A Simple and Flexible Timing Constraint Logic

Björn Lisper[1], Johan Nordlander[2]

[1] School of Innovation, Design, and Engineering, Mälardalen University, SE-721 23 Västerås, Sweden
[2] Department of Computer Science, Electrical and Space Engineering, Luleå University of Technology, SE-971 87 Luleå, Sweden

Abstract. Formats for describing timing behaviors range from fixed menus of standard patterns, to fully open-ended behavioral definitions; of which some may be supported by formal semantic underpinnings, while others are better characterized as primarily informal notations. Timing descriptions that allow flexible extension within a fully formalized framework constitute a particularly interesting area in this respect.
We present a small logic for expressing timing constraints in such an open-ended fashion, sprung out of our work with timing constraint semantics in the TIMMO-2-USE project [15]. The result is a non-modal, first-order logic over reals and sets of reals, which references the constrained objects solely in terms of event occurrences. Both finite and infinite behaviors may be expressed, and a core feature of the logic is the ability to restrict any constraint to just the finite ranges when a certain system mode is active.
Full syntactic and semantic definitions of our formula language are given, and as an indicator of its expressiveness, we show how to express all constraint forms currently defined by TIMMO-2-USE and AUTOSAR. A separate section deals with the support for mode-dependencies that have been proposed for both frameworks, and we demonstrate by an example how our generic mode-restriction mechanism formalizes the details of such an extension.

1 Introduction

Timing behavior descriptions exist in many different forms. Classical real-time scheduling theory defines the basic *periodic* and *sporadic* patterns to describe task activations, along with the simple notion of *relative deadlines* for capturing the desired behavior of a system's response. Digital circuits are often accompanied by *timing diagrams* [4], where selected scenarios from an infinitely repeating behavior are depicted graphically, specifically indicating the minimum and maximum distances between key events. In the automotive domain, the model-based development frameworks of AUTOSAR [6] and EAST-ADL [8] offer a rich palette of *built-in timing patterns* and constraints, commonly specified in terms of typical-case timing diagrams. On the theoretical side, *temporal* and *real-time logics* concentrate on a few basic building blocks, from which more complex timing formulae can be constructed using logical connectives.

The timing models of classical scheduling theory are well-understood, but limited in expressiveness and essentially closed — even though they have been successfully extended with notions such as jitter and release offsets, every extension has to show that it also can be understood and analyzed in ways that mirror the original theory. As a contrast, graphical timing diagrams appear inherently open-ended, but this is primarily the consequence of a lack of rigor in this informal notation. AUTOSAR and EAST-ADL can express some very complex timing behaviors, but pay the price of being both informal as well as closed to extension. A formal foundation for the timing constructs of both languages has previously been defined by the TIMMO-2-USE project, but extensibility of this foundation has so far not been addressed.

This paper contributes a retake on the TIMMO-2-USE formalization effort, by means of a *timing constraint logic* that is able to express all existing constraints, while also acting as a toolbox for building new and open-ended forms of well-defined timing behaviors. The logic, called *TiCL* (Timing Constraint Logic), is similar to existing real-time logics in this respect, but differs in the following important ways:

– TiCL is a logic of *pure timing constraints*. It does not attempt to express any functional properties of the systems it constrains, and it only interfaces to the latter via the notion of *event occurrences*. This separation of concerns is central to the ability to blend with EAST-ADL and AUTOSAR, whose complex semantics does not yet allow full formalization of functional behavior.
– TiCL is not a modal logic. In fact, TiCL just represents a carefully chosen selection of operators from a standard first-order logic over the real numbers and real number sets.
– TiCL is not restricted to infinite behaviors only. Finite behaviors can be expressed with ease, and one of the strengths of TiCL is a mechanism for restricting a generic constraint to just the finite ranges when a certain system *mode* is active.

In Section 2 we introduce introduce TADL, a language for timing constraints that was defined in the TIMMO project and has influenced the AUTOSAR Timing Extensions. We then define the syntax and semantics of TiCL in Section 3, establish some convenient notational short-hands (Section 4), and show how current TADL and AUTOSAR constraints are captured (Section 5). The mechanism for interpreting mode-dependencies is explained in Section 6. We discuss related work in Section 7. Verification and analysis issues are beyond the scope of the current paper, but the topic will be returned to in the concluding discussion (Section 8).

2 TADL

The *Timing Augmented Description Language* (TADL) [10] is a constraint language for describing timing requirements and properties within the automotive domain. It was originally defined in the TIMMO project, and is now being revised and formalized within the TIMMO-2-USE project: TiCL is an outcome of this work. The syntax of TADL is compliant to the AUTOSAR meta-model, but the TADL constraints can also be understood through a textual syntax.

TADL defines constraints on *events*, which are simply (finite or infinite) sequences of strictly increasing times. The definition does not specify whether times are integers or reals: the constraints have meaningful interpretations in both cases. An element in an event is an *occurrence* of the event.

TADL's constraints, as defined in [10], can be divided into three groups: *repetition rate constraints*, which concern single events, *delay constraints*, which concern the timing relation between *stimuli* and *responses*, and *synchronization constraints*, that require that corresponding occurrences of a group of events appear in sufficiently tight clusters.

