

**SOFTWARE VERIFICATION RESEARCH CENTRE
SCHOOL OF INFORMATION TECHNOLOGY
THE UNIVERSITY OF QUEENSLAND**

**Queensland 4072
Australia**

TECHNICAL REPORT

No. 98-7

**A Set-Theoretic Model for Real-Time
Specification and Reasoning**

**C. J. Fidge I. J. Hayes
A. P. Martin A. K. Wabenhorst**

Version 1.3, March 1999

Phone: +61 7 3365 1003

Fax: +61 7 3365 1533

<http://svrc.it.uq.edu.au>

A slightly different version of this paper was published in J. Jeuring (ed.), *Mathematics of Program Construction '98*, Springer-Verlag Lecture Notes in Computer Science 1422, pp. 188–206, 1998.

© Copyright 1998 Springer-Verlag.

Note: Most SVRC technical reports are available via anonymous ftp, from `svrc.it.uq.edu.au` in the directory `/pub/techreports`. Abstracts and compressed postscript files are available via `http://svrc.it.uq.edu.au`

A Set-Theoretic Model for Real-Time Specification and Reasoning

C. J. Fidge I. J. Hayes A. P. Martin A. K. Wabenhorst

Abstract.

Timed-trace formalisms have emerged as a powerful method for specifying and reasoning about concurrent real-time systems. We present a simple variant which builds methodically on set theory, and is thus suitable for use by programmers with little formal methods experience.

1 Introduction

Following an intensive period of research, formal methods for modelling real-time systems are now starting to mature. One of the most successful approaches has been to model real-time systems via time-varying functions—this accords with the way dynamic behaviour of time-dependent processes is modelled in the physical sciences. Prominent examples include the *Duration Calculus* [20], the *Temporal Agent Model* [15], and the *timed refinement calculus* [9]. Despite many superficial differences, these languages offer similar capabilities and collectively form part of the family of ‘timed trace’ formalisms.

Our overall goal is to address the dual challenge of making formal methods accessible to practising computer programmers, and of devising practical tools to support the application of formal methods in software engineering. Although successful in their own right, the timed-trace formalisms are intimidating for programmers who have little formal methods experience. Furthermore, they require unconventional reasoning methods, so it is not immediately clear how to upgrade existing tools to support them.

In this paper we present a timed-trace formalism that builds methodically on set theory. It borrows the best features of the above-named formalisms, yet introduces as few new operators as possible. The approach is compatible with the popular Z specification notation [17] so that programmers already familiar with Z will be able to learn it quickly, and so that tools that support Z-like languages can be upgraded to support the notation. (The timed-trace formalisms already do have Z-based definitions [4, 5, 1] and the Duration Calculus’s reasoning methods have been implemented in a theorem prover [16], but the outcomes are discouragingly complex.)

The timed-trace formalisms each consist of two parts, a mathematical notation for specifying and reasoning about requirements on timed traces [6, 5],

and a refinement formalism for structuring specifications and developing designs and programs [8, 15]. Here we consider the former only. Section 2 defines our notation, Section 3 summarises some of its laws, and Section 4 presents a small example.

2 A Mathematical Notation for Timed-Trace Predicates

In earlier work, Millerchip, Mahony and Hayes [11] defined a simple set-theoretic notation for concisely expressing properties of time intervals and used it to undertake a substantial case study. Duddy et al. then expanded this definition with operators for accessing interval endpoints [5]. Elsewhere, the Duration Calculus [6], Temporal Agent Model [15], and their predecessors [12], showed how a ‘chop’ (concatenate) operator can form the basis of an effective real-time modelling and reasoning capability. Here we build on these previous methods by combining Millerchip et al.’s interval and concatenation definitions with a simple way to access interval endpoints. We use \mathbb{Z} -like mathematical notation throughout.

2.1 Time

Let the absolute time domain \mathbb{T} be continuous, as modelled by the real numbers \mathbb{R} [7, 20]. (We assume the availability of real numbers in \mathbb{Z} [19].)

Definition 1 (Time)

$$\mathbb{T} == \mathbb{R}$$

□

A common alternative is to use the non-negative reals \mathbb{R}_+ , with time 0 taken to be the moment at which we start observing the system [13]. The definitions below work equally well with this type.

Another alternative is to assume a discrete time, represented by the integers or natural numbers. This is often suggested for digital systems [6]. However we note that such systems are merely a special case. They can be modelled as continuous systems, with a real-valued time domain, in which values change only at whole multiples of some *sampling period* [3].

2.2 Time Intervals

Time intervals are represented as the set of all times between some infimum a and supremum z [11].

Definition 2 (Time intervals) *The left-open, right-closed interval between values a (exclusive) and z (inclusive), both of type \mathbb{T} , is defined as*

$$(a \dots z] == \{t : \mathbb{T} \mid a < t \leq z\}.$$

Similarly for left and right-open $(a \dots z)$, left and right-closed $[a \dots z]$, and left-closed, right-open $[a \dots z)$ endpoint brackets.

□

This is consistent with mathematical tradition for real intervals. (The ‘...’ separator for defining sets of reals is reminiscent of Z ’s ‘..’ operator which defines sets of integers.)

Intervals defined in this way are all finite. In the literature, infinite intervals are often introduced by leaving one argument blank, or using the infinity symbol. For instance, the open interval from a onwards may be written $(a,)$ or (a, ∞) , meaning $\{t : \mathbb{T} \mid a < t\}$. We could define equivalent notations here, but do not need to because the sets of time intervals defined below allow unbounded properties to be expressed via unbounded unions of finite intervals.

2.3 All Time Intervals

The set \mathbb{I} of all (finite) time intervals is defined as all non-empty sets of times constructable using the above interval brackets [5].

