Introduction

Duration calculus [18, 36] is an interval temporal logic intended for expressing real-time requirements for systems and refining these requirements to specifications for one or more computer hardware and software components. Formalisms aimed at programming should subsequently be used for the stepwise development of the software components of the system to be realized. In the ProCoS project [19], a particular approach has been developed to safeguard that the resulting system conforms to the requirements expressed for it. The programming language considered is an occam-like language with real-time features, called PL. A timed version of the readiness model [27] is the basis of the semantics of PL. To bridge the gap between the state-based duration calculus and the event-based PL, a specification language for reactive systems with time constraints is introduced. This language, called SL, is closely related to the model that is the basis of the semantics of PL.

SL and PL have been given a semantics using duration calculus formulae to describe the meaning of SL specifications and using timing diagrams to describe the meaning of PL programs [32]. Because timing diagrams are in essence the objects with respect to which the truth of duration calculus formulae is usually defined, thus semantic links between duration calculus, SL and PL have been established that provide the ProCoS approach with a solid underpinning.

Several variations on the ProCoS approach are conceivable. Interesting ones may be obtained by retaining duration calculus as requirement language but choosing another specification language and/or programming language, e.g. RSL [29] is chosen in [34]. The mathematical model that is the

basis of the semantics of the language(s) concerned will bring on certain basic concepts and special notations. To describe the meaning of the language constructs, the use of these concepts and notations are most probably preferable to the use of timing diagrams and duration calculus formulae. A main objection to the use of duration calculus, which is geared to state-based description of systems, is that it does not match most languages related to programming real-time reactive systems, which are geared to event-based description, nicely. In this connection, it is interesting to see that it is not entirely clear how the definitions of the semantics of SL and PL presented in [32] relate to the earlier definitions, presented in [31] and [16, 25], which they replace. A main purpose of this paper is to make it generally easier to establish semantic links between duration calculus and formalisms aimed at programming.

Various mathematical models have been developed that are suited to be used as a semantic basis for programming languages and specification languages aimed at programming. The objects that underlie many of these models are transition systems. In most process algebras, for example, transition systems modulo an appropriate "process equivalence" are considered. Therefore it will often be relatively simple to establish a semantic link between duration calculus and the language(s) concerned in case a suitable interpretation of duration calculus formulae in transition systems is available. There exist several kinds of transition systems. The basic transition systems for the time free case have been extended to cope with time-dependent behaviour by adding time-stamps to states, transitions or both and/or by introducing special time transitions. Besides the time scale on which time is measured may be continuous or discrete and timing may be absolute or relative. The nature of the time scale makes all the difference.

In this paper, the interpretation of duration calculus formulae in timed frames is studied. Timed frames [7] cover virtually all kinds of transition systems for the discrete time case. Timed frames are in essence the two-phase transition systems considered in [6] as the objects underlying discrete time process algebras. They underlie well known discrete time process algebras such as ATP [26] and the discrete time extension of ACP presented in [5]. Two-phase transition systems are closely related to the real-time transition systems that underlie the real-time extension of ACP presented in [3]. In [4], it is shown that the model of the discrete time extension of ACP based on two-phase transition systems is isomorphic to a model based on the real-time transition systems that are discretized. For discretized real-time transition systems, it holds that transitions that may occur at some time between n and $n + 1$, may also occur at any other time between n and $n + 1$. Note that the real-time transition systems of [3] are more complicated than necessary and a simpler kind of transition systems is used in [15]. They are also closely related to the transition systems underlying Timed CCS [12] and the timed transition systems proposed in [21].

The discrete time case is considered suitable for formalisms aimed at programming. It permits to consider systems at a more abstract level than the continuous time case, a level where time is measured with finite precision. Often this level does not differ materially from the implementation level: software components of a system are executed on processors where the measure of time is provided by a discrete clock and, in case a physical system is controlled, the state of the physical system is sampled and adjusted at discrete points in time. Besides, the abstraction makes the time-dependent behaviour of programs amenable to analysis. Usually, the real-time requirements expressed for a system are meant to be interpreted in continuous time. In such cases, a step from continuous time to discrete time has to be made at some stage of the refinement of the requirements for the system to the specifications for its components. This is necessary to achieve that the requirements are met in continuous time if all of the specifications are satisfied in discrete time. The problem of refining real-time requirements to such specifications in the setting of duration calculus is addressed in [13].

In duration calculus, real-time requirements are formulated as properties about the duration of phases of system behaviour. These phases, which are called state variables, are interpreted as functions from the time domain $\mathbb{R}^+$ to the Boolean domain $\{0, 1\}$. One way to connect duration calculus to timed frames is to extract interpretations of state variables from paths in frames. Another way is to give the meaning of formulae directly with respect to paths in frames. In this paper, we connect

duration calculus to timed frames in both ways and show that truth for a path is equivalent to truth under the interpretation induced by that path. Connecting duration calculus to timed frames by embedding of duration calculus into a classical first-order logic for timed frames, called timed frame logic [8], is doomed to fail, but embedding of an interesting fragment is feasible. This matter is treated as well in this paper.

To put it differently, the truth of duration calculus formulae in timed frames is presented in three ways: (1) by starting from the original (discrete time) semantics of duration calculus, (2) by introducing a new semantics directed at timed frames and (3) by giving a translation to timed frame logic that leaves validity unchanged. Because the first presentation is equivalent to the second presentation, whichever one of them is most convenient for a particular purpose may be used at any time. Seeing that the first presentation just takes paths in timed frames as representations of interpretations of state variables and thus provides a very simple interface between duration calculus and timed frames, it is considered to be the primary presentation. However, the second presentation is, for example, more suited for building a model-checking tool to verify automatically whether the time-dependent behaviour of a program, as modeled by a timed frame, meets a timing constraint expressed for it in duration calculus. The third presentation is equivalent to the others for the fragment concerned, because the translation leaves validity unchanged. Therefore, this presentation may also be used at any time that it is sufficient to look at the fragment. It may be convenient when devising a ProCoS-like approach with a logic closely related to timed frame logic, e.g. a timed version of Dicky logic [1] or Hennessy-Milner logic [20], as specification language.

Strictly speaking, we consider an extension of the original duration calculus, known as the mean value calculus [37], which allows to deal with point intervals and consequently with events. We also consider timed frames of which the states can be equipped with propositions, called signal inserted timed frames. This addition enables us to represent phases of system behaviour.

The structure of this paper is as follows. First of all, we give a survey of timed frame algebra (Section 2) and timed frame logic (Section 3). Next, we present the syntax and semantics of the mean value calculus (Section 4). After that, we connect the mean value calculus to timed frames (Section 5). Finally, we discuss the connection between the mean value calculus and timed frame logic (Section 6).

2. Timed frames

Simple timed frames are built from states and labelled transitions. There are two kinds of transitions, which we shall call action steps and time steps. They represent the execution of actions and the passage of time to the next time slice, respectively. Time determinism is not built into timed frames: states may have more than one outgoing time steps. By the addition of an operation, called signal insertion, it becomes possible to assign propositional formulae to the states of a timed frame. The propositional formula assigned to a certain state is considered to hold in that state. This section contains a survey of simple timed frame algebra and its extension with signal insertion. We refer to [7] for further details, which include results about the connection between timed frames and discrete time processes. The survey is preceded by a small example to illustrate the use of timed frames.

2.1 Example

The example concerns a simple telephone answering machine. We use timed frame algebra for the description of the control component of the telephone answering machine. The example is based on a specification in SDL [14] due to Mauw [22].

In order to control the telephone answering, the control component of the answering machine has to communicate with the recorder component of the answering machine, the telephone connected with the answering machine, and the telephone network. When an incoming call is detected, the answering is not started immediately. If the incoming call is broken off or the receiver of the telephone is lifted within a period of 10 time units, answering is discontinued. Otherwise, an off-hook signal is issued to the network when this period has elapsed and a pre-recorded message is played. Upon termination of the message, a beep signal is issued to the network and the recorder is started. The recorder is

stopped when the call is broken off, or when 30 time units have passed in case the call has not been broken off earlier. Thereafter, an on-hook signal is issued to the network.

We will use action steps and time steps to describe this time-dependent behaviour of the controller. Time steps are denoted by terms of the form $s \xrightarrow{\sigma} s'$ and action steps are denoted by terms of the form $s \xrightarrow{a} s'$, where a is an action. Action steps and time steps are considered timed frames and the frame union operator $\oplus$ is used to put together larger timed frames. Natural numbers are taken as states. The behaviour of the controller is represented by the timed frame *TAMC0* defined by

$$\begin{aligned}
&\mathit{TAMC0} \;=\\
&\quad (0 \xrightarrow{\sigma} 0) \oplus (0 \xrightarrow{r(\mathit{incoming\ call})} 1) \oplus\\
&\quad \textstyle\bigoplus_{i=1}^{10}((i \xrightarrow{\sigma} S(i)) \oplus (i \xrightarrow{r(\mathit{rcv\ lifted})} 0) \oplus (i \xrightarrow{r(\mathit{end\ call})} 0)) \oplus\\
&\quad (11 \xrightarrow{s(\mathit{off\text{-}hook})} 12) \oplus (12 \xrightarrow{s(\mathit{play\ msg})} 13) \oplus\\
&\quad (13 \xrightarrow{\sigma} 13) \oplus (13 \xrightarrow{r(\mathit{end\ msg})} 14) \oplus (13 \xrightarrow{r(\mathit{end\ call})} 48) \oplus\\
&\quad (14 \xrightarrow{s(\mathit{beep})} 15) \oplus (15 \xrightarrow{s(\mathit{start\ rec})} 16) \oplus\\
&\quad \textstyle\bigoplus_{j=16}^{45}((j \xrightarrow{\sigma} S(j)) \oplus (j \xrightarrow{r(\mathit{end\ call})} 47)) \oplus\\
&\quad (46 \xrightarrow{s(\mathit{stop\ rec})} 48) \oplus (47 \xrightarrow{s(\mathit{stop\ rec})} 48) \oplus (48 \xrightarrow{s(\mathit{on\text{-}hook})} 0)
\end{aligned}$$

By designating state 0 as the root state, we obtain a transition system. Instead of its usual graphical representation, we give here a term for it.

It may be useful to know whether the state of the answering machine is one of playing, recording or otherwise. Using the signal insertion operator $\frown\!\!\!\!\rightarrow$ to assign to each state of *TAMC0* a propositional formula that indicates whether it is a state of playing, recording or otherwise, we get the signal inserted timed frame *TAMC1* defined by

$$\begin{aligned}
&\mathit{TAMC1} \;=\\
&\quad \mathit{TAMC0} \oplus \textstyle\bigoplus_{i=0}^{11}((\neg \mathit{playing} \wedge \neg \mathit{recording}) \frown\!\!\!\!\rightarrow i) \oplus\\
&\quad ((\mathit{playing} \wedge \neg \mathit{recording}) \frown\!\!\!\!\rightarrow 13) \oplus \textstyle\bigoplus_{j=16}^{46}((\neg \mathit{playing} \wedge \mathit{recording}) \frown\!\!\!\!\rightarrow j)
\end{aligned}$$

The frames *TAMC0* and *TAMC1* defined here differ slightly from the ones defined in [7]. There states 46 and 47 were identified, because both were considered states of playing. Here the latter state is considered an internal state of which nothing should be made visible via a propositional formula assigned to it. Further distinctions could have been made, e.g. between states of idling and states of waiting to answer for the states of not playing and not recording.