All repetition rate constraints can be seen as instances of a *generic repetition rate constraint.* Such a constraint is specified by four parameters *lower*, *upper*, ,*jitter*, and *span*. An event $\langle t_1, t_2, \ldots \rangle$ satisfies a generic repetition rate constraint iff there exists an sequence of times $\langle x_1, x_2, \ldots \rangle$ such that for all $i > 1$,

$$x_i \leq t_i \leq x_i + \mathit{jitter}$$

and for all $i \geq \mathit{span}$,

$$\mathit{lower} \leq x_i - x_{i-\mathit{span}} \leq \mathit{upper}$$

Now, a *periodic* repetition constraint is a generic repetition rate constraint where $\mathit{span} = 1$, and $\mathit{lower} = \mathit{upper}$. A *sporadic* repetition constraint has $\mathit{span} = 1$, and $\mathit{upper} = \infty$. TADL also defines more complex *pattern* repetition constraints, and *arbitrary* repetition constraints, see [10].

Delay constraints relate two events, called stimulus and response, by demanding that each occurrence of one event is matched by at least one occurrence of the other within some time window. Depending on whether these time windows are achored at the stimulus or response occurrences, TADL names the delay constraints *reaction* or *age*, respectively.

Both the *reaction* and *age* constraints are characterized by the parameters *lower*, and *upper*. A stimulus event $\langle s_1, s_2, \ldots \rangle$ and a response event $\langle r_1, r_2, \ldots \rangle$ satisfy a reaction constraint with parameter *lower*, *upper* iff for all s_i there exists r_j such that

$$s_i + \mathit{lower} \leq r_j \leq s_i + \mathit{upper}$$

The same events satisfy an age constraint with the same parameters, iff for all r_j there exists an s_i such that

$$r_j - \mathit{upper} \leq s_i \leq r_j - \mathit{lower}$$

Synchronization constraints were originally defined as rather complex constructs in both TADL and AUTOSAR, being syntactic (but notably not semantic) extensions of the delay constraints [10]. Both language have since then simplified the notion of synchronization considerably, and the upcoming release of TADL v2 defines synchronization as a constraint on a group of events S, characterized by a single parameter *tolerance*. Such a constraint is satisfied iff there is a sequence of times $\langle x_1, x_2, \ldots\rangle$ such that for all x_i and all events $\langle s_1, s_2, \ldots\rangle \in S$ there exists at least one s_j such that

$$x_i \leq s_j \leq x_i + \mathit{tolerance}$$

The TADL v1 definition of synchronization will be further discussed at the end of Section 5.

A common theme in the definitions above is that they all rely on an infinite number of indexed event occurrences, which excludes their use in scenarios that span only finite intervals (as in mode-switching systems, for example). They are also practically closed, due to the fact that the logic in which the definitions are expressed is left unspecified in the current TADL. Both these deficiencies will be addressed by the introduction of TiCL.

3 TiCL

The basic purpose of the timing constraint language TiCL is to express truth statements about the points in time when *events* occur. Points in time are interpreted as real values, and since such values are totally ordered, events can simply be understood as sets of reals (infinite or finite). We make the choice to also represent *time intervals* as sets of time values; such sets are however always infinite. Three different sets of variables form the basis of TiCL: one denoting time values (**Tvar**), one ranging over sets (**Svar**), and yet another form standing for arbitrary arithmetic values not denoting points in time (**Avar**). The syntax of TiCL is given in Fig. 1.

Syntactic categories

$r \in \mathbb{R}$ (arithmetic constants)

v	$\in$	**Avar** (arithmetic variables)	e, f	$\in$	**AExp** (arithmetic expressions)
X, Y	$\in$	**Svar** (set variables)	E, F	$\in$	**SExp** (set expressions)
x, y	$\in$	**Tvar** (time variables)	c, d	$\in$	**CExp** (constraint expressions)

Abstract syntax

$$e \rightarrow r \mid v \mid e + f \mid e - f \mid e * f \mid e / f \mid |E| \mid \lambda(E)$$

$$E \rightarrow X \mid \{x : c\}$$

$$c \rightarrow e \leq f \mid x \leq y \mid x \in E \mid c \wedge d \mid \neg c \mid \forall v : c \mid \forall x : c \mid \forall X : c$$

Fig. 1. TiCL syntax.

TiCL distinguishes between three kinds of terms. **AExp** is the set of arithmetic expressions formed from constants, variables, arithmetic operators, as well as the size $|E|$, or the *measure* $\lambda(E)$, of a set expression E. By measure we mean the total length of all continuous intervals in E (that is, the *Lebesgue* measure of E). The set expressions **SExp** take the form of a set variable X, or a set comprehension $\{x : c\}$ – the set of all times x such that constraint c (which may reference x) is true. **CExp**, finally, stands for the set of boolean constraint formulae formed from inequalities between arithmetic expressions, inequalities between time variables, set membership (x belongs to the set of times denoted by E), logical connectives, and quantification over arithmetic, time and set variables.