Definition 3 (All time intervals)

$$\begin{aligned} \mathbb{I} ::= & \{a, z : \mathbb{T} \mid a < z \bullet (a \dots z)\} \cup \\ & \{a, z : \mathbb{T} \mid a < z \bullet [a \dots z]\} \cup \\ & \{a, z : \mathbb{T} \mid a < z \bullet (a \dots z]\} \cup \\ & \{a, z : \mathbb{T} \mid a \leq z \bullet [a \dots z]\} \end{aligned}$$

□

The empty set is not a meaningful time interval. The shortest possible interval is a single point, denoted by a closed interval with identical endpoints. For instance, $[7 \dots 7] = \{7\}$ is a single-point interval with duration zero. Note that $(7 \dots 7)$ is *not* a valid interval, since it denotes the empty set.

2.4 Timed Traces

The central feature of the timed-trace formalisms is their use of functions from the time domain to model the dynamic behaviour of observable system properties [7, 13, 15].

Definition 4 (Timed traces) *Each variable v in a system, which may take on values from some type V , is modelled as a total function from the time domain to V :*

$$v : \mathbb{T} \rightarrow V .$$

□

As shown in Figure 1, range type V has a significant impact on the shape of the graph defined by such a trace. If V is the real numbers, as is usually the case when modelling properties of the physical environment, the graph of v may be a smooth curve. If V is a discrete type, as is usually the case when modelling

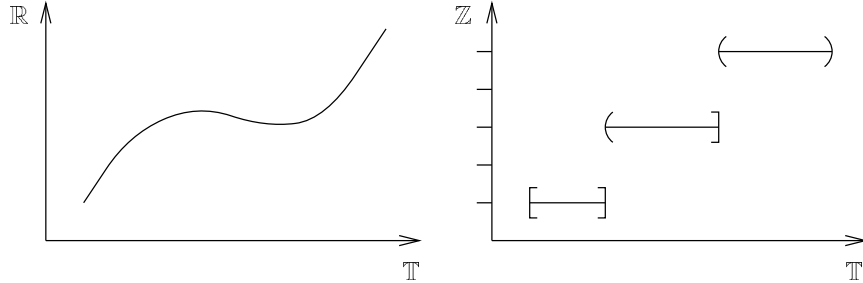


Fig. 1. Timed-trace graphs for continuous (left) and discrete (right) range types.

properties of digital hardware devices or computer program variables, the graph of v will exhibit a stepped pattern.

We have assumed that all timed traces are *total* functions. Earlier, Duddy et al. went to considerable lengths to accommodate partial functions, so that times when the observed variable does not have *any* well-defined value could be modelled by omitting such times from the domain [5, 7]. Here we instead represent times when a variable is ill-defined through underspecification of the range at that time. This is slightly weaker than Duddy et al.’s approach, since a partial function can capture the notion that sampling v when it is ill-defined may return a value outside type V . Nevertheless, this capability can be simulated when using total functions by adding a distinguished ‘not-in-type’ value \perp to V . Similarly, the Duration Calculus uses a ‘defined almost everywhere’ assumption which allows for a countable number of points of undefinedness in any observation interval [20]. Again we use a simpler approach and just assume that variables are defined everywhere.

For example, given a continuous type $Celsius == \mathbb{R}$, and a discrete type $Switch ::= On \mid Off$, the temperature of a room may be declared as

$$temp : \mathbb{T} \rightarrow Celsius,$$

and the state of an air-conditioner as

$$aircond : \mathbb{T} \rightarrow Switch.$$

2.5 Avoiding References to Time

An important feature of the Duration and timed refinement calculi is their provision for *lifting* predicates on timed traces [4, 5]. This mechanism makes it possible to elide most explicit references to the time domain when accessing timed-trace variables, thus supporting concise specifications. For example, given a $Celsius$ -valued constant $TooHot$, we may wish to write

$$TooHot < temp$$

to express the notion that the temperature is uncomfortably warm, even though *temp* is not a number, but a function. The lifting mechanisms account for this by creating overloaded versions of boolean and arithmetic operators to act on functions, as well as single values. Unfortunately, the general definitions of these lifting procedures are quite complex [4, 5]. Therefore, we instead use the simple substitution-based approach advocated by Millerchip et al [11].

Definition 5 (Time instantiation) *Let P be an expression containing free occurrences of timed-trace variables \tilde{v} , and t be a value of type \mathbb{T} . Then we define instantiation of expression P at time t as*

$$P @ t == P[\tilde{v}(t)/\tilde{v}],$$

where $P[\tilde{v}(t)/\tilde{v}]$ is expression P with any free occurrence ‘ v ’ of a timed-trace variable from \tilde{v} , which is not dereferenced, replaced by ‘ $v(t)$ ’. Appropriate renaming may be needed if t occurs bound in P .

□

In effect, this is a context-sensitive generalisation of the standard notion of syntactic substitution, relying on knowledge of which identifiers have been declared as timed traces. Care must be taken to recognise *implicit* dereferencing of timed-trace variables via, for example, relational image brackets, domain restriction and exclusion operators, integration and differentiation operators, and so on.

For example,

$$(TooHot < temp) @ 3 = TooHot < temp(3).$$

On the right, *temp* has been explicitly dereferenced by 3 because it was declared as a timed trace, but not dereferenced. Non-timed-trace constant *TooHot* is unchanged.

This special form of substitution is the only operator we introduce that differs substantially from the Z specification language.

2.6 Sets of Time Intervals

We can now introduce our principal specification modelling tool, brackets for defining the set of all time intervals during which some predicate is (everywhere) true [11]. We also use the opportunity to introduce notations for features of the intervals themselves, specifically their infimum α , supremum ω , and duration δ . For instance, given a positive \mathbb{T} -valued constant *TooLong*, we may wish to write

$$TooHot < temp \wedge TooLong \leq \delta$$

to express the notion that the temperature has been too high for too long a period.

Sets of time intervals are expressed by special brackets as follows.