2.2 Simple timed frames

Simple timed frames are built from states and transitions between states. The states are obtained by an embedding of natural numbers in states, and a pairing function on states. Simple timed frames contain two kinds of transitions: action steps and time steps. We consider action steps with a label from a finite set A of *actions*.

The signature of *(simple) timed frames* is as follows:

Sorts:

$\mathbb{N}$	natural numbers;
$\mathbb{S}$	*states*;
$\mathbb{F}_\mathrm{t}$	*timed frames*;

Constants & Functions:

$0 : \mathbb{N}$	zero;
$S : \mathbb{N} \to \mathbb{N}$	successor;
$\imath_\mathbb{N} : \mathbb{N} \to \mathbb{S}$	embedding of natural numbers in states;
$\rangle\!\!-\!\!\langle : \mathbb{S}^2 \to \mathbb{S}$	pairing of states;
$\emptyset : \mathbb{F}_\mathrm{t}$	*empty timed frame*;
$\imath_\mathbb{S} : \mathbb{S} \to \mathbb{F}_\mathrm{t}$	*embedding of states* in timed frames;
$\xrightarrow{a}: \mathbb{S}^2 \to \mathbb{F}_\mathrm{t}$	*action step construction* (one for each $a \in A$);
$\xrightarrow{\sigma}: \mathbb{S}^2 \to \mathbb{F}_\mathrm{t}$	*time step construction*;
$\oplus : \mathbb{F}_\mathrm{t}^2 \to \mathbb{F}_\mathrm{t}$	*timed frame union*.

Given the signature, (closed) terms are constructed in the usual way. We shall use the meta-variables n and m to stand for arbitrary terms of sort $\mathbb{N}$, the meta-variables s, s' and s'' to stand for arbitrary terms of sort $\mathbb{S}$, and the meta-variables X, Y and Z to stand for arbitrary terms of sort $\mathbb{F}_\mathrm{t}$. We write n instead of $\imath_\mathbb{N}(n)$ or $\imath_\mathbb{S}(\imath_\mathbb{N}(n))$ as well as s instead of $\imath_\mathbb{S}(s)$ when this causes no ambiguity. Terms of the forms $\imath_\mathbb{S}(s)$, $s \xrightarrow{a} s'$ and $s \xrightarrow{\sigma} s'$ denote *atomic* timed frames, i.e. timed frames that contain a single state or transition. The constant $\emptyset$ denotes the timed frame that contains neither states nor transitions. The operator $\oplus$ on timed frames gives the union of the states and transitions of its arguments. Pairing $\rangle\!\!-\!\!\langle$ is a simple means to define "fresh" states – in [11], it is used to define frame product. The axioms for timed frames are given in Table 1. These axioms characterize timed frames

(FA1)	$X \oplus Y$	$=$	$Y \oplus X$
(FA2)	$X \oplus (Y \oplus Z)$	$=$	$(X \oplus Y) \oplus Z$
(FA3)	$X \oplus X$	$=$	X
(FA4)	$X \oplus \emptyset$	$=$	X
(FA5)	$s \oplus (s \xrightarrow{a} s')$	$=$	$s \xrightarrow{a} s'$
(FA6)	$s' \oplus (s \xrightarrow{a} s')$	$=$	$s \xrightarrow{a} s'$
(TFA1)	$s \oplus (s \xrightarrow{\sigma} s')$	$=$	$s \xrightarrow{\sigma} s'$
(TFA2)	$s' \oplus (s \xrightarrow{\sigma} s')$	$=$	$s \xrightarrow{\sigma} s'$

Table 1: Axioms for timed frames.

as objects consisting of a finite set of states and a finite set of transitions (axioms (FA1)–(FA4)). In addition, timed frames are identified if they are the same after addition of the states occurring in the transitions to the set of states (axioms (FA5), (FA6), (TFA1) and (TFA2)). Notice that time steps are not treated different from action steps in the axioms for simple timed frames. However, the distinction between action steps and time steps is needed for the extension with signal insertion. Besides, it is of vital importance to relate timed frames to discrete time processes.

We define *iterated* frame union by

$$\bigoplus_{i=n}^{k} X_i = \begin{cases} \emptyset & \text{if } k < n, \\ X_n \oplus \bigoplus_{i=n+1}^{k} X_i & \text{otherwise.} \end{cases}$$

Every frame has a finite number of states and transitions, and can be denoted by a term of the form $\bigoplus_{i=1}^{m} X_i$, where the X_i are atomic. In [11], frame polynomials are introduced to deal with the

countably infinite case as well. However, in this paper only frames with a finite number of states and transitions – which correspond to *regular* discrete time processes as defined in [5] – are considered.

2.3 Signal inserted timed frames

In simple timed frames, states are not labelled. In signal inserted timed frames, we consider states with a label from the set of propositional formulae that can be built from a set $\mathbb{P}_{at}$ of *atomic propositions*, t, f, and the connectives $\neg$ and $\rightarrow$. The propositional formula assigned to a state is considered to hold in that state.

The signature for *signal inserted* timed frames is the signature of timed frames, where the sort $\mathbb{F}_{\mathrm{t}}$ is renamed to $\langle \mathbb{F}_{\mathrm{t}}, \mathbb{P} \rangle$, extended with the following:

Sorts:

$\mathbb{P}$	propositions;

Constants & Functions:

$p \ : \mathbb{P}$	for each $p \in \mathbb{P}_{at}$;
$\mathsf{t} \ : \mathbb{P}$	true;
$\mathsf{f} \ : \mathbb{P}$	false;
$\neg \ : \mathbb{P} \rightarrow \mathbb{P}$	negation;
$\rightarrow \ : \mathbb{P}^2 \rightarrow \mathbb{P}$	implication;
$\frown\!\!\!\rightarrow : \mathbb{P} \times \langle \mathbb{F}_{\mathrm{t}}, \mathbb{P} \rangle \rightarrow \langle \mathbb{F}_{\mathrm{t}}, \mathbb{P} \rangle$	*signal insertion.*

The signature of signal inserted timed frames is graphically presented in Figure 1. We shall use the

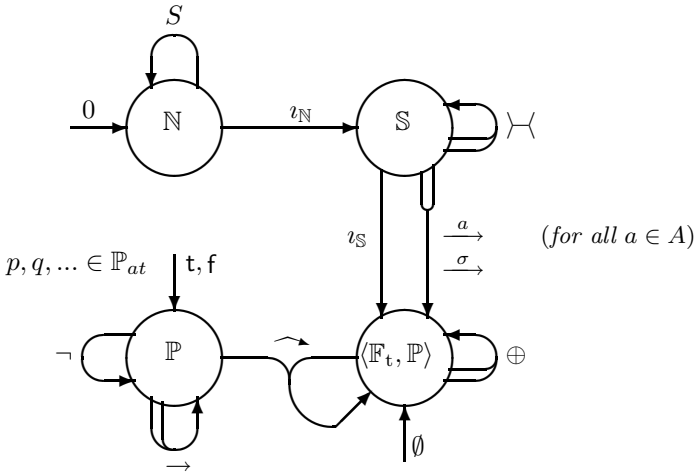

Figure 1: Signature of signal inserted timed frames.

meta-variables ϕ and ψ to stand for arbitrary terms of sort $\mathbb{P}$. As usual, we write $\phi \vee \psi$ for $\neg\phi \rightarrow \psi$, $\phi \wedge \psi$ for $\neg(\neg\phi \vee \neg\psi)$, and $\phi \leftrightarrow \psi$ for $(\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi)$. In Table 2 we give a complete proof system for propositional logic. The signal insertion operation $\frown\!\!\!\rightarrow$ assigns propositional formulae to the states contained in frames. The axioms for signal inserted timed frames are the axioms given in Table 1 (see Section 2.2) and the axioms given in Table 3. Additionally, we can use identities $\phi = \psi$ iff $\phi \leftrightarrow \psi$ is provable from the axiom schemas and the inference rule given in Table 2. The axioms in Table 3 express that signal insertion to a frame is tantamount to signal insertion to all its states, taken as frames (axioms (Ins1), (Ins5), (Ins6) and (TIns1)). The axioms (Ins2)–(Ins4) cover the special cases where signal insertion is not applied once, but zero times or more than once. A signal inserted state $\mathsf{f} \frown\!\!\!\rightarrow s$, i.e. a state where f holds, is an inconsistent state which absorbs all its incoming and outgoing action steps (axioms (Ins7) and (Ins8)). The axiom (TIns2) reflects the intuition that the passage

(P1)	$\phi \to (\psi \to \phi)$
(P2)	$(\phi \to (\psi \to \xi)) \to ((\phi \to \psi) \to (\phi \to \xi))$
(P3)	$(\neg\phi \to \neg\psi) \to (\psi \to \phi)$
(P4)	$\mathsf{t} \leftrightarrow (p \to p)$
(P5)	$\mathsf{f} \leftrightarrow \neg\mathsf{t}$
(MP)	$\dfrac{\phi \quad \phi \to \psi}{\psi}$

Table 2: A proof system for propositional logic.

(Ins1)	$\phi \overset{\frown}{\blacktriangleright} \emptyset$	$=$	$\emptyset$
(Ins2)	$\mathsf{t} \overset{\frown}{\blacktriangleright} X$	$=$	X
(Ins3)	$\phi \overset{\frown}{\blacktriangleright} (\psi \overset{\frown}{\blacktriangleright} X)$	$=$	$(\phi \wedge \psi) \overset{\frown}{\blacktriangleright} X$
(Ins4)	$(\phi \overset{\frown}{\blacktriangleright} X) \oplus (\psi \overset{\frown}{\blacktriangleright} X)$	$=$	$(\phi \wedge \psi) \overset{\frown}{\blacktriangleright} X$
(Ins5)	$\phi \overset{\frown}{\blacktriangleright} (X \oplus Y)$	$=$	$(\phi \overset{\frown}{\blacktriangleright} X) \oplus (\phi \overset{\frown}{\blacktriangleright} Y)$
(Ins6)	$\phi \overset{\frown}{\blacktriangleright} (s \xrightarrow{a} s')$	$=$	$(\phi \overset{\frown}{\blacktriangleright} s) \oplus (s \xrightarrow{a} s') \oplus (\phi \overset{\frown}{\blacktriangleright} s')$
(Ins7)	$(\mathsf{f} \overset{\frown}{\blacktriangleright} s) \oplus (s \xrightarrow{a} s')$	$=$	$(\mathsf{f} \overset{\frown}{\blacktriangleright} s) \oplus s'$
(Ins8)	$(s \xrightarrow{a} s') \oplus (\mathsf{f} \overset{\frown}{\blacktriangleright} s')$	$=$	$s \oplus (\mathsf{f} \overset{\frown}{\blacktriangleright} s')$
(TIns1)	$\phi \overset{\frown}{\blacktriangleright} (s \xrightarrow{\sigma} s')$	$=$	$(\phi \overset{\frown}{\blacktriangleright} s) \oplus (s \xrightarrow{\sigma} s') \oplus (\phi \overset{\frown}{\blacktriangleright} s')$
(TIns2)	$(\phi \overset{\frown}{\blacktriangleright} s) \oplus (s \xrightarrow{\sigma} s')$	$=$	$(s \xrightarrow{\sigma} s') \oplus (\phi \overset{\frown}{\blacktriangleright} s')$

Table 3: Additional axioms for signal insertion.

of time cannot change the propositions that hold in the current state. Axiom (TIns2) entails that inconsistent states remain inconsistent with progress of time. Thus, one inconsistent state would render all states inconsistent if there were also counterparts of the axioms (Ins7) and (Ins8) for time steps. Note that the equation $s \oplus (\phi \overset{\frown}{\blacktriangleright} s) = \phi \overset{\frown}{\blacktriangleright} s$ (reminiscent of the axioms (FA5) and (FA6)) is derivable from the axioms (Ins2) and (Ins4).