$$\begin{aligned}
\mathit{true} &\equiv 0 \leq 1 \\
\mathit{false} &\equiv 1 \leq 0 \\
c \vee d &\equiv \neg(\neg c \wedge \neg d) \\
c \Rightarrow d &\equiv \neg c \vee d \\
c \Leftrightarrow d &\equiv (c \Rightarrow d) \wedge (d \Rightarrow c) \\
e = f &\equiv e \leq f \wedge f \leq e \\
e \neq f &\equiv \neg(e = f) \\
e < f &\equiv e \leq f \wedge e \neq f
\end{aligned}
\qquad
\begin{aligned}
\exists v : c &\equiv \neg(\forall v : \neg c) \\
\exists x : c &\equiv \neg(\forall x : \neg c) \\
\exists X : c &\equiv \neg(\forall X : \neg c)
\end{aligned}$$

$$\begin{aligned}
\forall x \in E : c &\equiv \forall x : x \in E \Rightarrow c \\
\exists x \in E : c &\equiv \exists x : x \in E \wedge c \\
\exists X = E : c &\equiv \exists X : X = E \wedge c \\
\{x \in E : c\} &\equiv \{x : x \in E \wedge c\}
\end{aligned}$$

$$\begin{aligned}
E \subseteq F &\equiv \forall x : x \in E \Rightarrow x \in F \\
E = F &\equiv E \subseteq F \wedge F \subseteq E \\
E \neq F &\equiv \neg(E = F) \\
E \subset F &\equiv E \subseteq F \wedge E \neq F \\
x \notin E &\equiv \neg(x \in E)
\end{aligned}
\qquad
\begin{aligned}
E \cup F &\equiv \{x : x \in E \vee x \in F\} \\
E \cap F &\equiv \{x : x \in E \wedge x \in F\} \\
E^{\mathsf{C}} &\equiv \{x : x \notin E\} \\
E \backslash F &\equiv \{x : x \in E \wedge x \notin F\}
\end{aligned}$$

Fig. 2. Standard syntactic abbreviations.

The syntax of TiCL is thus entirely standard, and should—with the possible exception of the **Tvar**/**Avar** distinction—suggest an absolutely straightforward first-order logic semantics. The reason why time variables are kept distinct from their arithmetic counterparts is that absolute time values are never interesting in their own right; only their relative distances are. By making it impossible to form arithmetic expressions directly from time variables, the TiCL constraints become independent of the arbitrary point in time a user chooses to refer to as time "zero". This contrasts sharply to the arithmetic variables, which typically stand for aspects such as minimum interval length, maximum number of occurrences, etc—i.e., aspects whose absolute values are of prime interest in the definition of timing constraints.

As an example TiCL formula, here follows a constraint that demands the occurrences of event X to be no more than, and to occur no later than, the occurrences of event Y.[3]

$$|X| \leq |Y| \wedge \forall x : \forall y : \neg(x \in X \wedge y \in Y \wedge \neg(x \leq y))$$

4 Abbreviations

For added convenience, we complement the basic syntax of TiCL with a series of syntactic abbreviations, the first of which is defined in Fig. 2. By taking advantage of these notational short-hands, we may choose to express the example constraint of the previous section as follows:

$$|X| \leq |Y| \wedge \forall x \in X : \forall y \in Y : x \leq y$$

Sets of time values do not only represent the generally sparse points in time where different events occur, but also the notion of dense intervals – i.e., sets that contain *all* time values above or below chosen endpoints. Fig. 3 defines some useful interval constructors, that take either single time values, or sets of such values, as starting points.

$$\begin{array}{ll}
[x \leq] \equiv \{y : x \leq y\} & [E \leq] \equiv \{y : \exists x \in E : x \leq y\} \\
[x <] \equiv \{y : x < y\} & [E <] \equiv \{y : \forall x \in E : x < y\} \\
[\leq x] \equiv [x <]^{\complement} & [\leq E] \equiv [E <]^{\complement} \\
[< x] \equiv [x \leq]^{\complement} & [< E] \equiv [E \leq]^{\complement} \\
[x..y] \equiv [x \leq] \cap [< y] & [E] \equiv [E \leq] \cap [\leq E]
\end{array}$$

Fig. 3. Interval constructors.

Intervals are important for separating legal and illegal occurrences of events. The following operations filter out occurrences of an event that are either above or below a certain point in time.

$$\begin{aligned} E_{x<} &\equiv E \cap [x <] \\ E_{<x} &\equiv E \cap [< x] \end{aligned}$$

The first of these filters, in combination with the previous interval constructors, allows us to express the range of time values starting at some point x and ending right before the *next* occurrence of an event E.

$$[x..E] \equiv [x \leq] \cap [< (E_{x<})]$$

Generalizing the previous notation to two events E and F, we end up with an operator that captures a set of ranges, where E contains the possible starting points, and F the possible end-points.

$$[E..F] \equiv \{x : \exists y \in E : x \in [y..F]\}$$

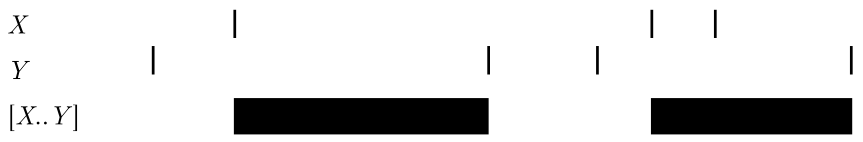

Fig. 4. Scenario illustrating the active ranges between two events.

Fig. 4 shows the intuition behind this range operator in graphical form.

As an inverse of the range operator, we may also define two operations that extract the set of starting points and end-points, respectively, from a set of disjoint intervals.

$$\begin{aligned} &E_\uparrow \equiv \{x \in E : \exists y < x : y \notin E \wedge \forall y' : y \leq y' < x \Rightarrow y' \notin E\} \\ &E_\downarrow \equiv (E^{\complement})_\uparrow \end{aligned}$$

Since the length of an interval is captured by the measure operator, we may introduce the relative distance between two time values as an arithmetic expression.

$$x - y \equiv \lambda([y..x]) - \lambda([x..y])$$

Note how the use of two swapped intervals makes the distance operator capable of returning both positive and negative results, depending on which of the times x and y that is greater.

