Verifying Event-Based Timing Constraints by Translation Into Presburger Formulae

Björn Lisper[1]

School of Innovation, Design, and Engineering, Mälardalen University, SE-721 23 Västerås, Sweden

Abstract. Abstract modeling of timing properties is often based on events. An event can be seen as a sequence of times. Timing constraints can then be expressed as constraints on events: an example is the TADL2 language that has been developed in the automotive domain.
Event-based constraints can express timing properties of implementations as well as timing requirements. An important step in timing verification is then to show that any events that comply with the properties of the implementation, i.e., that describe the timings of its possible behaviours, also satisfy the requirements.
Real-time software is often organised as a set of periodically repeating tasks, especially in domains with time-critical systems like automotive and avionics. This implementation naturally yields periodic events, where each event occurrence belongs to a periodically repeating time window. An interesting question is then: if some events are periodic in this fashion, will they then fulfil a timing constraint that describes a timing requirement? We show, for a number of TADL2 timing constraints, how to translate this implication into an equivalent Presburger formula. Since Presburger logic is decidable, this yields an automated method to decide whether the periodic implementation satisfies the timing requirements or not. Initial experiments with a Presburger solver indicate that the method is practical.

1 Introduction

Timing behavior descriptions exist in many different forms. Classical real-time scheduling theory defines the basic *periodic* [18] and *sporadic* [19] 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 [12] 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.

An important class of timing behavior descriptions is based on *events*: examples are TADL2 [10], a revised version of the *Timing Augmented Description Language* (TADL) [15] that forms the basis for timing specifications in AUTOSAR and EAST-ADL, and the CCSL language of the UML real-time profile MARTE [5]. Events are sequences (or sets) of times. A rich variety of timing properties, for single as well as multiple events, can be expressed in this fashion. An advantage with this way of describing timing properties is that it abstracts away from the underlying system by describing its possible timing behaviours through constraints on events. Once this is done, it can be checked in the event domain whether the system fulfils its timing requirements or not. If the possible timing behaviours of the system are described by a predicate *impl* on the events $e^1, \ldots, e^n$, and if the requirements are expressed by the predicate *req* on the same events, then the property that the system fulfils its timing requirements is expressed by the formula

$$\forall e^1, \ldots, e^n.[impl(e^1, \ldots, e^n) \Rightarrow req(e^1, \ldots, e^n)] \tag{1}$$

We have studied a case where formulae of this kind can be decided. Systems are often implemented in a fashion that gives rise to *periodic* events, where each event occurrence belongs to a regularly repeating time window of fixed size. The class of periodic events has certain mathematical properties that, when the *impl* predicate in (1) is expressed as a conjunction of periodic event constraints, allows many instances of (1) to be translated into an equivalent Presburger formula. We exemplify this by translating a number of instances of (1), where $req(e^1, \ldots, e^n)$ is given by different TADL2 constraints, into equivalent Presburger formulae. Since Presburger logic is decidable, this yields a route to automatic verification of these instances.

An important case where periodic events appear is for *periodic preemptive fixed priority based scheduling*, where real-time tasks are triggered periodically and higher priority tasks can preempt lower-priority tasks. The time windows for events marking the completions of such tasks can be established by a best- and a worst-case response-time analysis, well-known from classical real-time scheduling theory. The task model is very common in areas like automotive and avionics, and many real-time operating systems implement this scheduling policy.

In the widely used AUTOSAR standard [6] for development of automotive software, the smallest software entities that can be associated with events are *runnables*. These are grouped into tasks, which can be executed by the AUTOSAR Basic Software Layer according to this scheduling policy. Events arising from runnables will then be periodic. Timing requirements can be expressed over these events using constraints from the *AUTOSAR Timing Extensions* [7], which are directly based on the TADL timing constraints. This opens the possibility to verify timing constraints for AUTOSAR software automatically using our approach.