Definition 6 (Sets of time intervals) Given a predicate P , containing free occurrences of timed-trace variables \tilde{v} , we define all left-open, right-closed intervals during which P is true as

$$\langle P \rangle == \{ \alpha, \omega, \delta : \mathbb{T}; \Phi : \mathbb{I} \mid \alpha < \omega \wedge \delta = \omega - \alpha \wedge \Phi = (\alpha \dots \omega) \wedge (\forall \tau : \Phi \bullet P @ \tau) \bullet \Phi \}.$$

Similarly for left and right-open $\langle P \rangle$, left-closed, right-open $[P \rangle$, and, with the condition $\alpha \leq \omega$, left and right-closed $[P]$ brackets.

□

In other words, these brackets return all intervals Φ with infimum α , supremum ω , and duration δ during which predicate P is true at every time τ in the interval. If P does not hold over any such interval the empty set is returned.

The time-instantiation operator $@$ is used in Definition 6 so that P may contain non-dereferenced occurrences of timed-trace variables. Furthermore, our definition allows times α , ω and δ to appear free in P —they are bound to the infimum, supremum and length of the interval, respectively. A similar, but considerably more complicated, capability is achieved in the Duration and timed refinement calculi by lifting terms with respect to intervals, rather than times, and allowing functions on the *interval* domain to appear in P [5,4]. The δ notation is not fundamental, however the need to refer to the duration of an interval occurs so frequently that it is worth including this redundancy. In the Duration Calculus the much-used *length* operator ℓ serves the same role [13].

(Potentially, predicate P could also directly refer to the interval Φ itself, or even to the ‘current time’ τ , since the predicate appears within the scope of these two declarations. This ability is rarely needed, however, so we shall assume that Φ and τ do not appear in P below.)

For example,

$$\begin{aligned} & \langle \text{TooHot} < \text{temp} \wedge \text{TooLong} \leq \delta \rangle \\ & = \{ \alpha, \omega : \mathbb{T} \mid \alpha < \omega \wedge (\forall \tau : (\alpha \dots \omega) \bullet \text{TooHot} < \text{temp}(\tau) \wedge \\ & \qquad \qquad \qquad \text{TooLong} \leq \omega - \alpha) \bullet (\alpha \dots \omega) \} \end{aligned}$$

defines the set of all open intervals during which the temperature is too high for too long.

In use, the brackets defined above appear similar to the $[P]$ brackets used by the Duration Calculus [20]. However, $[P]$ denotes not a set of intervals, but a *formula* which must be interpreted in the context of a particular interval [6]. Also, we have a slightly stronger notion of equivalence. In the Duration Calculus $[P] = [Q]$ tells us that properties P and Q hold for the same duration within each fixed interval, whereas in our notation $\langle P \rangle = \langle Q \rangle$ says that P and Q are both true everywhere on the same (open) intervals. We allow values to be observed at all points, and make a distinction between open and closed endpoints. The Duration Calculus’s ‘almost everywhere’ assumption means that values may not be observable on some points, and, because it is concerned only with comparing interval lengths, the Duration Calculus treats endpoint closure as unimportant.

2.7 Unspecified Endpoints

So far we have made endpoints explicit, since this is consistent with mathematical convention for describing intervals. In writing specifications, however, which may involve a degree of deliberate underspecification, we sometimes do not care whether endpoints are open or closed. Interval set brackets that allow for either an open or closed endpoint are defined trivially.

Definition 7 (Unspecified endpoints) *Given a predicate P , the set of all right-open intervals during which P is true is defined as*

$$\llbracket P \rrbracket == \lceil P \rceil \cup \{ P \}.$$

Similarly for sets of right-closed $\llbracket P \rrbracket$, left-open $\{ P \}$, and left-closed $\lceil P \rceil$ intervals. Sets of intervals where neither endpoint is specified are defined as

$$\llbracket P \rrbracket == \{ P \} \cup \lceil P \rceil \cup \{ P \} \cup \lceil P \rceil.$$

□

In practice, specific endpoint brackets are typically used for properties which are fixed in absolute time—in such situations knowing whether the endpoint is included or not can be significant. Unspecified endpoint brackets are usually used for relating system properties where the particular times at which they occur is unimportant. The style of specification promoted by the Duration Calculus also reminds us that endpoint closure is irrelevant for properties dependent on interval lengths only.

2.8 Operators on Sets of Intervals

Since properties are expressed as sets of time intervals, conventional set operators can be used for manipulating them. However, an extra, frequently-needed capability is an operator for connecting intervals end-to-end, in order to support reasoning about sequences of behaviours [12, 20, 15]. (In interval logics this concept is called “chop” because it divides a temporal property into two subintervals [12].)

Definition 8 (Concatenation) *Concatenation of elements from two sets of intervals X and Y is defined via the binary operator ‘;’:*

$$X ; Y == \{ x : X ; y : Y ; z : \mathbb{I} \mid z = x \cup y \wedge (\forall t_1 : x ; t_2 : y \bullet t_1 < t_2) \bullet z \}.$$

□

In other words, an interval x from the left-hand set can be joined to an interval y from the right-hand set, to form a new interval z , if x occurs strictly before y , and the two intervals meet *exactly*, with no overlap or gap [5]. Allowing a point of overlap [12] would prevent us from using this operator to mark those times

when a predicate changes between true and false. Alternatively, allowing an undefined point between the intervals [11] would be inconsistent with our use of total functions. Importantly, whenever two intervals are joined in this way, the supremum of the left-hand interval *equals* the infimum of the right-hand one.

For example, a requirement that the air-conditioner is on at the end of any open interval where the temperature has been excessive for too long can be expressed by the following predicate.

$$(\textit{TooHot} < \textit{temp} \wedge \textit{TooLong} \leq \delta) \subseteq (\text{true}] ; [\textit{aircond} = \textit{On}$$

Notice the use of unspecified brackets so that we are not troubled by the endpoints used by the concatenation operator to join the intervals. Either an open-closed or closed-open concatenation is allowed. In this way concatenation can be made insensitive to endpoints. (Since the ‘*aircond = On*’ interval is open on the right, it must have a non-zero duration, regardless of its left-hand endpoint.)