The distance operator allows time translation of sets to be expressed, and the range notation to be generalized accordingly.

$$\begin{aligned} E \gg e &\equiv \{x : \exists y \in E : y - x = e\} \\ E \ll e &\equiv E \gg 0 - e \\ [x{+}e..y{+}f] &\equiv ([x \leq] \gg e) \cap ([< y] \gg f) \end{aligned}$$

We also provide an option for indexing a set of time values from zero and up. Since indexing must fail if the set in question contains too few elements, or if an index falls inside a continuous interval of elements, the indexing operator is integrated into a constraint form that simply becomes false under those circumstances.

$$x = E(e) \equiv x \in E \wedge |E_{<x}| = e$$

Another useful constraint form that is definable in terms of intervals is the subrange relation:

$$E \trianglelefteq F \equiv \exists x : \exists y : E = F \cap [x..y]$$

As a generic mechanism for open-ended extension, TiCL allows user-defined constraints to be named and placed in the available abbreviation environment

[3] We use a concrete syntax where quantifiers scope as far to the right as possible, and standard operator precedences apply.

alongside the notational short-hands introduced above. Each such constraint definition is of the form

$$C(\overline{x}, \overline{X}, \overline{v}) \equiv c$$

where C is a name drawn from some set of constraint identifiers, c is a constraint expression, and $\overline{x}$, $\overline{X}$, and $\overline{v}$ are zero or more distinct time, event, and arithmetic variables, respectively. A named constraint can be referred to by writing

$$C(\overline{y}, \overline{E}, \overline{e})$$

where the number of terms in $\overline{y}$, $\overline{E}$, and $\overline{e}$ must match the corresponding parameter lists in the definition of C.

To constitute a valid constraint definition, $C(\overline{x}, \overline{X}, \overline{v}) \equiv c$ must fulfill two conditions:

1. c must not contain any other free variables than those in $\overline{x}$, $\overline{X}$, and $\overline{v}$.
2. c must not refer to C, or any other constraint definition that directly or indirectly refers to C.

These conditions ensure that in any context, named constraints can be removed by simply macro-expanding their respective definitions.

5 Expressing TADL constraints

In this section we demonstrate the expressive power of TiCL by showing how the various timing constraints defined by both TIMMO and AUTOSAR can be captured formally. The intention is neither to explain the intuition behind these constraints here, nor to motivate any particular design choices in the definitions. In fact, some constraints are actually given multiple definitions, reflecting the alternatives that have appeared in different TIMMO or AUTOSAR versions. The focus in this section is primarily on the semantic details that distinguish such alternatives from each other.

The basic TADL *delay* constraint requires that for each occurrence of a stimulus event X, there must exist *some* occurrence of response event Y at a relative distance determined by a lower and an upper bound (v_l and v_u).

$$\mathit{delay}(X, Y, v_l, v_u) \equiv \forall x \in X : \exists y \in Y : v_l \leq y - x \leq v_u$$

Two *delay* constraints in a symmetric fashion form a *bidelay*. Such a constraint is not actually part of TADL, but we give it a name here nevertheless because it will prove useful in the specification of other TADL constraints.

$$\mathit{bidelay}(X, Y, v_l, v_u) \equiv \mathit{delay}(X, Y, v_l, v_u) \wedge \mathit{delay}(Y, X, -v_u, -v_l)$$

An alternative form of delay that will also be subsequently needed requires that each response occurrence is unique within the specified time window.

$$\mathit{unidelay}(X, Y, v_l, v_u) \equiv \forall x \in X : |Y \cap [x + v_l..x + v_u]| = 1$$

Yet another useful form is the *strong* delay, which demands that the stimulus and response events are related for each indexed occurrence.

$$\mathit{strongdelay}(X, Y, v_l, v_u) \equiv \forall i : \exists x = X(i) : \exists y = Y(i) : v_l \leq y - x \leq v_u$$

The differences between these delay forms are subtle but important. The basic *delay* allows multiple responses to a single stimuli, as well as responses that are shared by multiple stimuli. *bidelay* does the same, but disallows orphan responses. The *unidelay* constraint rules out multiple possible responses, but still allows the mapping of many stimuli onto a single response. *strongdelay* requires the stimulus and response occurrences to appear in lock-step.

TADL further defines a *repetition* constraint, which can be conveniently captured in two stages. First we introduce the basic notion of a repetition, which says that any stretch of v_s periods (i.e., any subrange of v_s+1 event occurrences) must have a distance between the first and last occurrence that is bounded by v_l and v_u.

$$\mathit{repeat}(X, v_l, v_u, v_s) \equiv \forall Y \trianglelefteq X : |Y| = v_s + 1 \Rightarrow v_l \leq \lambda([Y]) \leq v_u$$

Then we add the jitter component by means of a local event and *strongdelay*:

$$\mathit{repetition}(X, v_l, v_u, v_j, v_s) \equiv \exists Y : \mathit{repeat}(Y, v_l, v_u, v_s) \wedge \mathit{strongdelay}(Y, X, 0, v_j)$$

Notice how Y here takes the role of a set of ideal points in time, from which the actual event X may deviate by at most the jitter distance v_j.