A concern, however, is the potentially very high complexity for deciding Presburger formulae. This could render the verification method impractical. We have

performed some simple experiments with the the `iscc` calculator[1], which can handle general Presburger formulae. In all cases, the translated formulae were solved instantaneously. This indicates that the method may indeed be practical.

The rest of this paper is organised as follows. In Section 2 we define events, and introduce a syntax for timing constraints as a simple logic "TiCS" for sequences of times. Section 3 introduces TADL2, and we define the TADL2 constraints in TiCS. In Section 4 we show how to translate statements of form (1) into Presburger formulae, and we prove the equivalence of the translated formula for some typical cases where the events are periodic and the requirements are expressed as TADL2 constraints. In Section 5 we give an account for some initial experiments with a Presburger solver. We discuss related work in Section 6, and the paper is concluded with some reflections on future work in Section 7.

2 Events

Definition 1. *An* event e *is a strictly increasing, possibly infinite sequence of times* $\langle e_0, e_1, \ldots \rangle$. *Each time* e_i *is an* occurrence *of the event.*

We consider times to be integers. This is not a serious restriction: all results shown here are also valid for events with real-valued occurrences. For a *periodic* event, each occurrence belongs to a regularly appearing, fixed size time window:

Definition 2. *An event* $\langle e_0, e_1, \ldots \rangle$ *is* periodic *with* start time t_s, jitter $j \geq 0$, *and* periodicity $p > j$, *iff for all* $i \geq 0$ *holds that* $t_s + i \cdot p \leq e_i \leq t_s + i \cdot p + j$. *We write* $Per(e, t_s, p, j)$ *to denote that* e *is a periodic event with start time* t_s, *periodicity* p, *and jitter* j.

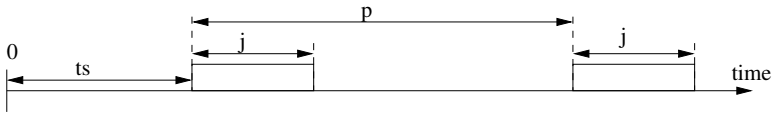

Fig. 1. Time windows for a periodic event.

Periodic events with jitter correspond to the periodic task model with output jitter [8]. Fig. 1 provides an illustration of the time windows to which the occurrences of a periodic event must belong.

We define a simple, formal syntax for constraints on events in the form of a first-order logic, see Fig. 2, where we also give a standard denotational semantics with semantic functions mapping expressions and environments "ρ" to values. We label the logic "TiCS" ("Timing Constraints for Sequences"). It is a variation

[1] `https://dtai.cs.kuleuven.be/cgi-bin/barvinok.cgi`

of the event logic "TiCL" [17], which has been used to give a formal semantics to the TADL2 timing constraints: the main difference between TiCS and TiCL is that in TiCL events are sets of times, whereas TiCS defines events as sequences of times.

TiCS allows timing constraints to be expressed as conditions on arithmetic expressions involving event occurrences. There are three kinds of variables: event variables e, arithmetic variables t, and index variables i. Event occurrences are of the form e_{i+n}, where n is a natural number. Quantification can be done over all three kinds of variables.

$$
\begin{array}{ll}
n \in \mathbb{N} \text{ (natural numbers)} & o \in \mathbf{Eocc} \text{ (event occurrences)} \\
z \in \mathbb{Z} \text{ (integers)} & t \in \mathbf{Avar} \text{ (arithmetic variables)} \\
e \in \mathbf{Evar} \text{ (event variables)} & a \in \mathbf{AExpr} \text{ (arithmetic expressions)} \\
i \in \mathbf{Ivar} \text{ (index variables)} & c \in \mathbf{CExpr} \text{ (constraint expressions)}
\end{array}
$$