Many other useful operators for manipulating intervals could be defined, but are not fundamental. For instance, a prominent feature of the timed refinement calculus, missing from our notation, is the concept of *maximal interval covers* [11, 5]. There ‘ $\{P\}$ ’ is used to define those intervals which are maximal on the left with respect to P . That is, they begin at the moment P first *became* true, and thus cannot be extended any further left. However we observe that specifications relying on this operator can always be equivalently expressed using concatenation, albeit more verbosely, using the ‘ $\{\neg P\}; \{P\}$ ’ approach frequently found in Duration Calculus specifications to mark moments when some property changes from false to true or vice versa [20].

3 Laws

In this section we present a selection of laws applicable to reasoning about the above specification notation. Of course, we inherit all the usual properties of set theory [2] for reasoning about both time intervals and sets of intervals. Below we introduce laws for reasoning about interval brackets and the concatenation operator. In doing so, we adapt some of the well-established laws defined for interval logic and the Duration Calculus [14, 6].

3.1 Properties of Sets of Time Intervals

The following laws show how logical expressions relate to interval set brackets. Let P , Q and R be predicates containing (possibly non-dereferenced) occurrences of timed-trace variables \tilde{v} . Unless otherwise stated, they may also contain free occurrences of α , ω and δ .

The first law tells us how predicates and sets of intervals are related.

Law 1 (Monotonicity) *If, for all α , ω , δ and τ of type \mathbb{T} , where $\delta = \omega - \alpha$ and $\alpha \leq \tau \leq \omega$, it is the case that*

$$P @ \tau \Rightarrow Q @ \tau,$$

then

$$\llbracket P \rrbracket \subseteq \llbracket Q \rrbracket.$$

□

From Law 1 and properties of the subset operator we can derive many other useful laws. For instance, if it is always the case that $P @ \tau \Leftrightarrow Q @ \tau$ then we may conclude $\llbracket P \rrbracket = \llbracket Q \rrbracket$. Also, thanks to the transitivity of the subset relation [2, p. 22], we gain an especially useful property. If $\llbracket P \rrbracket \subseteq \llbracket Q \rrbracket$, then for a predicate P' , *stronger* than P , we can conclude $\llbracket P' \rrbracket \subseteq \llbracket Q \rrbracket$. Similarly, for a predicate Q' , *weaker* than Q , we can conclude $\llbracket P \rrbracket \subseteq \llbracket Q' \rrbracket$.

For generality, Law 1 used unspecified endpoint brackets. In this, and other laws below, we also implicitly assume *specialisations* for specific endpoints via set difference [2, p. 23]. For instance, if $\llbracket P \rrbracket \subseteq \llbracket Q \rrbracket$ then we may also conclude $\llbracket P \rrbracket \subseteq \llbracket Q \rrbracket$, by subtracting all left-closed intervals from both sides, and so on.

We can identify two extreme cases of interval specification.

Law 2 (True and false)

$$\llbracket \text{true} \rrbracket = \mathbb{I}$$

$$\llbracket \text{false} \rrbracket = \emptyset$$

□

Thus we can express the notion that some property P is true in any interval by stating $\llbracket P \rrbracket = \mathbb{I}$. We also observe that if a property is true everywhere it is false nowhere: $\llbracket P \rrbracket = \mathbb{I} \Leftrightarrow \llbracket \neg P \rrbracket = \emptyset$.

Further laws show how other logic and set operators relate. For instance, conjunction and intersection interact in an obvious way.

Law 3 (And)

$$\llbracket P \rrbracket \cap \llbracket Q \rrbracket = \llbracket P \wedge Q \rrbracket$$

□

From Law 3 and the definition of subset [2, p. 21], we gain another useful property. If $\llbracket P \wedge Q \rrbracket \subseteq \llbracket R \rrbracket$, then we may conclude $\llbracket P \wedge Q \rrbracket \subseteq \llbracket Q \wedge R \rrbracket$, thus allowing predicates on the left to be used on the right.

The relationship between disjunction and union is weaker, however.

Law 4 (Or)

$$\llbracket P \rrbracket \cup \llbracket Q \rrbracket \subseteq \llbracket P \vee Q \rrbracket$$

□

To see the cause of the asymmetry consider a situation where P is true throughout interval $[1 \dots 10]$, but at no other times, and Q is similarly true everywhere in $[5 \dots 15]$ only. In this case, interval $[3 \dots 14]$ is a member of the right-hand side of Law 4 but not the left, so we cannot strengthen the inequality.

Another asymmetric relationship holds between negation and set difference.

Law 5 (Not)

$$\llbracket \neg P \rrbracket \subseteq \mathbb{I} \setminus \llbracket P \rrbracket$$

□

To illustrate the asymmetry consider the situation where P is false at all times except time 1. In this case, interval $(0 \dots 2)$ is a member of the right-hand side above, but not the left.

3.2 Properties of Concatenation

The following laws show how concatenation interacts with other set operators. Let S , T , U and V be sets of time intervals, i.e., elements of type $\mathbb{P}\mathbb{I}$, where \mathbb{P} denotes powerset.

From Definition 8 we can derive the following fundamental laws [14, p. 41].

Law 6 (Concatenation monotonicity) *If*

$$S \subseteq S' \text{ and } T \subseteq T',$$

then

$$S ; T \subseteq S' ; T'.$$

□

Law 7 (Concatenation associativity)

$$(S ; T) ; U = S ; (T ; U)$$

□

Definition 8 also tells us that the empty set of intervals acts as the zero of the concatenation operator. It thus serves the same role as formula ‘*false*’ in the Duration Calculus [14, p. 41].