The third pillar of TADL is the *synchronization* constraint, which in its weak form can be expressed as follows:

$$\mathit{sync}(X_1, \ldots, X_n, v_j) \equiv \exists Y : \mathit{bidelay}(Y, X_1, 0, v_j) \wedge \cdots \wedge \mathit{bidelay}(Y, X_n, 0, v_j)$$

That is, synchronization implies that each occurrence of each event X_i is sufficiently close to a "cluster" point of some set Y, and each such point in turn is sufficiently close to occurrences of all the X_i. Note that by choosing *bidelay* over the other delay forms in this definition, TADL deliberately accepts both overlapping synchronization clusters, and clusters containing more than one occurrence of some events. A strong synchronization variant, which requires all synchronized events to appear in a lock-step fashion akin to the *strongdelay* constraint, can easily be defined in terms of the latter (not shown here).

TADL has recently been extended with a constraint capturing the notion of bounded execution times, which is a bit challenging to formalize purely in terms of events. However, if one assumes the existence of events indicating not only the start and termination of the function of interest, but also preemption and resumption of that function, an *exectime* constraint can be defined quite succinctly in TiCL.

$$\mathit{exectime}(X, Y, X', Y', v_l, v_u) \equiv \forall x \in X : v_l \leq \lambda([x..Y] \backslash [X'..Y']) \leq v_u$$

This definition assumes that X and Y capture the points in time when the function of interest is started and terminated, and that preemption and resumption points for that function are given by events X' and Y', respectively. The set $[X'..Y']$ thus indicates the intervals during which the measured function is preempted, and those points in time should be excluded from each invocation interval in order to obtain an accurate execution time. The value to be constrained is the sum of the interval fragments that remain, which is equivalently expressed as the measure of the corresponding set. Fig. 5 shows a graphical illustration of an *exectime* scenario, where x_1 and x_2 denote the starting points of two separate invocations of the constrained function.

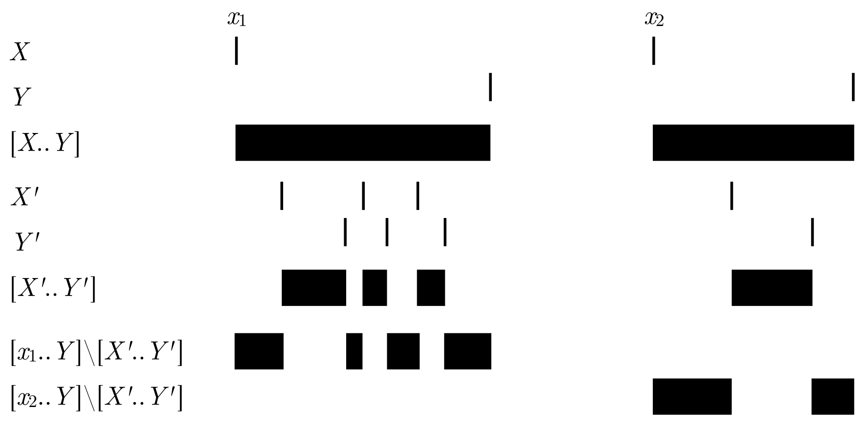

Fig. 5. Event scenario illustrating the *exectime* constraint.

$$\begin{aligned}
\mathit{sporadic}(X, v_l, v_u, v_j, v_m) &\equiv \mathit{repetition}(X, v_l, v_u, v_j, 1)\wedge \\
&\quad\ \mathit{minimum}(X, v_m) \\
\mathit{periodic}(X, v_p, v_j, v_m) &\equiv \mathit{sporadic}(X, v_p, v_p, v_j, v_m) \\
\mathit{pattern}(X, Y, v_1, \ldots, v_n, v_j, v_m) &\equiv \mathit{delay}(Y, X, v_1, v_1 + v_j) \wedge \cdots \wedge \\
&\quad\ \mathit{delay}(Y, X, v_n, v_n + v_j)\wedge \\
&\quad\ \mathit{minimum}(X, v_m) \\
\mathit{arbitrary}(X, v_1, \ldots, v_n, v'_1, \ldots, v'_n) &\equiv \mathit{repeat}(X, v_1, v'_1, 1) \wedge \cdots \wedge \\
&\quad\ \mathit{repeat}(X, v_n, v'_n, n) \\
\mathit{burst}(X, v_l, v_n, v_m) &\equiv \mathit{repeat}(X, v_l, \infty, v_n + 1)\wedge \\
&\quad\ \mathit{minimum}(X, v_m)
\end{aligned}$$

Fig. 6. Derived TADL constraint definitions.

Further TADL constraints are definable entirely in terms of the building blocks introduced so far. Fig. 6 shows the definitions that apply to TADL v2.[4] The *minimum* constraint referenced in several places is just a jitter-free repetition spanning subsequent occurrences, with infinity as its upper bound.

$$\mathit{minimum}(X, v) \equiv \mathit{repeat}(X, v, \infty, 1)$$

To further exemplify the precision that is possible to express using TiCL, we give a few alternative definitions of the *pattern* and *sync* constraints above. The *pattern1* variant represents one reasonable interpretation of the corresponding AUTOSAR constraint, which assumes that the underlying periodic cycle of the pattern is automatically detected. Also, constraint *sync1* captures the quite complicated definition of synchronization that was a part of TADL v1, where synchronization was not expressible without also embedding a delay from a reference event governing when synchronization must take place.