$$
\begin{array}{l}
o \to e_{i+n} \\
a \to z \mid t \mid i \mid o \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 \cdot a_2 \mid a_1/a_2 \\
c \to T \mid F \mid a_1 \geq a_2 \mid c_1 \wedge c_2 \mid c_1 \vee c_2 \mid \neg c \mid \forall e.c \mid \forall i.c \mid \forall t.c \mid \exists e.c \mid \exists i.c \mid \exists t.c
\end{array}
$$

$$
\begin{array}{l}
\epsilon \in \mathbf{Event} = \mathbb{N} \to \mathbb{Z} \quad \text{(events)} \\
\rho \in \mathbf{Env} = (\mathbf{Avar} \to \mathbb{Z}) \cup (\mathbf{Ivar} \to \mathbb{N}) \cup (\mathbf{Evar} \to \mathbf{Event}) \quad \text{(environments)} \\
\mathcal{A} \in \mathbf{AExpr} \to \mathbf{Env} \to \mathbb{Z} \\
\mathcal{C} \in \mathbf{CExpr} \to \mathbf{Env} \to \mathbb{B}
\end{array}
$$

$$
\begin{array}{rl}
\mathcal{A}[\![z]\!]\rho = z & \mathcal{C}[\![a_1 \geq a_2]\!]\rho = \mathcal{A}[\![a_1]\!]\rho \geq \mathcal{A}[\![a_2]\!]\rho \\
\mathcal{A}[\![t]\!]\rho = \rho(t) & \mathcal{C}[\![c_1 \wedge c_2]\!]\rho = \mathcal{C}[\![c_1]\!]\rho \wedge \mathcal{C}[\![c_2]\!]\rho \\
\mathcal{A}[\![i]\!]\rho = \rho(i) & \mathcal{C}[\![c_1 \vee c_2]\!]\rho = \mathcal{C}[\![c_1]\!]\rho \vee \mathcal{C}[\![c_2]\!]\rho \\
\mathcal{A}[\![e_{i+n}]\!]\rho = \rho(e)(\rho(i) + n) & \mathcal{C}[\![\neg c]\!]\rho = \neg(\mathcal{C}[\![c]\!]\rho) \\
\mathcal{A}[\![a_1 \oplus a_2]\!]\rho = \mathcal{A}[\![a_1]\!]\rho \oplus \mathcal{A}[\![a_2]\!]\rho, & \mathcal{C}[\![\forall e.c]\!]\rho = \forall \epsilon. \mathcal{C}[\![c]\!]\rho[e \mapsto \epsilon] \\
\quad \oplus \in \{+, -, \cdot, /\} & \mathcal{C}[\![\exists e.c]\!]\rho = \exists \epsilon. \mathcal{C}[\![c]\!]\rho[e \mapsto \epsilon] \\
\mathcal{C}[\![T]\!]\rho = T & \mathcal{C}[\![\forall i.c]\!]\rho = \forall n. \mathcal{C}[\![c]\!]\rho[i \mapsto n] \\
\mathcal{C}[\![F]\!]\rho = F & \mathcal{C}[\![\exists i.c]\!]\rho = \exists n. \mathcal{C}[\![c]\!]\rho[i \mapsto n] \\
 & \mathcal{C}[\![\forall t.c]\!]\rho = \forall z. \mathcal{C}[\![c]\!]\rho[t \mapsto z] \\
 & \mathcal{C}[\![\exists t.c]\!]\rho = \exists z. \mathcal{C}[\![c]\!]\rho[t \mapsto z]
\end{array}
$$