Law 8 (Concatenation zero)

$$S ; \emptyset = \emptyset ; S = \emptyset$$

□

Concatenation distributes over union [14, p. 41].

Law 9 (Concatenate union)

$$(S \cup T); U = (S; U) \cup (T; U)$$

$$S; (T \cup U) = (S; T) \cup (S; U)$$

□

The equivalent law for intersection is weaker, however.

Law 10 (Concatenate intersection)

$$(S \cap T); U \subseteq (S; U) \cap (T; U)$$

$$S; (T \cap U) \subseteq (S; T) \cap (S; U)$$

□

Ravn suggests the following example to illustrate the asymmetry [14, p. 42]:

$$(\{\delta = 1\} \cap \{\delta < 1\}); [\text{true}] \not\subseteq (\{\delta = 1\}; [\text{true}]) \cap (\{\delta < 1\}; [\text{true}]).$$

We can easily find intervals to satisfy the right-hand side. However, because $\{\delta = 1\} \cap \{\delta < 1\} = \emptyset$, due to the contradictory requirements on the duration, the left-hand side is empty.

A symmetric law for distributing concatenation over intersection can be provided for intervals of fixed length, however [14, pp. 41–2].

Law 11 (Concatenate fixed intersection) *If r is a non-negative value of type \mathbb{T} , then*

$$((S \cap [\delta = r]); T) \cap ((U \cap [\delta = r]); V) = (S \cap U \cap [\delta = r]); (T \cap V)$$

and

$$(S; (U \cap \{\delta = r\})) \cap (T; (V \cap \{\delta = r\})) = (S \cap T); (U \cap V \cap \{\delta = r\}).$$

Similarly in the first case with the $[\cdot]$ brackets replaced by $[\cdot]$, and in the second case with the $\{\cdot\}$ brackets replaced by $\{\cdot\}$.

□

The restriction on δ serves to fix the lengths of the intervals so that the two concatenations on the left occur at the same time. A specific endpoint is needed where the concatenations occur to ensure that the two pairs of intervals are joined with the same open-closed or closed-open combination.

Other useful properties of concatenation also rely on the lengths of intervals. An interval can be composed of two subintervals, provided that it does not have zero duration [14, p. 45].

Law 12 (Concatenate property) *If α , ω and δ do not occur free in P , then*

$$[\![P \wedge 0 < \delta]\!] = [\![P]\!]; [\![P]\!].$$

□

In specialisations of this law where the intervals on the left have open endpoints, and hence non-zero duration, we can omit the condition on δ and just state $\langle P \rangle = \langle P \rangle ; \llbracket P \rrbracket$, and so on. Law 12 restricts the appearance of α , ω and δ in predicate P because, in general, the *same* time-dependent predicate cannot appear on both sides of the concatenation operator. For instance, it is clear that $\llbracket \delta = 6 \rrbracket$ is not equal to $\llbracket \delta = 6 \rrbracket ; \llbracket \delta = 6 \rrbracket$.

A variant of Law 12 shows how concatenation relates to duration [14, p. 44].

Law 13 (Concatenate duration) *If α , ω and δ do not occur free in P , and r and s are non-negative values of type \mathbb{T} , where $r > 0$ or $s > 0$, then*

$$\llbracket P \wedge \delta = r + s \rrbracket = \llbracket P \wedge \delta = r \rrbracket ; \llbracket P \wedge \delta = s \rrbracket .$$

□

4 Example

As a concrete example we present a proof that a specified property is achieved via a periodic design. This is a common situation in embedded real-time devices that repeatedly sample some value, process it, and output the result.

4.1 Specifications

Consider the specification of a speedometer. Its input consists of a voltage v , proportional to the velocity of the vehicle, produced by an axle-mounted sensor. The required output is an estimate s of the vehicle's speed. We assume the system is calibrated so that a satisfactory value for s is calculated easily by multiplying v by some constant M .

$$\mid M : \mathbb{R}_+ \qquad \qquad \qquad \text{[Multiplier for estimating speed from voltage]}$$

We consider the value of output s to be acceptable if it differs from $v * M$ by no more than some error E .

$$\mid E : \mathbb{R}_+ \qquad \qquad \qquad \text{[Acceptable error magnitude]}$$

To describe the times when speed s has an acceptable value, we introduce an auxiliary, $\{0, 1\}$ -valued ‘good-speed’ function g , which equals 1 only when speed s is within the acceptable range. Let \mathbb{D} denote the set of differentiable real-to-real functions, i.e., continuous functions with graphs for which a unique tangent exists at all points [18, pp. 32–3]. Let \mathbb{S} be the set of integrable real-to-real functions, e.g., those that exhibit bounded piecewise continuity [18, p. 182]. Boundedness means that there is some finite constant greater than the magnitude of the function at all points. Piecewise continuity requires that any interval can be partitioned into a finite sequence of nonoverlapping open subintervals over each of which the function is continuous. For some number a and non-negative number b , let $a \pm b == \{c : \mathbb{R} \mid a - b \leq c \leq a + b\}$.

<i>InputOutput</i>	
$v : \mathbb{D}$	[Input voltage]
$s : \mathbb{T} \rightarrow \mathbb{R}$	[Output speed]
$g : \mathbb{S} \cap (\mathbb{T} \rightarrow \{0, 1\})$	[Auxiliary good-speed function]
$\llbracket g = 1 \Leftrightarrow s \in v * M \pm E \rrbracket = \mathbb{I}$	

In other words, in all time intervals, g equals 1 if and only if speed s stays within the acceptable error bounds.

Since it would be unreasonable to expect the system to produce acceptable values at *all* times, we merely require that in any observation interval of length at least I , that there will be acceptable outputs for some proportion P of the time.