$$\begin{aligned}\mathit{pattern1}(X, v_p, v_j, v_m, v_1, \ldots, v_n) \equiv\ & \exists Y : \mathit{periodic}(Y, v_p, 0, 0) \wedge \\ & \mathit{pattern}(X, Y, v_1, \ldots, v_n, v_j, v_m)\end{aligned}$$

$$\begin{aligned}\mathit{sync1}(Y, X_1, \ldots, X_n, v_l, v_u, v_j) \equiv\ & \mathit{unidelay}(Y, X_1, v_l, v_u) \wedge \cdots \wedge \\ & \mathit{unidelay}(Y, X_n, v_l, v_u) \wedge \\ & \exists Y' : \mathit{strongdelay}(Y, Y', v_l, v_u) \wedge \\ & \mathit{sync}(Y', X_1, \ldots, X_n, v_j)\end{aligned}$$

6 Modes

Mode dependency is a design pattern that is used frequently in many AUTOSAR and EAST-ADL models, and which naturally also has an impact on the notion of timing correctness of such models. Simply put, a mode is an abstraction over the state of a system, such that at each point in time, the mode is either *active* or *inactive*. Modes are typically used to guard different functional behaviors, emphasizing orderly distributed mode transitions over distributed behavior in general. A mode-dependent timing constraint is then understood as a constraint that only has to hold while the referenced mode is active; outside those active intervals, the constraint should count as being vacuously true.

However, while the basic intuition behind modes is relatively simple, its application to timing correctness presents some interesting design problems. The fundamental challenge is that timing constraints express properties that generally involve multiple points in time, and a mode change that occurs in the middle of such an interval may very well render a mode-dependent constraint

[4] TADL v2 also includes a group of delay and synchronization constraints that use an externally provided *causality relation* to filter out the event occurences that that a particular delay or synchronization window should contain. TiCL can easily be complemented with the machinery necessary to express this extension, but we do not show it here in the interest of notational brevity.

ambiguous. A *delay* constraint serves as a simple illustration. Should an absent response be tolerated if a mode deactivation intervenes? Should a stray response be accepted if it *could* have been caused by a stimulus outside the current mode interval?

Examination of real world scenarios has led us to believe that the generic answer should be yes to all such questions. A mode-dependent constraint should be considered satisfied if it holds for the event occurrences within each active interval, *plus* some hypothetic and optimally chosen occurrence pattern outside each interval. If this idea is formalized correctly, it should be possible to put a mode-restriction on an arbitrary constraint and obtain a meaningful semantics, even if the constraint has not been defined with the specific challenges of mode-switching in mind.

The approach we have taken is to model modes as sets of time values, just like we do for events and arbitrary intervals. To make the mode intuition clear, however, we introduce a distinct class of variables to range over modes:

$$M \in \textbf{Svar} \text{ (mode identifiers)}$$

Semantically, a mode identifier M stands for some set of time values, just like an X or a Y. In particular, if X and Y are events indicating the activation and deactivation of some mode M, a natural way to express this formally is to introduce M in some scope c as follows:

$$\exists M = [X..Y] : c$$

Alternatively, a mode M can be defined as some combination of other modes using union, disjunction, or any other defined operator on general sets.

We now introduce a syntax for mode-restricted constraints, by means of a decoration to the application of a named constraint macro.

$$C(\overline{y}, E_1, \ldots, E_n, \overline{e})\%M$$

The core of our mode-restriction mechanism is the semantics given to this constraint form. As before, we proceed in terms of a translation of the syntax form, that results in a constraint term where the new syntax is absent. We begin with the simple case where C takes only one event argument.

$$C(\overline{y}, E, \overline{e})\%M \equiv \forall x \in M_{\uparrow} : \exists Y = [x..M_{\downarrow}] : \exists X \subseteq Y^{\mathsf{C}} : C(\overline{y}, (E \cap Y) \cup X, \overline{e})$$

The definition should be read as follows. For the given mode M, its activation and deactivation points ($M_{\uparrow}$ and $M_{\downarrow}$) are identified. Then, for each activation point x in $M_{\uparrow}$, a freely chosen X, subset of the possible time values *outside* the current activation interval Y, is added to the subrange of event parameter E that falls within Y. That is, the translated, mode-independent application of C takes $E \cap Y \cup X$ as an argument in place of E, which captures the intuition that a mode both ignores and assumes the best about occurrences that fall outside its active intervals. Generalized to n event arguments, the translation becomes

$$\begin{aligned} C(\overline{y}, E_1, \ldots, E_n, \overline{e})\%M \equiv \forall x \in M_{\uparrow} : \exists Y = [x..M_{\downarrow}] :& \\ \exists X_1 \subseteq Y^{\mathsf{C}} : \ldots : \exists X_n \subseteq Y^{\mathsf{C}} :& \\ C(\overline{y}, (E_1 \cap Y) \cup X_1, \ldots, (E_n \cap Y) \cup X_n, \overline{e})& \end{aligned}$$