Fig. 2. Syntactic categories, abstract syntax, and semantics

We will make free use of derived operators like $\Rightarrow$, $>$, $=$, $\neq$, which are definable in the language. We will write e_i for e_{i+0}. We will write $a \leq a' \leq a''$ for $a \leq a' \wedge' a \leq a''$. We will sometimes use set inclusion $x \in S$ when this formula can be expressed as a predicate in TiCS: for instance, given an interval $[l, u]$ we may write $x \in [l, u]$ for $l \leq x \leq u$. We will use the the shorthands $\forall p(x).c$ and $\exists p(x).c$ for $\forall x.(p(x) \Rightarrow c)$ and $\exists x.(p(x) \wedge c)$, respectively. We will allow ourselves the use of the infinity symbol "∞" in lieu of integers, when the semantics is clear: for instance, $\infty \geq z$ will always be true whenever $z \in \mathbb{Z}$. Using this notation we can express the property of being a periodic event as the following constraint:

$$Per(e, t_s, p, j) = \forall i \geq 0.[t_s + i \cdot p \leq e_i \leq t_s + i \cdot p + j]$$

When defining TADL2 constraints below we will allow nonrecursive "macros" defined in this way to appear in the formulae: their semantics can be defined by simple substitution. Finally we will use metanotation like "$e^1, \ldots, e^n$", or "$c_1 \wedge \cdots \wedge c_n$", to describe a varying number of arguments, or expressions.

TiCS, being a first-order logic containing basic arithmetics, is undecidable. Presburger arithmetic is a decidable fragment.

3 TADL2

The *Timing Augmented Description Language* (TADL2) [10] is a constraint language for describing timing requirements and properties within the automotive domain. It was originally defined in the TIMMO project, and was subsequently revised and formalised within the TIMMO-2-USE project[2]. The syntax of TADL is compliant to the AUTOSAR meta-model, but the TADL2 constraints can also be understood through a textual syntax.

TADL2 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.

The TADL2 constraints 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 *synchronisation constraints*, which 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*. This constraint is specified by four parameters *lower*, *upper*, *jitter*, and *span* where $span > 0$. An event $\langle t_0, t_1, \ldots \rangle$ satisfies a generic repetition rate constraint iff there exists a sequence of times $\langle x_0, x_1, \ldots \rangle$ such that for all $i \geq 0$,

$$x_i \leq t_i \leq x_i + jitter, \quad \text{and} \quad lower \leq x_{i+span} - x_i \leq upper$$

A *periodic* repetition constraint is a generic repetition rate constraint where $span = 1$, and $lower = upper$. This uniquely decides x_i to be $x_0 + i \cdot lower$, and we can write the constraint as $\exists x_0.Per(\langle t_0, t_1, \ldots \rangle, x_0, lower, jitter)$ with Per given by Definition 2. A *sporadic* repetition constraint has $span = 1$, and $upper = \infty$. TADL2 also defines more complex *pattern* repetition constraints, and *arbitrary* repetition constraints, see [10].

(The reason why we define a slightly different periodic constraint "Per" in Section 2 is that the TADL2 *Periodic* constraint is too weak to allow the results that we prove in Section 4. These results rely on knowledge about the relative offsets of periodic events, and this information is not present for the *Periodic* constraint.)

Delay constraints relate two events, called stimulus and response, by demanding that each occurrence of the stimulus is matched by at least one occurrence

[2] `https://itea3.org/project/timmo-2-use.html`

of the response within some time window. The basic delay constraint takes the parameters *lower*, and *upper*, and relates the stimulus event $\langle s_0, s_1, \ldots \rangle$ and the response event $\langle r_0, r_1, \ldots \rangle$ through the following constraint: for all i there exists a j such that

$$s_i + lower \leq r_j \leq s_i + upper$$

Synchronisation constraints concern a group of events S, characterised by a single parameter *tolerance*. The basic synchronisation constraint is fulfilled if there are time windows of size *tolerance* such that (1) each time window contains at least one occurrence of each event in S, and (2) there are no "spurious" event occurrences outside these windows. In other words, this constraint is satisfied iff there is a sequence of times $\langle x_0, x_1, \ldots \rangle$ such that (1) for all events $\langle s_0, s_1, \ldots \rangle \in S$ and for all i there exists a j such that