3. Timed frame logic

Timed Frame Logic (TFL) is a classical first-order logic with:

1. quantification over natural numbers, states, *transition labels* and *paths*;
2. standard constants and functions concerning natural numbers, states, propositions, transition labels and paths;
3. equality and some additional standard predicates concerning paths.

TFL was first proposed as a logic for timed frames in [8]. That paper reports in detail about various issues, including the distinctive power of TFL and the embedding of other logics (CTL and Dicky logic) in TFL. In this section, we only present the syntax and semantics of TFL.

3.1 Syntax

The signature for the terms of TFL is the signature of signal inserted timed frames restricted to natural numbers, states and propositions, and extended with the following:

Sorts:

$\mathbb{L}$	*transition labels*;
Π	*paths*;

Constants & Functions:

$+ : \mathbb{N}^2 \to \mathbb{N}$	addition;
$a : \mathbb{L}$	for each $a \in A$;
$\sigma : \mathbb{L}$	time step label;
$\imath_{\mathbb{S}} : \mathbb{S} \to \Pi$	embedding of states in paths;
$\dot{\longrightarrow} : \Pi \times \mathbb{L} \times \mathbb{S} \to \Pi$	appending of transitions to paths.

For the sorts $\mathbb{N}$, $\mathbb{S}$, $\mathbb{L}$ and Π, we assume countably infinite sets of variables $\mathcal{V}_{\mathbb{N}}$, $\mathcal{V}_{\mathbb{S}}$, $\mathcal{V}_{\mathbb{L}}$ and $\mathcal{V}_{\Pi}$, respectively. Terms of these sorts are constructed from the variables and the constant and function symbols in the usual way. For the sort $\mathbb{P}$, we only consider variable-free terms. We shall use the meta-variables t and t' to stand for arbitrary terms of any sort, the meta-variable μ to stand for an arbitrary term of sort $\mathbb{L}$ and the meta-variable π to stand for an arbitrary term of sort Π.

The atomic formulae of TFL are inductively defined by the following formation rules:

1. if t, t' are terms of the same sort, then $t = t'$ is an atomic formula;
2. if n, π, and s are terms of sort $\mathbb{N}$, Π and $\mathbb{S}$, respectively, then $\mathsf{S_s}(n, \pi, s)$ is an atomic formula;
3. if n, π, and μ are terms of sort $\mathbb{N}$, Π and $\mathbb{L}$, respectively, then $\mathsf{S_l}(n, \pi, \mu)$ is an atomic formula;
4. if π is a term of sort Π, then $\mathsf{E}(\pi)$ is an atomic formula;
5. if ϕ is a variable-free term of sort $\mathbb{P}$ and s is a term of sort $\mathbb{S}$, then $\mathsf{H}(\phi, s)$ is an atomic formula.

The formulae of TFL are inductively defined by the following formation rules:

1. atomic formulae are formulae;
2. if Φ is a formula, then $\neg\Phi$ is a formula;
3. if Φ, Ψ are formulae, then $\Phi \wedge \Psi$ is a formula;
4. if $x \in \mathcal{V}_D$, where $D \in \{\mathbb{N}, \mathbb{S}, \mathbb{L}, \Pi\}$, and Φ is a formula, then $\forall x \in D \cdot \Phi$ is a formula.

The meaning of the atomic formulae of the first form is as usual. The meaning of the atomic formulae of the last four forms can informally be explained as follows: $\mathsf{S_s}(n, \pi, s)$ is true iff s is the $(n+1)$-th state in path π, $\mathsf{S_l}(n, \pi, \mu)$ is true iff μ is the label of the $(n+1)$-th transition in path π, $\mathsf{E}(\pi)$ is true in a frame iff the path π exists in the frame, and $\mathsf{H}(\phi, s)$ is true in a frame iff the proposition ϕ holds in the state s of the frame. Obviously, the truth of the atomic formula of the forms $\mathsf{S_s}(n, \pi, s)$ and $\mathsf{S_l}(n, \pi, \mu)$ are not frame dependent. For the selection of states and transition labels from paths, standard predicates are provided instead of standard functions because the latter would be partial functions.

Additional connectives ($\vee$, $\rightarrow$, $\leftrightarrow$) and the existential quantifier are introduced as abbreviations in the usual way. The abbreviation $t \neq t'$ for $\neg(t = t')$ is also used.

3.2 Example

In Section 2.1, the control component of a telephone answering machine was modelled by a timed frame. One of its properties is the following:

> When the off-hook signal is issued to the telephone network, nothing has happened since the detection of the last incoming call and meanwhile 10 time units have passed.

This property can be expressed in TFL as follows:

$$
\begin{aligned}
&\forall \pi \in \Pi \cdot \forall n \in \mathbb{N} \cdot \\
&\quad \mathsf{E}(\pi) \wedge \mathsf{S_l}(0, \pi, \mathit{r(incoming\ call)}) \wedge \mathsf{S_l}(n+1, \pi, \mathit{s(off\text{-}hook)}) \wedge \\
&\quad (\forall m \in \mathbb{N} \cdot 1 \leq m \leq n \rightarrow \neg \mathsf{S_l}(m, \pi, \mathit{r(incoming\ call)})) \rightarrow \\
&\qquad\qquad n = 10 \wedge \forall k \in \mathbb{N} \cdot 1 \leq k \leq n \rightarrow \mathsf{S_l}(k, \pi, \sigma)
\end{aligned}
$$

Here we write $l \leq m \leq n$ for $\exists k \in \mathbb{N} \cdot k + l = m \wedge \exists k' \in \mathbb{N} \cdot k' + m = n$.

3.3 Semantics

The interpretation of the sort, constant and function symbols from the signature of the TFL terms is the interpretation in the initial model for this signature and the usual equations concerning 0, S and $+$. This interpretation can be extended to the TFL terms in the usual way. We write $[\![t]\!]_\alpha$ for the interpretation of term t under an assignment α. An assignment is a function mapping each variable to an element of the interpretation of its sort in the initial model. If D is a sort symbol, we write D for its interpretation as well. It is always clear from the context whether the symbol or its interpretation is meant.

The predicates symbols $\mathsf{S_s}$, $\mathsf{S_l}$, E and H have also a standard meaning which was explained informally above. In case of $\mathsf{S_s}$ and $\mathsf{S_l}$, the meaning is frame independent. These symbols stand for the ternary relations $\boldsymbol{S_s} \subseteq \mathbb{N} \times \Pi \times \mathbb{S}$ and $\boldsymbol{S_l} \subseteq \mathbb{N} \times \Pi \times \mathbb{L}$ inductively defined by

$$i \leq n \quad \Rightarrow \quad (i, s_1 \xrightarrow{\mu_1} s_2 \ldots \xrightarrow{\mu_n} s_{n+1}, s_{i+1}) \in \boldsymbol{S_s}$$

and

$$i < n \quad \Rightarrow \quad (i, s_1 \xrightarrow{\mu_1} s_2 \ldots \xrightarrow{\mu_n} s_{n+1}, \mu_{i+1}) \in \boldsymbol{S_l}$$

In case of E and H, the meaning is frame dependent. For each frame F, these symbols stand for the unary relation $\boldsymbol{E}(F) \subseteq \Pi$ and the binary relation $\boldsymbol{H}(F) \subseteq \mathbb{P} \times \mathbb{S}$ inductively defined by

$$\begin{array}{rcl} s_1 \oplus F = F & \Rightarrow & s_1 \in \boldsymbol{E}(F), \\ \bigoplus_{i=1}^{n}(s_i \xrightarrow{\mu_i} s_{i+1}) \oplus F = F & \Rightarrow & s_1 \xrightarrow{\mu_1} s_2 \ldots \xrightarrow{\mu_n} s_{n+1} \in \boldsymbol{E}(F) \end{array}$$

and

$$(\phi \frown s) \oplus F = F \quad \Rightarrow \quad (\phi, s) \in \boldsymbol{H}(F)$$

The truth of a formula Φ in frame F under assignment α, written $F \models_\alpha \Phi$, is inductively defined by

$$\begin{array}{rcl} F \models_\alpha t = t' & \Leftrightarrow & [\![t]\!]_\alpha = [\![t']\!]_\alpha, \\ F \models_\alpha \mathsf{S_s}(n, \pi, s) & \Leftrightarrow & ([\![n]\!]_\alpha, [\![\pi]\!]_\alpha, [\![s]\!]_\alpha) \in \boldsymbol{S_s}, \\ F \models_\alpha \mathsf{S_l}(n, \pi, \mu) & \Leftrightarrow & ([\![n]\!]_\alpha, [\![\pi]\!]_\alpha, [\![\mu]\!]_\alpha) \in \boldsymbol{S_l}, \\ F \models_\alpha \mathsf{E}(\pi) & \Leftrightarrow & [\![\pi]\!]_\alpha \in \boldsymbol{E}(F), \\ F \models_\alpha \mathsf{H}(\phi, s) & \Leftrightarrow & ([\![\phi]\!]_\alpha, [\![s]\!]_\alpha) \in \boldsymbol{H}(F), \\ F \models_\alpha \neg\Phi & \Leftrightarrow & \text{not } F \models_\alpha \Phi, \\ F \models_\alpha \Phi \wedge \Psi & \Leftrightarrow & F \models_\alpha \Phi \text{ and } F \models_\alpha \Psi, \\ F \models_\alpha \forall x \in D \cdot \Phi & \Leftrightarrow & \text{for all } d \in D, F \models_{\alpha(x \to d)} \Phi \\ & & (\text{for } D \in \{\mathbb{N}, \mathbb{S}, \mathbb{L}, \Pi\}). \end{array}$$

Here we write $\alpha(x \to d)$ for the assignment α' such that $\alpha'(y) = \alpha(y)$ if $y \neq x$ and $\alpha'(x) = d$.

A formula Φ is *valid in* a frame F, written $F \models \Phi$, iff $F \models_\alpha \Phi$ for all assignments α. A formula Φ is *valid*, written $\models \Phi$, iff $F \models \Phi$ for all frames F.

A frame F has *inconsistent states* iff there is a state s such that $(\mathsf{f} \frown s) \oplus F = F$. The following results concerning the distinctive power of TFL were proved in [8]:

1. $F \neq F' \;\Rightarrow\; (\text{for some } \Phi,\ F \models \Phi \not\Leftrightarrow F' \models \Phi)$
2. if F and F' have no inconsistent states:
   $(\text{for some } \Phi,\ F \models \Phi \not\Leftrightarrow F' \models \Phi) \;\Rightarrow\; F \neq F'$

This means that for frames without inconsistent states equality coincides with the nonexistence of a distinguishing TFL formula.