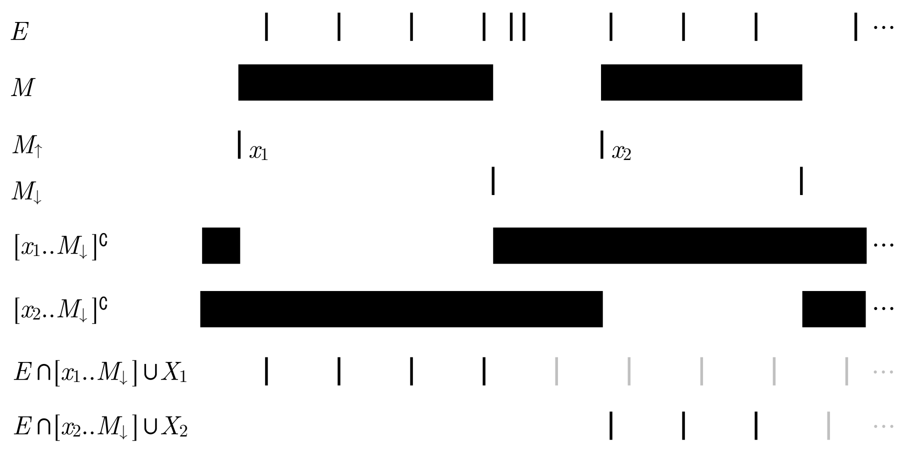
E
M
$M_\uparrow$
x_1
x_2
$M_\downarrow$
$[x_1..M_\downarrow]^\complement$
$[x_2..M_\downarrow]^\complement$
$E \cap [x_1..M_\downarrow] \cup X_1$
$E \cap [x_2..M_\downarrow] \cup X_2$

Fig. 7. Event scenario involving a mode-restricted constraint $\mathit{cyclic}(E, v)\%M$.

To illustrate the power of this interpretation of mode-dependencies, we define a somewhat contrived, but still perfectly sound, variant of a repetition constraint:

$$\mathit{cyclic}(X, v) \equiv \mathit{delay}(X, X, v, v)$$

This constraint is special because it only holds for infinitely repeating events.

The top of Fig. 7 shows the initial trace of an event E that certainly does not satisfy $\mathit{cyclic}(E, v)$ for any v. However, the intent is now that the constraint only has to hold while mode M is active; i.e., $\mathit{cyclic}(E, v)\%M$ must be true. While it is clear that E is repetitive during the activity intervals, just limiting E to those intervals would not work—all such E subsets would be finite. But by interpreting mode-restriction as a constraint on mode-limited event subsets *extended with arbitrarily chosen points outside the mode interval*, even infinitely demanding constraints like *cyclic* become possible to apply in finite contexts.

What Fig. 7 depicts below the trace of E is the assumed activity intervals of mode M, its activation and deactivation points, the complements of the first and second activity intervals of M (i.e., the sets from which suitable subsets X may be drawn), and the resulting, purely cyclic patterns that result when the relevant subsets of E are suitably extended.

7 Related Work

TiCL shares its main objective with the various real-time (timed/temporal) logics that have been proposed in the context of model-checking and verification of timed automata: to offer a comprehensive formalism for specifying the timing behavior of a system in a logically robust way [3, 7, 13, 12, 1, 9]. This line of research is uniformly dealing with *modal* logics; i.e., logics whose semantics is based on sequences of system states and atomic predicates on these. Such generality allows the integration of timing aspects into arbitrary specifications of functional behavior, as would be expected by the model-checking approach. TiCL exhibits a much weaker connection between timing properties and their underlying systems, by only allowing relations on the occurrences of abstract events as its atomic formulas. This makes TiCL unsuitable as a general model-checking specification language, although it also makes for a very clean identification of the properties that are purely concerned with timing.

At the same time, TiCL is fundamentally more expressive than the modal approaches in being a first-order logic. The additional power stems primarily from the universally (and existentially) quantified variables of TiCL, which may range over both points in time as well as sets of such values. Temporal logics allow only a limited form of quantification through temporal operators, whose closest counterparts in TiCL would be quantifiers introducing variables used just as event indices. A mode-dependency operator like ours, which critically depends on the ability to quantify over sets, appears very difficult to express in a temporal logic style, if at all possible. It should also be noted that the core TiCL operators and quantifiers are entirely standard in the logic field, whereas the various temporal and real-time logics are to a large extent identified by the custom operators they provide. Of course, TiCL pays a price for this generality by being undecidable, but its intended role as a disambiguation tool for humans is more dependent on a standard semantics and a carefully delimited syntax than on decidability issues. Moreover, there are reasons to believe that practically significant fragments of TiCL are indeed decidable, analogous to the case for first-order logics in general.

Amon et al. [4] define a specification language for capturing the logic of timing diagrams in a form that resembles our constraint language minus the event variables and with restricted integer arithmetic (no division operator, multiplication by literals only). This sublanguage corresponds to Presburger formulas, for which automatic and efficient verification procedures exist. It remains to be seen to what extent the approach allows extension towards the full TiCL syntax (one particular sub-case that appears particularly benign is top-level quantification over real-valued sets, which should imply little more than just iterated verification).

CCSL [5] is a language for specifying timing constraints in the UML profile MARTE [14] for modeling and analysis of real-time systems. CCSL can specify *clocks*, which correspond to events, and relations between them. Relations include various sub- and precedence constraints. It seems that TiCL quite readily could express counterparts to CCSL clock constraints on events.

Timed automata [2] are automata extended with various clock variables, which can be used to model real-time systems. Model checking can be performed over timed automata to verify that the models have certain properties. The properties, which typically are reachability properties, are then specified in some temporal logic. UPPAAL [11] is a well-known tool for modeling and verification using timed automata.