$$x_i \leq s_j \leq x_i + tolerance$$

and (2) for all i there exists a j such that

$$s_i - tolerance \leq x_j \leq s_i$$

Following [10, 17] the twelve most important TADL2 constraints are expressed below in TiCS. First, the repetition rate constraints:

$$\begin{aligned}
Repeat(e, l, u, s) &= \forall i \geq 0.[l \leq e_{i+s} - e_i \leq u] \quad (s > 0) \\
Repetition(e, l, u, s, j) &= \exists e'.[Repeat(e', l, u, s) \wedge StrongDelay(e', e, 0, j)] \\
Sporadic(e, l, u, j, m) &= Repetition(e, l, u, 1, j) \wedge Repeat(e, m, \infty, 1) \\
Periodic(e, p, j, m) &= Sporadic(e, p, p, j, m) \quad (p > 0) \\
Pattern(e, p, o_1, \ldots, o_n, j, m) &= \exists e'.[Periodic(e', p, 0, 0) \wedge Repeat(e, m, \infty, 1) \wedge \\
&\qquad Delay(e', e, o_1, o_1 - j) \wedge \\
&\qquad \ldots \\
&\qquad Delay(e', e, o_n, o_n - j)] \\
Arbitrary(e, l_1, \ldots, l_n, & \\
u_1, \ldots, u_n) &= Repeat(e, l_1, u_1, 1) \wedge \cdots \wedge Repeat(e, l_n, u_n, n) \\
Burst(e, l, o, m) &= Repeat(e, l, \infty, o) \wedge Repeat(e, m, \infty, 1)
\end{aligned}$$

Then, the delay constraints:

$$\begin{aligned}
Delay(e, e', l, u) &= \forall i \geq 0.\exists k \geq 0.[l \leq e'_k - e_i \leq u] \\
StrongDelay(e, e', l, u) &= \forall i \geq 0.[l \leq e'_i - e_i \leq u] \\
Order(e, e') &= \forall i \geq 0.[e_i < e'_i]
\end{aligned}$$

Finally, the synchronisation constraints:

$$\begin{aligned}
Synch(e^1, \ldots, e^n, w) &= \exists e'.[Delay(e', e^1, 0, w) \wedge Delay(e^1, e', -w, 0) \wedge \\
&\qquad \ldots \\
&\qquad Delay(e', e^n, 0, w) \wedge Delay(e^n, e', -w, 0)] \\
StrongSynch(e^1, \ldots, e^n, w) &= \exists e'.[StrongDelay(e', e^1, 0, w) \wedge \\
&\qquad \ldots \\
&\qquad StrongDelay(e', e^n, 0, w)]
\end{aligned}$$

In addition, TADL2 contains five constraints that cannot always be expressed in TiCS: an *execution time constraint*, and four constraints that are variations of basic constraints but where these constraints are restricted to hold only between certain event occurrences as specified by an auxiliary *causality relation* on these. As TADL2 does not define the nature of this relation further, we cannot guarantee that these constraints are always expressible in TiCS. However note that if the causality relation can be expressed in the TiCS syntax, then the full constraint can be as well and the verification machinery developed here can be applied. We will not consider these constraints further here: see [10] for details.

In addition to the constraints TADL2 also allows timing expressions to be symbolic. Symbolic variables can be defined, or constrained, and used in timing constraints. Typical usages are to parameterise timing requirements for easy update, to constrain the ranges of parameters, and to aid time budgeting by specifying bounds on sums of delays. TiCS supports symbolic timing expressions right away, and constraints on symbolic variables can simply be conjoined with the timing constraints.

4 Transforming TADL2 Constraints into Presburger Formulae

We will now show how to transform statements of the form (1), where the antecedent specifies events to be periodic according to Definition 2, and the consequent is chosen from a selection of TADL2 constraints, into equivalent Presburger formulae. The correctness of the transformation depends on a certain property of the repeating time windows for the periodic events. We now define this property, and show that it holds for these time windows.