4. Duration calculus

The original Duration Calculus (DC) was introduced in [36]. Its discrete time semantics can be found in e.g. [17]. Several extensions have been proposed, notably the Mean Value Calculus (MVC) in [37] and the Extended Duration Calculus (EDC) in [38]. As explained in Section 1, we consider MVC with discrete time semantics.

In both DC and MVC, a system is modelled by a number of functions from the time domain $\mathbb{R}^+$ to the Boolean domain $\{0, 1\}$. These functions are called the state variables of the system. State variables, durations and the chop modality are the distinctive features of DC. For a state variable (or a Boolean combination of state variables) P, its duration in a time interval, written $\int P$ in DC, is the integral of P over the time interval. For formulae Φ and Ψ, the formula $\Phi \,;\, \Psi$, where ; denotes the chop modality, can be formed. This formula is true at a time interval that can be divided into two intervals where Φ is true at the first interval and Ψ is true at the second interval. In MVC, durations are replaced by mean values and interval-lengths. For a state variable (or a Boolean combination of state variables) P, its mean value, written $\overline{P}$, is the mean value of P over a time interval if the interval is not a point interval, and the value of P at the point otherwise. ℓ stands for the length of a time interval. In MVC, the duration of P can be written $\overline{P} * \ell$.

4.1 Syntax

We assume a countably infinite set of logical variables $\mathcal{V}$ and a countably infinite set of state variables $\mathcal{SV}$. Furthermore, we assume a finite sets of function symbols (each with an associated arity) and a finite set of predicate symbols (each with an associated arity). In MVC we have, in addition to the syntactic categories of terms and formulae, the syntactic category of *state expressions*.

The state expressions are inductively defined by the following formation rules:

1. 0 and 1 are state expressions;
2. each $v \in \mathcal{SV}$ is a state expression;
3. if P is a state expression, then $\neg P$ is a state expression;
4. if P, Q are state expressions, then $P \wedge Q$ is a state expression.

The terms of MVC are inductively defined by the following formation rules:

1. ℓ is a term;
2. each $x \in \mathcal{V}$ is a term;
3. if P is a state expression, then $\overline{P}$ is a term;
4. if $r_1, \ldots, r_n$ are terms and f is an n-ary function symbol, then $f(r_1, \ldots, r_n)$ is a term.

The formulae of MVC are inductively defined by the following formation rules:

1. $\mathtt{t}$ is a formula;
2. if r, r' are terms, then $r = r'$ is a formula;
3. if $r_1, \ldots, r_n$ are terms and p is an n-ary predicate symbol, then $p(r_1, \ldots, r_n)$ is a formula;
4. if Φ is a formula, then $\neg\Phi$ is a formula;
5. if Φ, Ψ are formulae, then $\Phi \wedge \Psi$ and $\Phi \,;\, \Psi$ are formulae;
6. if $x \in \mathcal{V}$ and Φ is a formula, then $\forall x \cdot \Phi$ is a formula.

Additional connectives ($\vee$, $\rightarrow$, $\leftrightarrow$) and the existential quantifier are introduced as abbreviations in the usual way.

The following abbreviations are also frequently used: $\lceil P \rceil^0$ for $\ell = 0 \wedge \overline{P} = 1$ and $\lceil P \rceil$ for $\ell > 0 \wedge \neg(\ell > 0 \,;\, \lceil \neg P \rceil^0 \,;\, \ell > 0)$. Their meaning can be informally explained as follows: $\lceil P \rceil^0$ is true at an interval iff the interval is a point interval and P has the value 1 at that point, and $\lceil P \rceil$ is true at an interval iff the interval is not a point interval and P has the value 1 everywhere inside the interval – P may have the value 0 at the begin-point and the end-point.

4.2 Semantics

We assume that there is a function $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}$ associated with each n-ary function symbol f and a relation $\boldsymbol{p} : \mathbb{R}^n$ with each n-ary predicate symbol p. We write $[b, e]$, where $b, e \in \mathbb{R}^+$ and $b \leq e$, for bounded and closed intervals.

The truth of MVC formulae is defined below with respect to an interpretation of state variables and an assignment of logical variables. Let $N \in \mathbb{N}$. Then a (discrete) *interpretation* $\mathcal{I}$ over the interval $[0, N]$ is a function $\mathcal{I} : \mathcal{SV} \to ([0, N] \to \{0, 1\})$ where, for each $v \in \mathcal{SV}$, the discontinuity points of $\mathcal{I}(v)$ belong to $\mathbb{N}$. Likewise, we only consider discrete intervals, i.e. intervals $[b, e]$ where $b, e \in \mathbb{N}$. We write $\mathit{Intv}(N)$ for $\{[b, e] \mid b, e \in \mathbb{N}, 0 \leq b \leq e \leq N\}$. An *assignment* α is a function $\alpha : \mathcal{V} \to \mathbb{R}$.

The value of a state expression P under interpretation $\mathcal{I}$ over $[0, N]$ is the function $[\![P]\!]^{\mathcal{I}} : [0, N] \to \{0, 1\}$ inductively defined by

$$
\begin{aligned}
[\![0]\!]^{\mathcal{I}}(t) &= 0, \\
[\![1]\!]^{\mathcal{I}}(t) &= 1, \\
[\![v]\!]^{\mathcal{I}}(t) &= \mathcal{I}(v)(t), \\
[\![\neg P]\!]^{\mathcal{I}}(t) &= 1 - [\![P]\!]^{\mathcal{I}}(t), \\
[\![P \wedge Q]\!]^{\mathcal{I}}(t) &= \begin{cases} 1 & \text{if } [\![P]\!]^{\mathcal{I}}(t) = 1 \text{ and } [\![Q]\!]^{\mathcal{I}}(t) = 1 \\ 0 & \text{otherwise.} \end{cases}
\end{aligned}
$$

The value of a term r under interpretation $\mathcal{I}$ over $[0, N]$ and assignment α is the function $[\![r]\!]^{\mathcal{I}}_{\alpha} : \mathit{Intv}(N) \to \mathbb{R}$ inductively defined by

$$
\begin{aligned}
[\![\ell]\!]^{\mathcal{I}}_{\alpha}([b, e]) &= e - b, \\
[\![x]\!]^{\mathcal{I}}_{\alpha}([b, e]) &= \alpha(x), \\
[\![\overline{P}]\!]^{\mathcal{I}}_{\alpha}([b, e]) &= \begin{cases} \displaystyle\int_b^e [\![P]\!]^{\mathcal{I}}(t)dt/(e - b) & \text{if } e - b > 0 \\ [\![P]\!]^{\mathcal{I}}(e) & \text{otherwise,} \end{cases} \\
[\![f(r_1, \ldots, r_n)]\!]^{\mathcal{I}}_{\alpha}([b, e]) &= \boldsymbol{f}([\![r_1]\!]^{\mathcal{I}}_{\alpha}([b, e]), \ldots, [\![r_n]\!]^{\mathcal{I}}_{\alpha}([b, e])).
\end{aligned}
$$

The truth of a formula Φ at interval $[b, e] \in \mathit{Intv}(N)$ under interpretation $\mathcal{I}$ over $[0, N]$ and assignment α, written $\mathcal{I}, [b, e] \models_{\alpha} \Phi$, is inductively defined by

$$
\begin{aligned}
\mathcal{I}, [b, e] &\models_{\alpha} \mathtt{t}, \\
\mathcal{I}, [b, e] &\models_{\alpha} r = r' \quad \Leftrightarrow \quad [\![r]\!]^{\mathcal{I}}_{\alpha}([b, e]) = [\![r']\!]^{\mathcal{I}}_{\alpha}([b, e]), \\
\mathcal{I}, [b, e] &\models_{\alpha} p(r_1, \ldots, r_n) \quad \Leftrightarrow \quad ([\![r_1]\!]^{\mathcal{I}}_{\alpha}([b, e]), \ldots, [\![r_n]\!]^{\mathcal{I}}_{\alpha}([b, e])) \in \boldsymbol{p}, \\
\mathcal{I}, [b, e] &\models_{\alpha} \neg\Phi \quad \Leftrightarrow \quad \text{not } \mathcal{I}, [b, e] \models_{\alpha} \Phi, \\
\mathcal{I}, [b, e] &\models_{\alpha} \Phi \wedge \Psi \quad \Leftrightarrow \quad \mathcal{I}, [b, e] \models_{\alpha} \Phi \text{ and } \mathcal{I}, [b, e] \models_{\alpha} \Psi, \\
\mathcal{I}, [b, e] &\models_{\alpha} \Phi \,;\, \Psi \quad \Leftrightarrow \quad \text{for some } m \in \mathbb{N} \text{ where } m \in [b, e], \\
&\qquad\qquad\quad\ \ \mathcal{I}, [b, m] \models_{\alpha} \Phi \text{ and } \mathcal{I}, [m, e] \models_{\alpha} \Psi, \\
\mathcal{I}, [b, e] &\models_{\alpha} \forall x \cdot \Phi \quad \Leftrightarrow \quad \text{for all } d \in \mathbb{R},\ \mathcal{I}, [b, e] \models_{\alpha(x \to d)} \Phi.
\end{aligned}
$$

We write $\mathcal{I}, [b, e] \models \Phi$ to indicate that $\mathcal{I}, [b, e] \models_{\alpha} \Phi$ for all assignments α.

A formula Φ is *valid in* an interpretation $\mathcal{I}$ over $[0, N]$, written $\mathcal{I} \models \Phi$, iff $\mathcal{I}, [0, N] \models \Phi$. A formula Φ is *valid*, written $\models \Phi$, iff $\mathcal{I} \models \Phi$ for all interpretations $\mathcal{I}$ over $[0, N]$, for all N.

5. Duration calculus for timed frames

In this section, we consider the truth of MVC formulae in signal inserted timed frames. First of all, we show how paths in a frame induce interpretations of state variables and we take the truth of an

MVC formula under all interpretations induced by paths in a frame as the validity of the formula in that frame. After that, we make a more direct connection by introducing the truth of MVC formulae for paths in frames. Truth for a path is equivalent to truth under the interpretation induced by the path. In other words, we consider the truth of MVC formulae in signal inserted timed frames from two different angles that agree with each other: an angle that focusses on the original semantics of MVC and an angle that focusses on a new semantics directed at signal inserted timed frames. Each of the two resulting presentations of the truth of MVC formulae in signal inserted timed frames can be safely used at different times. The usefulness of this is further discussed in Section 1. An additional advantage of an elaboration from more than one angle is that it leads to definitions that are not biased by a particular angle.

To begin with, we introduce some auxiliary notions and notations to make the main definitions easier to comprehend.

A *proper path* is a path of the form $s_1 \xrightarrow{\mu_1} s_2 \ldots \xrightarrow{\mu_n} s_{n+1}$ where $\mu_n \neq \sigma$. So proper paths can not end in a time step. We write $\pi \in \boldsymbol{\Pi_p}$ to indicate that π is a proper path.