$I : \mathbb{T}$	[Minimum observation interval]
$P : \mathbb{R}$	[Acceptable proportion of good output times]
$0 < I$	
$0 < P < 1$	

The system is required to produce good outputs only after starting time 0, to allow for initialisation activities. The overall requirement is then expressed as follows. Let $\int_a^b f$ denote the integral of function f between times a and b [18, p. 174].

<i>SpeedOk</i>
<i>InputOutput</i>
$(0 \leq \alpha \wedge I \leq \delta) \subseteq (\delta * P \leq \int_\alpha^\omega g)$

This exploits the fact that in intervals where function g is everywhere equal 1, the integral of g also equals the duration of the interval. The Duration Calculus makes extensive use of this property by making all predicates $\{0, 1\}$ -valued, so that integrals and durations are interchangeable [20].

To make a solution to this requirement feasible we need an assumption about the maximum acceleration of the vehicle, and hence worst-case rate of change of the voltage v . The magnitude of the voltage's rate of change is assumed to be no greater than some constant R .

$R : \mathbb{R}_+$	[Worst-case voltage rate of change]
--------------------	-------------------------------------

The following assumption states that this is true in any interval after starting time 0. Let f' denote the derivative of function f [18, p. 33]. (This notation should not be confused with the priming convention used when expressing state-machine models in Z .)

<i>VoltageRate</i>
$v : \mathbb{D}$
$\llbracket 0 \leq \alpha \rrbracket \subseteq \llbracket v' \leq R \rrbracket$

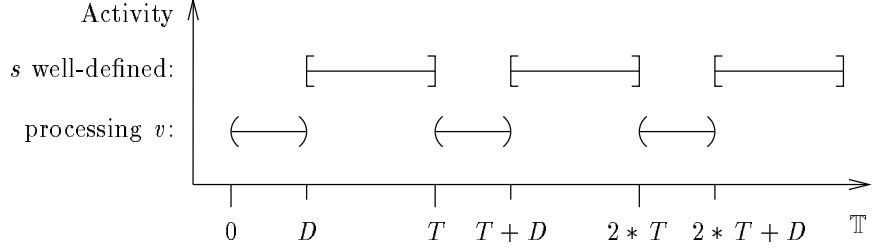


Fig. 2. Behaviour of periodic speedometer.

We now propose a solution to the above requirement in the form of a periodic behaviour, with period T and relative deadline D .

$$\begin{array}{|l}
 T : \mathbb{T} \\
 D : \mathbb{T} \\
 \hline
 0 < D < T
 \end{array}
 \begin{array}{l}
 \text{[Period]} \\
 \text{[Deadline relative to start of period]}
 \end{array}$$

As shown in Figure 2, the intention is that the system will start processing voltage v after the start of each period, producing a new value for speed s by the deadline. In every period there is therefore an initial open interval of duration D during which input v will be sampled, and the value of output s may be ill-defined. In the remainder of the period, output s must equal M times the sampled value of v . Thus, in the n^{th} period, speed s will be well-defined for at least the closed interval from time $n * T + D$ to $(n + 1) * T$. Let $f(\cdot)$ be the image of set S through function f .

$$\begin{array}{|l}
 \text{Speedometer} \\
 \text{InputOutput} \\
 \hline
 \{n : \mathbb{N} \bullet [n * T + D \dots (n + 1) * T]\} \subseteq \{s/M \in v((\alpha - D \dots \alpha))\}
 \end{array}$$

The set on the left consists of all closed intervals during which s should be well defined. It can be expressed equivalently without the set-comprehension brackets as $[\exists n : \mathbb{N} \bullet \alpha = n * T + D \wedge \delta = T - D]$. On the right, acceptable values of s in such intervals are defined, relative to values of v . Notice that interval $(\alpha - D \dots \alpha)$, used to define the times when v may be sampled, lies *outside* the interval defined by the enclosing $[\cdot]$ brackets. Such expressive power is one of the advantages of making the current infimum and supremum explicitly accessible.

4.2 Proof

Our task now is to show that the design expressed by *Speedometer* satisfies requirement *SpeedOk* in the presence of assumption *VoltageRate* [13]. In other words, we must prove that

$$(\text{Speedometer} \wedge \text{VoltageRate}) \Rightarrow \text{SpeedOk} .$$

As a first step we use the concatenation operator to extend the speedometer property to cover an entire period, not just the part after the deadline. To do this we prefix an open interval of duration D .

$$\begin{aligned}
& \textit{Speedometer} \\
\Rightarrow & \text{'by Law 6'} \\
& (\delta = D); \{n : \mathbb{N} \bullet [n * T + D \dots (n + 1) * T]\} \\
& \subseteq (\delta = D); [s/M \in v((\alpha - D \dots \alpha))] \\
\Leftrightarrow & \text{'by Definition 8'} \\
& \{n : \mathbb{N} \bullet (n * T \dots (n + 1) * T)\} \\
& \subseteq (\delta = D); [s/M \in v((\alpha - D \dots \alpha))] \tag{1}
\end{aligned}$$

Next, we wish to determine how the input voltage v behaves during a period. To do so, we firstly observe that

$$\{n : \mathbb{N} \bullet (n * T \dots (n + 1) * T)\} \subseteq (\delta = T), \tag{2}$$

which tells us the length of each period, and

$$\{n : \mathbb{N} \bullet (n * T \dots (n + 1) * T)\} \subseteq [0 \leq \alpha], \tag{3}$$

which tells us the earliest starting time of each period. The second of these can be combined with the overall assumption to determine how v may change in a period.

$$\begin{aligned}
& (3) \wedge \textit{VoltageRate} \\
\Rightarrow & \text{'by transitivity of subset and set difference'} \\
& \{n : \mathbb{N} \bullet (n * T \dots (n + 1) * T)\} \subseteq (|v'| \leq R) \tag{4}
\end{aligned}$$

This known rate-of-change can then be partitioned into the two subintervals of interest in each period.