Definition 3. *Let $R = \langle R_0, R_1, \ldots \rangle$ be a sequence of sets of times. Let E be a set of events.*

1. *R is a* sequence of regions *for E iff for all $i \geq 0$, and all $e \in E$, holds that $e_i \in R_i$.*
2. *R is* tight *for E iff it is a sequence of regions for E, and for any event e, where $e_i \in R_i$ for all $i \geq 0$, holds that $e \in E$.*

Lemma 1. *If R is tight for E, then $e \in E \iff \forall i.e_i \in R_i$.*

Proof. $\Rightarrow$: since if R is tight for E, then R is a sequence of regions for E. $\Leftarrow$: by the definition of tightness.

Let us write $win(t_s, p, j, i)$ for the ith time window $[t_s + i \cdot p, t_s + i \cdot p + j]$ containing the ith occurrence of a periodic event according to Definition 2.

Lemma 2. $\{\, win(t_s, p, j, i) \mid i \geq 0 \,\}$ *is tight for* $\{\, e \mid Per(e, t_s, p, j) \,\}$.

Proof. Immediate from Definition 2.

A tight sequence of regions fully characterises its set of events: each event occurrence belongs to the corresponding region, and any event in the set can be generated by picking a time from each region as the corresponding occurrence.

The transformation into a Presburger formula follows a common pattern. The first step is to transform the consequent in (1), i.e., the formula specifying the requirements, by replacing event occurrences with arithmetic variables to obtain a formula free of such occurrences:

- If the consequent contains the term $\forall i.C(e_{i+n})$, where $Per(e, t_s, p, j)$, then this term is replaced by $\forall i.\forall t \in win(t_s, p, j, i+n).C(t)$. That is: e_{i+n} is replaced by the arithmetic variable t throughout, where t ranges over the interval of e_{i+n}.
- If there are several distinct event occurrences indexed by the same quantified index variable, then each occurrence is replaced by a distinct arithmetic variable ranging over its interval. For instance, $\forall i.C(e_{i+n}, e'_{i+m})$ is translated into $\forall i.\forall t \in win(t_s, p, j, i+n).\forall t' \in win(t'_s, p', j', i+m).C(t, t')$ (given that $Per(e, t_s, p, j)$, and $Per(e', t'_s, p', j')$).
- Terms with existentially quantified index variables are transformed in the same way.

We now give a formal definition. The translation is defined relative to a function R mapping event variables to sequences of regions, and we denote it "P_R". We define it for a fragment of TiCS without quantification over events, and where formulas w.l.o.g. are in prenex normal form (all quantifiers are at the outermost level). The first restriction is important, whereas the second is merely for convenience as it allows a more succinct definition of the translation. In Fig. 3 we define the syntax of this fragment.

$$
\begin{array}{lll}
a & \to z \mid t \mid i \mid e_{i+n} \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 \cdot a_2 \mid a_1/a_2 & \textbf{AExpr} \\
c^- & \to T \mid F \mid a_1 \geq a_2 \mid c_1^- \wedge c_2^- \mid c_1^- \vee c_2^- \mid \neg c^- & \textbf{CExpr}^- \\
c & \to c^- \mid \forall t.c \mid \exists t.c \mid \forall i.c \mid \exists i.c & \textbf{CPExpr}
\end{array}
$$

Fig. 3. Abstract syntax for restricted constraint expressions

First we introduce some notation. $FVI(c)$ denotes the set of index variables that are free in c:

Definition 4. *For any expression c and index variable i, $Eo(c, i)$ is the set of event occurrences in c of the form e_{i+n}, for some event variable e and natural number n. Furthermore $Eo(c) = \bigcup_{i \in FVI(c)} Eo(c, i)$.*

For instance, if $r, s \in \mathbf{Evar}$, then $Eo(s_i + l \leq r_j \leq s_i + u, i) = \{s_i\}$, $Eo(s_i + l \leq r_j \leq s_i + u, j) = \{r_j\}$, $Eo(s_i + l \leq r_j \leq s_i + u) = \{s_i, r_j\}$, and $Eo(e_i \leq e_{i+1}, i) = \{e_i, e_{i+1}\}$.