The partial *path composition* function $\bullet : \Pi \times \Pi \to \Pi$ is inductively defined by

$$\begin{array}{lll} & & s \bullet s = s \\ & & (\pi \xrightarrow{\mu} s) \bullet s = \pi \xrightarrow{\mu} s \\ \pi_1 \bullet \pi_2 = \pi_3 & \Rightarrow & \pi_1 \bullet (\pi_2 \xrightarrow{\mu} s) = \pi_3 \xrightarrow{\mu} s \end{array}$$

Path composition yields the concatenation of two paths, provided that the last state of the first path equals the first state of the second path. Otherwise its result is undefined.

A *timed action step* $s \xrightarrow{t,a} s'$ $(t \in \mathbb{N})$ is a path of the form $s \xrightarrow{\sigma} s_1 \ldots \xrightarrow{\sigma} s_t \xrightarrow{a} s'$.

Similarly, a *timed action path* $s_1 \xrightarrow{t_1,a_1} s_2 \ldots s_n \xrightarrow{t_n,a_n} s_{n+1}$ is a path of the form $(s_1 \xrightarrow{t_1,a_1} s_2) \bullet \ldots \bullet (s_n \xrightarrow{t_n,a_n} s_{n+1})$.

Note that a timed action path $s_1 \xrightarrow{t_1,a_1} s_2 \ldots s_n \xrightarrow{t_n,a_n} s_{n+1}$ hides the states $s_{i1}, \ldots, s_{it_i}$ between s_i and s_{i+1} (for $1 \leq i \leq n$). However, the propositions that hold in these states are the same as the ones that hold in s_i. In the definitions to come, all paths of the form corresponding to the same timed action path may be identified. Therefore we will loosely write $\pi = s_1 \xrightarrow{t_1,a_1} s_2 \ldots s_n \xrightarrow{t_n,a_n} s_{n+1}$. Note also that the timed action paths cover exactly the proper paths.

5.1 Interpretations induced by paths

First of all, we consider the case where state variables simply correspond to atomic propositions that may hold in the states of a frame. Next, we admit state variables to correspond alternatively to sequences of actions that may be performed from the states till time passes to the next time slice. This latter case must be considered to be more appropriate for signal inserted timed frames, because they exhibit the interplay between the performance of actions and the consequent visible state changes.

Atomic propositions as state variables In this case, we take the set $\mathbb{P}_{at}$ of atomic propositions as the set $\mathcal{SV}$ of state variables.

Let $\pi = s_1 \xrightarrow{t_1,a_1} s_2 \ldots s_n \xrightarrow{t_n,a_n} s_{n+1}$ be a proper path. Then the *time length* of π, written $\boldsymbol{\ell}(\pi)$, is defined by

$$\boldsymbol{\ell}(\pi) = \sum_{i=1}^{n} t_i$$

Let F be a frame and $\pi = s_1 \xrightarrow{t_1,a_1} s_2 \ldots s_n \xrightarrow{t_n,a_n} s_{n+1}$ be a proper path such that $\pi \in \boldsymbol{E}(F)$. Then

the set of atomic propositions that *hold for* π at time t, written $\boldsymbol{P}(F)(\pi,t)$, is defined by

$$\begin{aligned} p \in \boldsymbol{P}(F)(\pi,t) \;\Leftrightarrow\; & \\ \exists k \in \mathbb{N} \cdot k+1 \leq n \wedge \sum_{i=1}^{k} t_i \leq t < \sum_{i=1}^{k+1} t_i \wedge (p \frown s_{k+1}) \oplus F = F \vee & \\ t = \sum_{i=1}^{n} t_i \wedge (p \frown s_{n+1}) \oplus F = F & \end{aligned}$$

With immediate transitions, i.e. with $t_i = 0$ for some i $(1 \leq i \leq n)$, several actions seem to take place in sequence at the same discrete time point. Before proceeding, we have a look at the origin of this peculiarity. Timed frames are meant for modeling the time-dependent behaviour of programs at a level of abstraction where time is measured with finite precision by using a discrete time scale. It means that the discrete time points just divide real time into time slices and, although actions and state changes take place in real time, only the time slices in which actions and state changes take place are considered to be of importance. Discrete time process algebras such as ATP [26] and the discrete time extension of ACP presented in [5] offer exactly this abstraction. Naturally, it is in accordance with the intended meaning of a time step – the passage of time to the next time slice – for it derives this meaning from its use in these discrete time process algebras. However, the discrete time semantics of MVC does not offer a similar abstraction because it only allows for state changes at the discrete time points. For this reason, the sequence of actions taking place within a time slice must be treated as a single transition that yields only one state change.

We define the interpretation $\mathcal{I}_\pi^F$ over $[0,\boldsymbol{\ell}(\pi)]$ *induced by* a proper path π in frame F by

$$\mathcal{I}_\pi^F(v)(t) = \begin{cases} 1 & \text{if } v \in \boldsymbol{P}(F)(\pi,t) \\ 0 & \text{otherwise} \end{cases}$$

In this way, proper paths in a frame correspond to interpretations for MVC.

A formula Φ is *valid in* a frame F, written $F \models \Phi$, iff $\mathcal{I}_\pi^F, [0,\boldsymbol{\ell}(\pi)] \models \Phi$ for all proper paths π such that $\pi \in \boldsymbol{E}(F)$.

Sequences of actions as state variables Now we add the set A^+ of non-empty sequences of actions to the set $\mathcal{SV}$ of state variables.

Let F be a frame and $\pi = s_1 \xrightarrow{t_1,a_1} s_2 \ldots s_n \xrightarrow{t_n,a_n} s_{n+1}$ be a proper path such that $\pi \in \boldsymbol{E}(F)$. Then the set of sequences of actions that *happen in* π at time t, written $\boldsymbol{A}(F)(\pi,t)$, is defined by

$$\begin{aligned} a'_1 \ldots a'_m \in \boldsymbol{A}(F)(\pi,t) \;\Leftrightarrow\; & \\ \exists k \in \mathbb{N} \cdot k+m \leq n \wedge \sum_{i=1}^{k+1} t_i = t = \sum_{i=1}^{k+m} t_i \wedge & \\ (k \neq 0 \;\Rightarrow\; \sum_{i=1}^{k} t_i < t) \wedge (k+m \neq n \;\Rightarrow\; t < \sum_{i=1}^{k+m+1} t_i) \wedge \bigwedge_{j=1}^{m} (a_{k+j} = a'_j) & \end{aligned}$$

Note that the set $\boldsymbol{A}(F)(\pi,t)$ is either the empty set or a singleton set. In the former case, no sequence of actions happens at time t. In the latter case, t must be a discrete time point and the sequence of actions is the complete sequence of actions that happens at that time point. Only the complete sequence is considered to happen because only the state change corresponding to the complete sequence is visible.

We re-define the interpretation $\mathcal{I}_\pi^F$ over $[0,\boldsymbol{\ell}(\pi)]$ *induced by* a proper path π in frame F by

$$\mathcal{I}_\pi^F(v)(t) = \begin{cases} 1 & \text{if } v \in \boldsymbol{P}(F)(\pi,t) \text{ or } v \in \boldsymbol{A}(F)(\pi,t) \\ 0 & \text{otherwise} \end{cases}$$

5.2 Example
In Section 2.1, the control component of a telephone answering machine was modelled by a signal inserted timed frame. One of its properties is the following:

> The waiting-to-answer phase lasts for at most 10 time units.

This property is easy to express in MVC using both atomic propositions and sequences of actions as state variables:

$$\lceil r(\textit{incoming call})\rceil^0 ; \lceil \neg \textit{playing} \wedge \neg \textit{recording}\rceil \rightarrow \ell \leq 10$$

In Section 3.2, the following property was expressed in TFL:

> When the off-hook signal is issued to the telephone network, nothing has happened since the detection of the last incoming call and meanwhile 10 time units have passed.

This property can be expressed in MVC as well:

$$\begin{aligned}&\lceil r(\textit{incoming call})\rceil^0 ; \lceil \neg r(\textit{incoming call})\rceil ; \lceil s(\textit{off-hook})s(\textit{play msg})\rceil^0 \rightarrow \\ &\quad \ell = 10 \wedge \lceil r(\textit{incoming call})\rceil^0 ; \lceil \neg \bigvee_{e\in A^+} e\rceil ; \lceil s(\textit{off-hook})s(\textit{play msg})\rceil^0\end{aligned}$$

The formula $\lceil \neg \bigvee_{e\in A^+} e\rceil$ is used to characterize a non-point interval in which no actions happen.

5.3 Truth for paths in frames
We can also define the truth of a formula Φ for a proper path in a frame (instead of under its induced interpretation). Only the chop modality needs some special attention. We can not simply chop a proper path π in any two proper paths π_1 and π_2 for which $\pi = \pi_1 \bullet \pi_2$. Not all (proper) subpaths π' with $\boldsymbol{\ell}(\pi') = 0$ consist of a single state. However, in order to be in accordance with the interpretation induced by the path, such instant subpaths have to be treated in a way like single states. To accommodate this, we introduce the set of admissible *divisions* for a proper path π, written $\boldsymbol{D}(\pi)$. It is defined by

$$\begin{aligned}&(\pi_1,\pi_2) \in \boldsymbol{D}(\pi) \;\Leftrightarrow \\ &\quad \pi_1,\pi_2 \in \boldsymbol{\Pi_p} \wedge \\ &\quad \exists \pi_1',\pi',\pi_2' \in \Pi \cdot \\ &\qquad \pi_1 = \pi_1' \bullet \pi' \wedge \pi_2 = \pi' \bullet \pi_2' \wedge \pi = \pi_1' \bullet \pi' \bullet \pi_2' \wedge \pi' \in \boldsymbol{\Pi_p} \wedge \boldsymbol{\ell}(\pi') = 0 \wedge \\ &\qquad \neg(\exists \pi_1'',\pi'' \in \Pi \cdot \pi_1' \neq \pi_1'' \wedge \pi_1' = \pi_1'' \bullet \pi'' \wedge \pi'' \in \boldsymbol{\Pi_p} \wedge \boldsymbol{\ell}(\pi'') = 0) \wedge \\ &\qquad \neg(\exists \pi'',\pi_2'' \in \Pi \cdot \pi_2' \neq \pi_2'' \wedge \pi_2' = \pi'' \bullet \pi_2'' \wedge \pi'' \in \boldsymbol{\Pi_p} \wedge \boldsymbol{\ell}(\pi'') = 0)\end{aligned}$$

Suppose $\pi = s_1 \xrightarrow{t_1,a_1} s_2 \ldots s_n \xrightarrow{t_n,a_n} s_{n+1}$ and let $\pi' = s'_{i+1} \xrightarrow{0,a_{i+1}} s_{i+2} \ldots s_{i+m} \xrightarrow{0,a_{i+m}} s_{i+m+1}$ ($0 \leq i \leq n$, $1 \leq m \leq n-i$) be a subpath of π that can not be extended at either side to a longer subpath without time steps. The preceding definition makes precise the way in which admissible divisions (π_1,π_2) of π treat π' like a single state, viz. π_1 ends at s_{i+m+1} if and only if π_2 begins at s'_{i+1}, π_1 never ends at another state of π' and π_2 never begins at another state of π'.

Atomic propositions as state variables To begin with, we consider the case where state variables correspond to atomic propositions.

Let F be a frame and $\pi = s_1 \xrightarrow{t_1,a_1} s_2 \ldots s_n \xrightarrow{t_n,a_n} s_{n+1}$ be a proper path such that $\pi \in \boldsymbol{E}(F)$.