$$\begin{aligned}
& \{n : \mathbb{N} \bullet (n * T \dots (n + 1) * T)\} \\
& \subseteq \text{'by (2) and (4) and absorption [10, p. 71]'} \\
& (|v'| \leq R) \cap (\delta = T) \\
& = \text{'by Law 3'} \\
& (|v'| \leq R \wedge \delta = T) \\
& = \text{'by Law 13'} \\
& (|v'| \leq R \wedge \delta = D); [|v'| \leq R \wedge \delta = T - D] \tag{5}
\end{aligned}$$

For each subinterval of each period, we now know how voltage v behaves, and how speed s is defined in terms of v . These two pieces of information are combined as follows.

$$\begin{aligned}
& \{n : \mathbb{N} \bullet (n * T \dots (n + 1) * T)\} \\
\subseteq & \text{'by (1) and (5) and absorption'} \\
& ((\delta = D) ; [s/M \in v((\alpha - D \dots \alpha))]) \cap \\
& ((|v'| \leq R \wedge \delta = D) ; [|v'| \leq R \wedge \delta = T - D]) \\
= & \text{'by Law 3'} \\
& ((\delta = D) ; [s/M \in v((\alpha - D \dots \alpha))]) \cap \\
& ((|v'| \leq R) \cap (\delta = D) ; [|v'| \leq R \wedge \delta = T - D]) \\
= & \text{'by Law 11'} \\
& ((|v'| \leq R) \cap (\delta = D) ; \\
& [s/M \in v((\alpha - D \dots \alpha))]) \cap [|v'| \leq R \wedge \delta = T - D]) \\
= & \text{'by Law 3'} \\
& (|v'| \leq R \wedge \delta = D) ; \\
& [s/M \in v((\alpha - D \dots \alpha)) \wedge |v'| \leq R \wedge \delta = T - D] \tag{6}
\end{aligned}$$

We then use the behaviour of voltage v , and the known durations, to determine how stale the value of s may be. Firstly we express the rate of change as a property relative to the point where the concatenation occurs. Notice the use of expressions in which v occurs in both dereferenced and non-dereferenced instances.

$$\begin{aligned}
& \{n : \mathbb{N} \bullet (n * T \dots (n + 1) * T)\} \\
\subseteq & \text{'by (6) and Laws 1 and 6'} \\
& (v \in v(\omega) \pm D * R \wedge \delta = D) ; \\
& [s/M \in v((\alpha - D \dots \alpha)) \wedge v(\alpha) \in v \pm (T - D) * R] \\
= & \text{'by Definition 6'} \\
& (v((\omega - D \dots \omega)) \subseteq v(\omega) \pm D * R \wedge \delta = D) ; \\
& [s/M \in v((\alpha - D \dots \alpha)) \wedge v(\alpha) \in v \pm (T - D) * R] \tag{7}
\end{aligned}$$

We then exploit the fact that time ω for an interval on the left-hand side of a concatenation equals time α for the right-hand one.

$$\begin{aligned}
& \{n : \mathbb{N} \bullet (n * T \dots (n + 1) * T)\} \\
\subseteq & \text{'by (7) and Definition 8 and Laws 1 and 6'} \\
& (\delta = D) ; [s/M \in v(\alpha) \pm D * R \wedge v(\alpha) \in v \pm (T - D) * R] \\
\subseteq & \text{'by Laws 1 and 6'} \\
& (\delta = D) ; [s \in v * M \pm T * R * M] \tag{8}
\end{aligned}$$

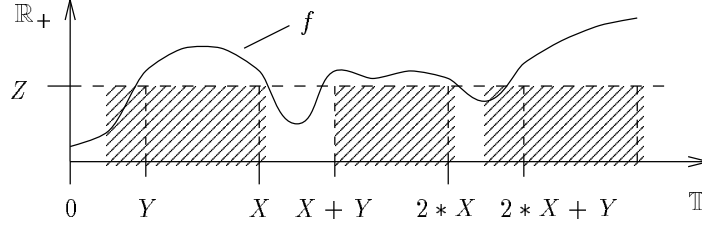


Fig. 3. A behaviour of function f consistent with the condition for Lemma 1.

The definition of auxiliary function g can be used to simplify this predicate.

$$\begin{aligned}
 & (8) \wedge \text{InputOutput} \\
 & \Rightarrow \text{'by Laws 1 and 6'} \\
 & \{n : \mathbb{N} \bullet (n * T \dots (n + 1) * T)\} \subseteq \{\delta = D\}; [g = 1] \quad (9)
 \end{aligned}$$

This step introduces proviso $T * R * M \leq E$, relating the sampling period and input rate of change to the acceptable error. A smaller acceptable error will require a faster sampling rate, or a slower rate of change, as one would expect.

Having proven a property of the system relative to fixed intervals, we need a way of generalising its applicability. To do this we introduce the following lemma which uses integration to extend repetitive properties to longer intervals.

Lemma 1. *Let f be a non-negative function of type $\mathbb{R} \rightarrow \mathbb{R}_+$. Also let X and Y be non-negative values of type \mathbb{T} , where $Y \leq X$, and Z be a non-negative real value. If*

$$\{n : \mathbb{N} \bullet (n * X \dots (n + 1) * X)\} \subseteq \{\delta = Y\}; [Z \leq f]$$

then

$$\{0 \leq \alpha \wedge X \leq \delta\} \subseteq \{\delta * Z * (X - Y) / (X + Y) \leq \int_{\alpha}^{\omega} f\}.$$

□

In other words, we initially know that in every period, of duration X , the value of f is always at least Z , except for an initial ‘undefined’ subinterval of duration Y . In Figure 3 we can see that f does not enter the shaded areas. From this we wish to conclude that in any open interval, of duration at least X , the integral of f will be at least the duration times $Z * (X - Y) / (X + Y)$.