Transitions in timed automata can be guarded by constraints on the clocks: thus, timed automata can to some extent include timing constraints in the models. However, the style is state-oriented rather than event-oriented and thus quite different from TiCL. Also, timed automata and their temporal logics are usually designed to be decidable, allowing efficient procedures for model checking whereas TiCL favours expressiveness. An interesting question, of course, is whether some nontrivial fragment of TiCL can be translated into timed automata as it would allow automatic verification of that fragment.

8 Conclusions and Further Research

We have presented TiCL, a simple logic for expressing timing constraints on events. TiCL came out of the work with TADL, a language for specifying timing requirements and -properties that is intended to be used with AUTOSAR and EAST-ADL in the automotive domain. TiCL offers a rich syntax, on top of a simple kernel language, for defining constraints on events defined as sets of times. We showed how to express TADL's timing constraints in TiCL, as well as some other timing constraints that seem natural and useful. We also introduced mode-dependent TiCL constraints, with a special mode restriction operator, and gave the operator a semantics by translation into TiCL without this operator.

Expressing timing constraints by translation into a logic like TiCL has several advantages. One advantage is that the semantics of timing constraints becomes well-defined and unambiguous, since TiCL itself has a very clear, standard semantics. In particular this is true for mode-dependent constraints, since they have hitherto never been given a stringent semantics although they are present in both AUTOSAR, EAST-ADL, and TADL. We believe that we have found the "right" definition of mode dependency, and the semantics of this definition can be used to give a well-defined semantics for mode dependency in, say, TADL as well.

Another advantage is that tools for validating or verifying timing constraints can work by translation into TiCL. Tools that check the validity of event traces vis-a-vis some timing constraints, or that simulate systems based on timing properties expressed in, say, TADL, can work on TiCL rather than TADL. Since TiCL is simple, with a clear semantics, such tools will be easier to implement for TiCL. This is similar to compilers, where programs are first translated into some intermediate format that is easier to work on.

TiCL also offers a way to express timing constraints in situations where the fixed format constraints of languages like TADL turn out not to be applicable. A "power user" can easily define new timing constraints in TiCL that are tailored to special needs. Similarly, if later versions of AUTOSAR or TADL will have a modified set of timing constraints, then these will most likely be expressible in TiCL as well. Once a translation to TiCL is established the new constraints will have a well-defined semantics, and tools that are based on TiCL will work immediately.

Finally, TiCL opens a possible route for formal verification of timing constraints. Although TiCL itself is not a decidable logic, it does not seem unlikely that there are nontrivial fragments that are decidable. Timing constraints that can be expressed within such fragments will then be possible to verify formally by an automated decision procedure.

References

1. Abadi, M., Lamport, L., Taylor, R.W.: An old-fashioned recipe for real time. In: ACM Transactions on Programming Languages and Systems. pp. 1–27. Springer-Verlag (1992)
2. Alur, R., Courcoubetis, C., Dill, D.: Model-checking for real-time systems. In: Proc. Logic in Computer Science. pp. 414–425. IEEE (Jun 1990)
3. Alur, R., Henzinger, T.A.: A really temporal logic. J. ACM 41(1), 181–203 (Jan 1994)
4. Amon, T., Borriello, G., Hu, T., Liu, J.: Symbolic timing verification of timing diagrams using Presburger formulas. In: Proc. 34th annual Design Automation Conference. pp. 226–231. ACM, New York, NY, USA (1997)
5. André, C., Mallet, F.: Clock constraints in UML/MARTE CCSL. Research report, INRIA (May 2008)
6. Homepage of the AUTOSAR project (2009), `www.autosar.org`
7. Chaochen, Z., Hoare, C.A.R., Ravn, A.P.: A calculus of durations. Inf. Process. Lett. 40(5), 269–276 (1991)
8. Cuenot, P., Frey, P., Johansson, R., Lönn, H., Papadopoulos, Y., Reiser, M.O., Sandberg, A., Servat, D., Kolagari, R.T., Törngren, M., Weber, M.: The EAST-ADL architecture description language for automotive embedded software. In: Giese, H., Karsai, G., Lee, E., Rumpe, B., Schätz, B. (eds.) Model-Based Engineering of Embedded Real-Time Systems. Lecture Notes in Comput. Sci., vol. 6100, pp. 297–308. Springer-Verlag, Schloss Dagstuhl, Germany (Nov 2007)
9. Grüninger, M., Menzel, C.: The process specification language (PSL) theory and applications. AI Mag. 24(3), 63–74 (Sep 2003)
10. Johansson, R., Frey, P., Jonsson, J., Nordlander, J., Pathan, R.M., Feiertag, N., Schlager, M., Espinoza, H., Richter, K., Kuntz, S., Lönn, H., Kolagari, R.T., Blom, H.: TADL: Timing augmented description language, version 2. Technical report (Oct 2009)
11. Larsen, K.G., Pettersson, P., Yi, W.: UPPAAL in a nutshell. Int. Journal on Software Tools for Technology Transfer 1, 134–152 (1997)
12. Mattolini, R., Nesi, P.: An interval logic for real-time system specification. IEEE Trans. Softw. Eng. 27(3), 208–227 (Mar 2001)
13. Moszkowski, B.: A temporal logic for multilevel reasoning about hardware. Computer 18, 10–19 (1985)
14. UML profile for MARTE: Modeling and analysis of real-time embedded systems. Tech. rep., OMG (Nov 2009), `www.omg.org/spec/MARTE/1.0`
15. Homepage of the TIMMO-2-USE project (2012), `www.timmo-2-use.org`