The value of a state expression P for path π in F is the function $[\![P]\!]^{(F,\pi)} : \mathbb{S} \rightarrow \{0,1\}$ inductively defined by

$$
\begin{aligned}
[\![0]\!]^{(F,\pi)}(s) &= 0,\\
[\![1]\!]^{(F,\pi)}(s) &= 1,\\
[\![v]\!]^{(F,\pi)}(s) &= \begin{cases} 1 & \text{if } (v \overset{\frown}{\rightarrow} s) \oplus F = F\\ 0 & \text{otherwise,}\end{cases}\\
[\![\neg P]\!]^{(F,\pi)}(s) &= 1 - [\![P]\!]^{(F,\pi)}(s),\\
[\![P \wedge Q]\!]^{(F,\pi)}(s) &= \begin{cases} 1 & \text{if } [\![P]\!]^{(F,\pi)}(s) = 1 \text{ and } [\![Q]\!]^{(F,\pi)}(s) = 1\\ 0 & \text{otherwise.}\end{cases}
\end{aligned}
$$

The value of a term r for path π in F under assignment α is the value $[\![r]\!]_\alpha^{(F,\pi)} : \mathbb{R}$ inductively defined by

$$
\begin{aligned}
[\![\ell]\!]_\alpha^{(F,\pi)} &= \boldsymbol{\ell}(\pi),\\
[\![x]\!]_\alpha^{(F,\pi)} &= \alpha(x),\\
[\![\overline{P}]\!]_\alpha^{(F,\pi)} &= \begin{cases} (\sum_{i=1}^{n} [\![P]\!]^{(F,\pi)}(s_i) \times t_i)/\boldsymbol{\ell}(\pi) & \text{if } \boldsymbol{\ell}(\pi) > 0\\ [\![P]\!]^{(F,\pi)}(s_{n+1}) & \text{otherwise,}\end{cases}\\
[\![f(r_1,\ldots,r_n)]\!]_\alpha^{(F,\pi)} &= \boldsymbol{f}([\![r_1]\!]_\alpha^{(F,\pi)},\ldots,[\![r_n]\!]_\alpha^{(F,\pi)}).
\end{aligned}
$$

The truth of a formula Φ for path π in F under assignment α, written $F,\pi \models_\alpha \Phi$, is inductively defined by

$$
\begin{aligned}
F,\pi &\models_\alpha \mathtt{t},\\
F,\pi &\models_\alpha r = r' &&\Leftrightarrow\quad [\![r]\!]_\alpha^{(F,\pi)} = [\![r']\!]_\alpha^{(F,\pi)},\\
F,\pi &\models_\alpha p(r_1,\ldots,r_n) &&\Leftrightarrow\quad ([\![r_1]\!]_\alpha^{(F,\pi)},\ldots,[\![r_n]\!]_\alpha^{(F,\pi)}) \in \boldsymbol{p},\\
F,\pi &\models_\alpha \neg\Phi &&\Leftrightarrow\quad \text{not } F,\pi \models_\alpha \Phi,\\
F,\pi &\models_\alpha \Phi \wedge \Psi &&\Leftrightarrow\quad F,\pi \models_\alpha \Phi \text{ and } F,\pi \models_\alpha \Psi,\\
F,\pi &\models_\alpha \Phi\,;\Psi &&\Leftrightarrow\quad \text{for some } (\pi_1,\pi_2) \in \boldsymbol{D}(\pi),\\
&&&\qquad F,\pi_1 \models_\alpha \Phi \text{ and } F,\pi_2 \models_\alpha \Psi,\\
F,\pi &\models_\alpha \forall x \cdot \Phi &&\Leftrightarrow\quad \text{for all } d \in \mathbb{R},\ F,\pi \models_{\alpha(x\to d)} \Phi.
\end{aligned}
$$

We write $F,\pi \models \Phi$ to indicate that $F,\pi \models_\alpha \Phi$ for all assignments α.

The following result relates the truth of formulae for paths with the interpretations induced by paths.

Lemma 1. *Truth for a path and truth under the interpretation induced by the path are equivalent:*

$$\textit{for all paths } \pi \in \boldsymbol{\Pi_p},\ \ F,\pi \models_\alpha \Phi \ \Leftrightarrow\ \mathcal{I}_\pi^F, [0,\boldsymbol{\ell}(\pi)] \models_\alpha \Phi$$

Proof. Let $\pi = s_1 \xrightarrow{t_1,a_1} s_2 \ldots s_n \xrightarrow{t_n,a_n} s_{n+1}$. It follows immediately from the definition of $\mathcal{I}_\pi^F$ that $[\![v]\!]^{(F,\pi)}(s_{k+1}) = [\![v]\!]^{\mathcal{I}_\pi^F}(t)$ if $\sum_{i=1}^{k} t_i \le t < \sum_{i=1}^{k+1} t_i$, for $k < n$; and $[\![v]\!]^{(F,\pi)}(s_{n+1}) = [\![v]\!]^{\mathcal{I}_\pi^F}(t)$ if $t = \sum_{i=1}^{n} t_i$. It is easy to show by induction on the construction of state expressions that this extends from state variables to state expressions. Hence we obtain by induction on the construction of terms $[\![r]\!]_\alpha^{(F,\pi)} = [\![r]\!]_\alpha^{\mathcal{I}_\pi^F}([0,\boldsymbol{\ell}(\pi)])$. From this it follows by induction on the construction of formulae that $F,\pi \models_\alpha \Phi \ \Leftrightarrow\ \mathcal{I}_\pi^F, [0,\boldsymbol{\ell}(\pi)] \models_\alpha \Phi$. Only the case of formulae of the form $\Phi\,;\Psi$ is not trivial. It requires to show that paths π_1, π_2 and π such that $(\pi_1,\pi_2) \in \boldsymbol{D}(\pi)$ determine uniquely an $m \in \mathbb{N}$, where $m \in [0,\boldsymbol{\ell}(\pi)]$, such that $\mathcal{I}_{\pi_1}^F, [0,\boldsymbol{\ell}(\pi_1)] \models_\alpha \Phi \ \Leftrightarrow\ \mathcal{I}_\pi^F, [0,m] \models_\alpha \Phi$ and $\mathcal{I}_{\pi_2}^F, [0,\boldsymbol{\ell}(\pi_2)] \models_\alpha \Phi$

$\Leftrightarrow \mathcal{I}^F_\pi, [m, \boldsymbol{\ell}(\pi)] \models_\alpha \Phi$. From the definition of truth under an interpretation it readily follows by induction on the construction of formulae that $\mathcal{I}, [b, e] \models_\alpha \Phi \Leftrightarrow \mathcal{I}^{[b,e]}, [0, e-b] \models_\alpha \Phi$ where $\mathcal{I}^{[b,e]}$ is the interpretation over $[0, e-b]$ defined by $\mathcal{I}^{[b,e]}(v)(t) = \mathcal{I}(v)(t-b)$. So it suffice to show that π_1, π_2 and π determine uniquely an m such that $\mathcal{I}^F_{\pi_1} = (\mathcal{I}^F_\pi)^{[0,m]}$ and $\mathcal{I}^F_{\pi_2} = (\mathcal{I}^F_\pi)^{[m,\boldsymbol{\ell}(\pi)]}$. This follows rather directly from the definitions of $\mathcal{I}^F_\pi$ and $\boldsymbol{D}(\pi)$. □

Corollary. *The validity of a formula Φ in a frame F can be characterized by*

$$F \models \Phi \Leftrightarrow \textit{for all paths } \pi \in \boldsymbol{\Pi_p} \textit{ such that } \pi \in \boldsymbol{E}(F),\ F, \pi \models \Phi$$

Sequences of actions as state variables The definitions given above for the case where only atomic propositions are taken as state variables are standard with the exception of the clauses concerning the distinctive features of MVC: state variables, interval-lenghts, mean values and the chop modality. With respect to paths in frames, their meaning turns out to be quite natural. The possible presence of immediate transitions in paths is largely responsible for the small complication with the chop modality.

If we take sequences of actions as state variables as well, we get the following additional clause in the definition of the value of state expressions:

$$[\![a'_1 \dots a'_m]\!]^{(F,\pi)}(s) \quad = \quad \begin{cases} 1 & \text{if } \boldsymbol{\ell}(\pi) = 0, s = s_{n+1} \text{ and } a'_1 \dots a'_m = a_1 \dots a_n \\ 0 & \text{otherwise} \end{cases}$$

It is questionable whether this counts for natural. The possible presence of immediate transitions in paths is largely responsible here as well. Lemma 1 goes through for this case.

6. From duration calculus to timed frame logic

In this section, we discuss the connection between MVC and TFL. First, we touch upon the impossibility of embedding MVC into TFL. Thereafter, we show that an interesting fragment of MVC can be embedded. An embedding of (a fragment of) MVC into TFL is a mapping that translates the formulae of (the fragment of) MVC to TFL formulae such that validity in a frame, as defined in Section 5.1, remains the same after translation. In Section 5, we presented the truth of MVC formulae in timed frames in two ways: (1) by starting from the original discrete time semantics of duration calculus and (2) by introducing a new semantics directed at timed frames. In this section, we present it in a third way for a fragment of MVC: by giving a translation to TFL that leaves validity unchanged. Therefore this presentation is equivalent to the first and second one for the fragment concerned – a propositional fragment which allows only the use of integrals and point values instead of the unrestricted use of mean values.

6.1 Embedding

The angle from which we consider the truth of MVC formulae in timed frames in this section focusses on the embedding of an interesting fragment of MVC into TFL. The resulting presentation of the truth of formulae from this fragment in timed frames can be safely used whenever restriction to the fragment suffices. The use of this is further discussed in Section 1. Besides, it is considered useful to gain a better understanding of how MVC and a logic designed for expressing and verifying properties of timed frames relate to each other. Notice that the current angle is rather different from the ones of Section 5. Both previous angles led to main definitions referring to timed frames, whereas the current angle involves an a priori choice not to have main definitions referring to timed frames. Even so, the characterization of validity of MVC formulae in timed frames, as given in Section 5.3, is deemed to facilitate devising an embedding of MVC into TFL.

The embedding of MVC into TFL will immediately fail because TFL does not support real numbers. Let us therefore just assume that the sort $\mathbb{R}$ of real numbers, and sufficient standard functions and predicates concerning real numbers, have been added to TFL. We find that all standard predicates of

TFL are necessary. The path existence predicate E is needed for the implicit universal quantification over all paths in a frame. The proposition presence predicate H is needed to represent state variables by atomic propositions. The state and transition label selection predicates $\mathsf{S_s}$ and $\mathsf{S_l}$ are needed to model the chop modality. However, the standard predicates of TFL are not sufficient. In order to deal with interval-lenghts and mean values, more is needed. Obviously, support for recursion – e.g. countably infinite disjunction or a fixpoint operator – will do.

At first sight, it seems that TFL lacks expressive power. However, in Section 3.3 we pointed out the fact, proved in [8], that any timed frame can be distinguished from another one. So at least any finite set of frames is definable in TFL. This is not the case in MVC, because its distinctive power with respect to frames is less. This follows immediately from the fact that certain state changes are considered to be invisible. In this connection, recall that the interpretation induced by a path in a frame, as defined in Section 5.1, is independent of the intermediate states in each timed action step of the path.