Proof outline Since f equals at least Z for at least $X - Y$ time units in any one period, term $Z * (X - Y)$ represents the *least* integral of f in a period. Term $X + Y$ is the duration of a worst-case observation interval. This is one covering two undefined subintervals, but only one defined subinterval. The first such worst-case interval is $[0 \dots X + Y]$. In longer observation intervals, that similarly begin and end with whole undefined subintervals, the ratio of defined

to undefined subintervals improves on this worst case, going from 1/2 to 2/3 to 3/4, and so on.

□

The overall proof is then completed by appealing to the lemma.

$$\begin{aligned}
(9) &\Rightarrow \text{'by Lemma 1'} \\
&\{0 \leq \alpha \wedge T \leq \delta\} \subseteq \{\delta * (T - D)/(T + D) \leq \int_{\alpha}^{\omega} g\} \\
&\Rightarrow \text{'by definition'} \\
&\textit{SpeedOk} \tag{10}
\end{aligned}$$

The last step introduces provisos $T \leq I$ and $P \leq (T - D)/(T + D)$. The first of these tells us that the period must be no greater than the minimum observation interval. The second relates the deadline and period to the proportion of time during which good values can be observed. As the value of deadline D approaches the period T , it reduces the possible proportion P of 'good' time. Again, this matches our intuitions, since a larger deadline relative to the period allows output s to be ill-defined for longer intervals.

Thus, if we satisfy the provisos on the system constants introduced during steps (9) and (10), we can finally conclude that our proposed design does indeed satisfy its specification.

5 Conclusion

We have introduced a set-theoretic model for specification and reasoning using timed traces. It embodies many features found in previous real-time formalisms, yet requires little specialised mathematical knowledge, so should be amenable to industrial uptake by programmers with minimal formal methods experience, and representable in existing formal development tools. The definitions avoid the usual need for implicit expression lifting with respect to intervals by making the current interval endpoints explicitly accessible. At the time of writing we are extending the set of laws with rules for induction, and starting work on a theorem-prover representation of the model.

Acknowledgements We gratefully acknowledge our debt to unpublished work by Brendan Mahony, Keith Duddy, Luke Everett and Colin Millerchip for inspiring the overall approach adopted in this paper. We wish to thank Brendan Mahony, Graeme Smith and the anonymous referees for their many helpful comments on this work. This research is funded by the Information Technology Division of the Defence Science and Technology Organisation.

References

1. S. Atkinson and D. Scholefield. Transformational vs reactive refinement in real-time systems. *Information Processing Letters*, 55:201–210, 1995.

2. E. J. Billington, D. Donovan, B. D. Jones, S. Oates-Williams, and A. Street. *Discrete Mathematics: Logic and Structures*. Longman, 1990.
3. J. G. Bollinger and N. A. Duffie. *Computer Control of Machines and Processes*. Addison-Wesley, 1988.
4. S. M. Brien, M. Engel, He Jifeng, A. Ravn, and H. Rischel. Z description of duration calculus. Draft, Oxford University Computing Laboratory, August 1993.
5. K. Duddy, L. Everett, C. Millerchip, B. Mahony, and I. J. Hayes. Z-based notation for the specification of timing properties. Draft, Department of Computer Science, University of Queensland, June 1995.
6. M. R. Hansen and Zhou Chaochen. Duration calculus: Logical foundations. *Formal Aspects of Computing*, 9(3):283–330, 1997.
7. B. Mahony and I. J. Hayes. Using continuous real functions to model timed histories. In *Proc. Sixth Australian Software Engineering Conference (ASWEC'91)*, Sydney, July 1991.
8. B. P. Mahony. The refinement calculus and data-flow processes. In *Proc. Second Australasian Refinement Workshop*, pages 1–28, Brisbane, September 1992.
9. B. P. Mahony and I. J. Hayes. A case-study in timed refinement: A mine pump. *IEEE Transactions on Software Engineering*, 18(9):817–826, September 1992.
10. A. Margaris. *First Order Mathematical Logic*. Blaisdell, 1967.
11. C. Millerchip, B. Mahony, and I. J. Hayes. The generic problem competition: A whole system specification of the boiler system. Software Verification Research Centre, University of Queensland, June 1993.
12. B. Moszkowski. *Executing Temporal Logic Programs*. Cambridge University Press, 1986.
13. E.-R. Olderog, A. P. Ravn, and J. U. Skakkebak. Refining system requirements to program specifications. In C. Heitmeyer and D. Mandrioli, editors, *Formal Methods for Real-Time Computing*, volume 5 of *Trends in Software*, chapter 5, pages 107–134. Wiley, 1996.
14. A. P. Ravn. *Design of Embedded Real-Time Computing Systems*. PhD thesis, Department of Computer Science, Technical University of Denmark, 1995.
15. D. Scholefield, H. Zedan, and He Jifeng. A specification-oriented semantics for the refinement of real-time systems. *Theoretical Computer Science*, 131:219–241, 1994.
16. J. U. Skakkebak. *A Verification Assistant for a Real-Time Logic*. PhD thesis, Department of Computer Science, Technical University of Denmark, 1994.
17. J. M. Spivey. *The Z Notation: A Reference Manual*. Prentice Hall International, 1989.
18. G. B. Thomas, Jr. *Calculus and Analytic Geometry*. Addison-Wesley, 4th edition, 1968.
19. S. H. Valentine. An algebraic introduction of real numbers into Z. In H. Habrias, editor, *7th International Conference on: Putting into practice methods and tools for information system design, Z Twenty Years On — What is its Future?*, Nantes, France, October 1995.
20. Zhou Chaochen. Duration calculi: An overview. In D. Bjorner, M. Broy, and I. Pottosin, editors, *Formal Methods in Programming and Their Applications*, volume 735 of *Lecture Notes in Computer Science*, pages 256–266. Springer-Verlag, 1993. Extended abstract.