In retrospect, it is not very surprising that MVC and TFL are not more closely related. MVC was designed to be a logic for specifying and reasoning about real-time requirements for systems. TFL was designed to be a logic for expressing and verifying properties of objects that are meant to model programs with timing constraints – derived from the real-time requirements for the system in which the program concerned is embedded. In consequence, MVC has been geared to properties about the duration of phases of system behaviour – which may comprise many states and state changes – within a given time interval, while TFL is more suited for properties about the time points at which program actions – which yield a single state change – are performed. In other words, these logics are originally meant to be used for quite different purposes. For instance, MVC is not intended to be used for describing that certain actions must be performed cyclically, whereas TFL is not intended to be used for specifying bounds on the duration of a certain state over time periods exceeding some minimal length. Certainly, this does not mean that there are no tricks to use either logic to a certain extent for the purpose the other is meant to be used for.

6.2 Fragments

Although MVC and TFL are meant to be used for different purposes, it is still useful to investigate whether there exist fragments of MVC that can be embedded into TFL. The latter logic is more suitable than MVC to express and verify properties of frames and has more distinctive power with respect to frames. Besides, it will presumably be refinements of real-time requirements for which it is interesting to verify whether they are met by frames. These refinements will be formulated in a fragment of MVC anyhow, e.g. the fragment consisting of the implementables introduced in [30].

Identifying a fragment of MVC that can be embedded into TFL is not too difficult if one realizes that: (1) with the discrete time semantics the value of terms of the form $\overline{P} * \ell$ (i.e. $\int P$) is always in $\mathbb{N}$, and (2) the main reason for replacing integrals ($\int P$) by mean values ($\overline{P}$) in MVC was to add the possibility to deal with point values ($\lceil P \rceil^0$).

In the fragment that we have in view, the terms and formulae are restricted with respect to the occurrences of terms of the form ℓ or $\overline{P}$ such that integrals and point values are covered. Further restrictions on the function symbols ensure that the value of all terms is always in $\mathbb{N}$. However, since TFL has no support for recursion, more restrictions on the (atomic) formulae are needed – mainly with respect to the occurrences of logical variables and terms in which state expressions occur. For example, TFL can not deal with formulae of the form $\exists x \cdot \overline{P} * \ell = x$, not even if the range of x is restricted to $\mathbb{N}$, or $\overline{P} * \ell = \overline{Q}$. These restrictions make quantification as well as terms other than constants for natural numbers useless, and thus they result in a simple propositional fragment of MVC. Nevertheless, this fragment is powerful enough to represent all forms of implementables used in [30]. Besides, the fragment embedded into TFL here is closely related to the fragment of DC for which model-checking is covered in [17].

We assume a constant for each natural number. We shall use the meta-variable k to stand for an arbitrary such a term. The formulae of the fragment are inductively defined by the following formation rules:

1. $\mathtt{t}$ is a formula;
2. if P is a state expression, then $\ell = 0 \wedge \overline{P} = 1$ is a formula;
3. if P is a state expression and k a constant, then $\overline{P} * \ell = k$ is a formula;
4. if Φ is a formula, then $\neg\Phi$ is a formula;
5. if Φ, Ψ are formulae, then $\Phi \wedge \Psi$ and $\Phi \,;\, \Psi$ are formulae.

Note that we introduced in Section 4 the abbreviations $\int P$ and $\lceil P \rceil^0$ for $\overline{P} * \ell$ and $\ell = 0 \wedge \overline{P} = 1$, respectively. We shall use these abbreviations from now on. Note further that we can represent $\int P \geq k$ and $\ell \geq k$ by $\int P = k \,;\, \mathtt{t}$ and $\int 1 = k \,;\, \mathtt{t}$, respectively. It is easy to check that all forms of implementables from [30] can be represented as well.

In the definition of the translation, we write:

$$nrtr(\pi, n) \quad \text{for} \quad (\exists s \in \mathbb{S} \cdot \mathsf{S_s}(n, \pi, s)) \wedge \neg(\exists s \in \mathbb{S} \cdot \mathsf{S_s}(n{+}1, \pi, s))$$

$$proper(\pi) \quad \text{for} \quad \left\{ \begin{array}{l} \neg nrtr(\pi, 0) \rightarrow \\ \exists n \in \mathbb{N} \cdot nrtr(\pi, n{+}1,) \wedge \exists \mu \in \mathbb{L} \cdot \mathsf{S_l}(n, \pi, \mu) \wedge \mu \neq \sigma \end{array} \right.$$

$$com(\pi_1, \pi_2, \pi) \quad \text{for} \quad \left\{ \begin{array}{l} \exists m \in \mathbb{N} \cdot nrtr(\pi_1, m) \wedge \forall n \in \mathbb{N} \cdot \\ \quad (n \leq m \rightarrow \forall s \in \mathbb{S} \cdot \mathsf{S_s}(n, \pi_1, s) \leftrightarrow \mathsf{S_s}(n, \pi, s)) \wedge \\ \quad (n < m \rightarrow \forall \mu \in \mathbb{L} \cdot \mathsf{S_l}(n, \pi_1, \mu) \leftrightarrow \mathsf{S_l}(n, \pi, \mu)) \wedge \\ \quad (\forall s \in \mathbb{S} \cdot \mathsf{S_s}(n, \pi_2, s) \leftrightarrow \mathsf{S_s}(m{+}n, \pi, s)) \wedge \\ \quad (\forall \mu \in \mathbb{L} \cdot \mathsf{S_l}(n, \pi_2, \mu) \leftrightarrow \mathsf{S_l}(m{+}n, \pi, \mu)) \end{array} \right.$$

$$div(\pi, \pi_1, \pi_2) \quad \text{for} \quad \left\{ \begin{array}{l} proper(\pi) \wedge proper(\pi_1) \wedge proper(\pi_2) \wedge \\ \exists \pi_1' \in \Pi \cdot \exists \pi' \in \Pi \cdot \exists \pi_2' \in \Pi \cdot \\ \quad com(\pi_1', \pi', \pi_1) \wedge com(\pi', \pi_2', \pi_2) \wedge com(\pi_1, \pi_2', \pi) \wedge \\ \quad proper(\pi') \wedge \neg(\exists n \in \mathbb{N} \cdot \mathsf{S_l}(n, \pi', \sigma)) \wedge \\ \quad \neg(\exists \pi_1'' \in \Pi \cdot \exists \pi'' \in \Pi \cdot \pi_1' \neq \pi_1'' \wedge com(\pi_1'', \pi'', \pi_1') \wedge \\ \qquad\qquad proper(\pi'') \wedge \neg(\exists n \in \mathbb{N} \cdot \mathsf{S_l}(n, \pi'', \sigma))) \wedge \\ \quad \neg(\exists \pi'' \in \Pi \cdot \exists \pi_2'' \in \Pi \cdot \pi_2' \neq \pi_2'' \wedge com(\pi'', \pi_2'', \pi_2') \wedge \\ \qquad\qquad proper(\pi'') \wedge \neg(\exists n \in \mathbb{N} \cdot \mathsf{S_l}(n, \pi'', \sigma))) \end{array} \right.$$

These abbreviations can informally be explained as follows: $nrtr(\pi, n)$ is true iff there are n transitions in π, $proper(\pi)$ is true iff π is a proper path, $com(\pi_1, \pi_2, \pi)$ is true iff π is the path composition of π_1 and π_2, and $div(\pi, \pi_1, \pi_2)$ is true iff (π_1, π_2) is an admissible division of π. The abbreviation $com(\pi_1, \pi_2, \pi)$ is only used to define $div(\pi, \pi_1, \pi_2)$. The abbreviation $div(\pi, \pi_1, \pi_2)$ has been chosen to stand for a formula that resembles the definition of $\boldsymbol{D}(\pi)$ in Section 5.3 strongly.

The translation of a MVC formula Φ from the fragment is the TFL formula $\forall \pi \in \Pi \cdot proper(\pi) \wedge \mathsf{E}(\pi) \rightarrow (\!|\Phi|\!)$, where $(\!|\Phi|\!)$ is inductively defined by

$$
\begin{aligned}
(\!|\mathsf{t}|\!) &= \mathsf{t},\\
(\!|\lceil P \rceil^0|\!) &= \begin{cases} \neg(\exists n' \in \mathbb{N} \cdot \mathsf{S_I}(n', \pi, \sigma)) \wedge \\ \exists n \in \mathbb{N} \cdot \exists s \in \mathbb{S} \cdot \\ \quad \mathsf{S_s}(n, \pi, s) \wedge \mathit{nrtr}(\pi, n) \wedge (\!|P|\!), \end{cases}\\
(\!|\textstyle\int P = k|\!) &= \begin{cases} \exists n_1 \in \mathbb{N} \cdot \ldots \exists n_k \in \mathbb{N} \cdot \\ \quad \bigwedge_{i=1}^{k} ((\bigwedge_{j=1}^{i-1} n_i \neq n_j) \wedge \exists n \in \mathbb{N} \cdot \exists s \in \mathbb{S} \cdot \\ \quad\quad \mathsf{S_s}(n, \pi, s) \wedge \mathsf{S_I}(n, \pi, \sigma) \wedge n = n_i \wedge (\!|P|\!)) \wedge \\ \quad \neg \exists n_{k+1} \in \mathbb{N} \cdot \\ \quad\quad (\bigwedge_{j=1}^{k} n_{k+1} \neq n_j) \wedge \exists n \in \mathbb{N} \cdot \exists s \in \mathbb{S} \cdot \\ \quad\quad\quad \mathsf{S_s}(n, \pi, s) \wedge \mathsf{S_I}(n, \pi, \sigma) \wedge n = n_{k+1} \wedge (\!|P|\!), \end{cases}\\
(\!|\neg\Phi|\!) &= \neg(\!|\Phi|\!),\\
(\!|\Phi \wedge \Psi|\!) &= (\!|\Phi|\!) \wedge (\!|\Psi|\!),\\
(\!|\Phi\,;\Psi|\!) &= \begin{cases} \exists \pi_1 \in \Pi \cdot \exists \pi_2 \in \Pi \cdot \mathit{div}(\pi, \pi_1, \pi_2) \wedge \\ \quad (\exists \pi \in \Pi \cdot \pi = \pi_1 \wedge (\!|\Phi|\!)) \wedge (\exists \pi \in \Pi \cdot \pi = \pi_2 \wedge (\!|\Psi|\!)). \end{cases}
\end{aligned}
$$

For state expressions P, the TFL formula $(\!|P|\!)$ is inductively defined by

$$
\begin{aligned}
(\!|0|\!) &= \mathsf{H}(\mathsf{f}, s),\\
(\!|1|\!) &= \mathsf{H}(\mathsf{t}, s),\\
(\!|p|\!) &= \mathsf{H}(p, s),\\
(\!|a_1 \ldots a_m|\!) &= \mathit{nrtr}(\pi, n) \wedge n = m \wedge \textstyle\bigwedge_{i=0}^{m-1} \mathsf{S_I}(i, \pi, a_{i+1}),\\
(\!|\neg P|\!) &= \neg(\!|P|\!),\\
(\!|P \wedge Q|\!) &= (\!|P|\!) \wedge (\!|Q|\!).
\end{aligned}
$$

The translation appears to be rather intricate. This is mainly due to the use of predicates in TFL to represent partial functions for the selection of states and transition labels from paths. Notice that the translation of $\int P = k$ can be paraphrased as follows: path π has exactly k different states with an outgoing time step where P holds.

The following result shows that the translation from MVC formulae to TFL formulae is an embedding.

Lemma 2. *Validity remains the same after translation:*

$$F \models \Phi \;\Leftrightarrow\; F \models \forall \pi \in \Pi \cdot \mathit{proper}(\pi) \wedge \mathsf{E}(\pi) \rightarrow (\!|\Phi|\!)$$

Proof. We take the characterization of $F \models \Phi$ in the corollary of Lemma 1 as its definition. It is straightforward to derive from the clauses for the truth of formulae with respect to paths in frames special clauses for formulae of the forms $\lceil P \rceil^0$ and $\int P = k$. Hence the proof proceeds by induction on the construction of state expressions and formulae of the fragment. Again the case of formulae of the form $\Phi\,;\Psi$ is relatively hard. In order to verify that the TFL formula $\mathit{div}(\pi, \pi_1, \pi_2)$ expresses that $(\pi_1, \pi_2) \in \boldsymbol{D}(\pi)$, it has to be checked whether $\mathit{com}(\pi_1, \pi_2, \pi)$ expresses that $\pi_1 \bullet \pi_2 = \pi$, i.e. $\pi_1 \bullet \pi_2 = \pi \;\Leftrightarrow\; \models \mathit{com}(\pi_1, \pi_2, \pi)$. This follows from the definition of $\bullet$ by induction on the construction of paths. □

7. Closing remarks

In Section 5.1, we defined the truth of MVC formulae in timed frames by defining how to extract interpretations of state variables from paths in timed frames and then, using the standard discrete time semantics of MVC, how to establish validity of MVC formulae in timed frames. In Section 5.3, we characterized the truth of MVC formulae in timed frames by introducing a new semantics for MVC that describes the meaning of MVC terms and formulae with respect to paths in timed frames instead of interpretations of state variables. Because the presentations are equivalent, either presentation can be safely used as the primary one at different times. In Section 6, we found that only fragments of MVC can be embedded into TFL and we worked out the embedding of an interesting fragment. Thus, we presented the truth of formulae from this fragment in timed frames in still another, indirect way. In the rest of this section, we discuss some points which were raised by the material in Sections 5 and 6, but for which space could not be found there.

Timed frames are meant for modeling the time-dependent behaviour of programs at a level of abstraction where time is measured with finite precision by using a discrete time scale. The discrete time semantics of MVC does not offer such an abstraction because it only allows for state changes at discrete time points. Therefore, the sequence of actions taking place within a time slice had to be treated in Section 5 as a single transition that yields only one state change. However, this leads to the peculiarity that several actions seem to take place in sequence at the same time point. Ongoing work on duration calculus aims to deal with such cases, both in discrete and continuous time, in a more satisfactory way by introducing a micro time (see e.g. [28, 35]).

In [7], results concerning the connection between timed frames and discrete time processes are given in the setting of the discrete time extension of ACP presented in [5]. This extension has already been used as the basis of the semantics of some languages related to programming, e.g. the language T of the ToolBus software interconnection architecture [9] and a semantically clear subset of SDL [10]; and it is envisaged to use it for the semantics of other languages related to programming as well, e.g. a tractable subset of a widely used modern programming language such as C++ [33] or Java [2]. A useful topic for further work is the lifting of the results from Section 5 concerning the connection of duration calculus with timed frames to processes as studied in the setting of the discrete time extension of ACP. This should be relatively easy using the above-mentioned results from [7]. After that, semantic links of MVC with languages with a semantics based on this process algebra can be established virtually without further effort.

Embedding Dicky logic [1] or Hennessy-Milner logic [20] into TFL is a considerably easier job than embedding an interesting fragment of MVC into TFL. This is also the case for the extension of Hennessy-Milner logic where states have propositions assigned to them and where a special label, standing for a time step, is added to the set of transition labels. Because of the simplicity of their embedding into TFL, we consider these logics to be closely related to TFL. The use of Hennessy-Milner logic as a basic specification language for reactive systems has been illustrated in, for example, [24]. The extension outlined above allows for specifying time constraints. Therefore, it may well be used as specification language in a ProCoS-like approach. The results from Section 6 concerning the embedding of MVC into TFL facilitate devising such an approach, for a semantic link between MVC and the extension of Hennessy-Milner logic can now be established by just giving the simple embedding of the extension of Hennessy-Milner logic into TFL .

References

1. A. Arnold. *Finite Transition Systems*. Prentice-Hall, 1994.
2. K. Arnold and J. Gosling. *The Java Programming Language*. Addison-Wesley, 1996.
3. J.C.M. Baeten and J.A. Bergstra. Real time process algebra. *Formal Aspects of Computing*, 3:142–188, 1991.
4. J.C.M. Baeten and J.A. Bergstra. Discrete time process algebra (extended abstract). In W.R. Cleaveland, editor, *CONCUR'92*, pages 401–420. LNCS 630, Springer-Verlag, 1992. Full version:

Report P9208b, Programming Research Group, University of Amsterdam.

5. J.C.M. Baeten and J.A. Bergstra. Discrete time process algebra. *Formal Aspects of Computing*, 8:188–208, 1996.
6. J.C.M. Baeten and J.A. Bergstra. Discrete time process algebra: Absolute time, relative time and parametric time. *Fundamenta Informaticae*, 29:51–76, 1997.
7. J.A. Bergstra, W.J. Fokkink, and C.A. Middelburg. Algebra of timed frames. *International Journal of Computer Mathematics*, 61:227–255, 1996.
8. J.A. Bergstra, W.J. Fokkink, and C.A. Middelburg. A logic for signal inserted timed frames. Logic Group Preprint Series 155, Utrecht University, Department of Philosophy, January 1996.
9. J.A. Bergstra and P. Klint. The discrete time ToolBuS – A software coordination architecture. *Science of Computer Programming*, 31:205–229, 1998.
10. J.A. Bergstra, C.A. Middelburg, and Y.S. Usenko. Discrete time process algebra and the semantics of SDL. CWI Report SEN-R9809, Centre for Mathematics and Computer Science, June 1998. To appear in J.A. Bergstra, A. Ponse and S.A. Smolka, editors, Handbook of Process Algebra, Elsevier.
11. J.A. Bergstra and A. Ponse. Frame algebra with synchronous communication. In R.J. Wieringa and R.B. Feenstra, editors, *Information Systems – Correctness and Reusability*, pages 3–15. World Scientific, 1995.
12. L. Chen. An interleaving model for real-time systems. In A. Nerode and M. Taitslin, editors, *Symposium on Logical Foundations of Computer Science*, pages 81–92. LNCS 620, Springer-Verlag, 1992.
13. Dang Van Hung and Phan Hong Giang. A sampling semantics of duration calculus. In B. Jonsson and J. Parrow, editors, *Formal Techniques for Real-Time and Fault Tolerant Systems*, pages 188–207. LNCS 1135, Springer-Verlag, 1996.
14. J. Ellsberger, D. Hogrefe, and A. Sarma. *SDL, Formal Object-oriented Language for Communicating Systems*. Prentice-Hall, 1997.
15. W.J. Fokkink and A.S. Klusener. An effective axiomatization for real time ACP. *Information and Computation*, 122:286–299, 1995.
16. M. Fränzle and B. von Karger. Proposal for a programming language core for ProCoS II. ProCoS II document [Kiel MF 11/3], Christian-Albrechts-Universität Kiel, 1993.
17. M.R. Hansen. Model checking discrete duration calculus. *Formal Aspects of Computing*, 6A:826–845, 1994.
18. M.R. Hansen and Zhou Chaochen. Duration calculus: Logical foundations. *Formal Aspects of Computing*, 9:283–330, 1997.
19. He Jifeng, C.A.R. Hoare, M. Fränzle, M. Müller-Olm, E.-R. Olderog, M. Schenke, M.R. Hansen, A.P. Ravn, and H. Rischel. Provably correct systems. In H. Langmaack, W.-P. de Roever, and J. Vytopil, editors, *Formal Techniques for Real-Time and Fault Tolerant Systems*, pages 288–335. LNCS 863, Springer-Verlag, 1994.
20. M. Hennessy and R. Milner. Algebraic laws for non-determinism and concurrency. *Journal of the ACM*, 32:137–161, 1985.
21. Z. Manna and A. Pnueli. Models of reactivity. *Acta Informatica*, 30:609–678, 1993.
22. S. Mauw. Example specifications in φSDL. Computing Science Report 96-04, Eindhoven University of Technology, Department of Mathematics and Computing Science, 1996.
23. C.A. Middelburg. Truth of duration calculus formulae in timed frames. Research Report 82, United Nations University, International Institute for Software Technology, September 1996.

24. R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.
25. M. Müller-Olm. A new proposal for TimedPL's semantics. ProCoS II document [Kiel MMO 10/1], Christian-Albrechts-Universität Kiel, 1994.
26. X. Nicollin and J. Sifakis. The algebra of timed processes ATP: Theory and application. *Information and Computation*, 114:131–178, 1994.
27. E.-R. Olderog and C.A.R. Hoare. Specification-oriented semantics for communicating processes. *Acta Informatica*, 23:9–66, 1986.
28. P.K. Pandya and Dang Van Hung. Duration calculus of weakly monotonic time. Research Report 122, United Nations University, International Institute for Software Technology, October 1997. To appear in Proceedings of Formal Techniques for Real-Time and Fault Tolerant Systems, 1998.
29. RAISE Language Group. *The RAISE Specification Language*. Prentice-Hall, 1992.
30. A.P. Ravn. *Design of Embedded Real-Time Computing Systems*. PhD thesis, Technical University of Denmark, 1995.
31. M. Schenke. A timed specification language for concurrent reactive systems. In D.J. Andrews, J.F. Groote, and C.A. Middelburg, editors, *Semantics of Specification Languages*, pages 152–167. Workshop in Computing Series, Springer-Verlag, 1994.
32. M. Schenke. *Development of Correct Real-Time Systems by Refinement*. PhD thesis, Fachbereich Informatik, Universität Oldenburg, 1997.
33. B. Stroustrup. *The C++ Programming Language*. Addison-Wesley, 1986.
34. Xia Yong. DC/RJ$^-$: A justification assistant for duration calculus. Research Report 126, United Nations University, International Institute for Software Technology, November 1997.
35. Zhou Chaochen and M.R. Hansen. Chopping a point. In He Jifeng, J. Cooke, and P. Wallis, editors, *BCS-FACS 7th Refinement Workshop*. Electronic Workshop in Computing Series, Springer-Verlag, 1996.
36. Zhou Chaochen, C.A.R. Hoare, and A.P. Ravn. A calculus of durations. *Information Processing Letters*, 40:269–276, 1991.
37. Zhou Chaochen and Li Xiaoshan. A mean value calculus of durations. In A.W. Roscoe, editor, *A Classical Mind: Essays in Honour of C.A.R. Hoare*, pages 431–451. Prentice-Hall, 1994.
38. Zhou Chaochen, A.P. Ravn, and M.R. Hansen. An extended duration calculus for hybrid real-time systems. In R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors, *Hybrid Systems*, pages 36–59. LNCS 736, Springer-Verlag, 1993.