Next, we assume a function $t\colon \mathbf{Eocc} \to \mathbf{Avar}$ that maps each event occurrence e_{i+n} to a syntactic arithmetic variable $t(e_{i+n}) \in \mathbf{Avar}$. Our transformation replaces each e_{i+n} with $t(e_{i+n})$. We assume that all variables $t(e_{i+n})$ are fresh.

Definition 5. *(Substitution of event occurrences) Let* $c^- \in \mathbf{CExpr}^-$*, and let EO be a set of event occurrences: then* $c^-[e_{i+n} \leftarrow t(e_{i+n}) \mid e_{i+n} \in EO]$ *is the expression resulting when every occurrence of* e_{i+n} *in* c*, where* $e_{i+n} \in EO$*, is concurrently replaced by the variable* $t(e_{i+n})$*.*

(We could make a fully formal, recursive definition over the structure of c^-.) Basically this is a first order substitution, the only difference being that we replace syntactic event instances e_{i+n} rather than single variables. Since there never can be any overlaps between the expressions e_{i+n} to be replaced, and since the substitution does not introduce such expressions, this kind of substitution is well-defined. We need some more meta-notation to define the translation:

Definition 6. *For any finite set of event occurrences* $EO = \{o_1, \ldots, o_n\}$*, arithmetic variables* $t(o_1), \ldots, t(o_n)$*, sets of integers* $T(o_1), \ldots, T(o_n)$*, and constraint expression* c*, we define:*

$$\forall(\, t(o) \in T(o) \mid o \in EO\,).c = \\ \forall t(o_1) \cdots \forall t(o_n).(t(o_1) \in T(o_1) \wedge \cdots \wedge t(o_n) \in T(o_n) \;\Rightarrow\; c)$$

Thus, the meta-notation denotes a formula where the variables $t(o_1), \ldots, t(o_n)$ are universally quantified over c while ranging over the sets $T(o_1), \ldots, T(o_n)$.

We now define our transformation P_R for expressions in **CPExpr**. We assume that for each $e \in \mathbf{Evar}$ there is a set of events $E(e)$ such that $R(e)$ is a sequence of regions for $E(e)$:

Definition 7.

$$\begin{aligned} P_R(c^-) &= c^-[e_{i+n} \leftarrow t(e_{i+n}) \mid e_{i+n} \in Eo(c^-)], \quad c^- \in \mathbf{CExpr}^- \\ P_R(\forall t.c) &= \forall t.P_R(c) \\ P_R(\exists t.c) &= \exists t.P_R(c) \\ P_R(\forall i.c) &= \forall i.\forall(\, t(e_{i+n}) \in R(e)_{i+n} \mid e_{i+n} \in Eo(c,i)\,).P_R(c) \\ P_R(\exists i.c) &= \exists i.\forall(\, t(e_{i+n}) \in R(e)_{i+n} \mid e_{i+n} \in Eo(c,i)\,).P_R(c) \end{aligned}$$

It is clear that $P_R(c)$ will contain no event occurrences. If all numerical subexpressions in c are linear in the index variables, arithmetic variables and event occurrences, and if the set memberships $t(e_{i+n}) \in R(e)_{i+n}$ can be expressed as Presburger formulae, then $P_R(c)$ will be a Presburger formula. For instance, if $i, k \in \mathbf{Ivar}$ and $r, s \in \mathbf{Evar}$ then

$$P_R(\forall i.\exists k.(s_i{+}l \le r_k \le s_i{+}u)) = \forall i.\forall t \in R(s)_i.\exists k.\forall t' \in R(r)_k.(t{+}l \le t' \le t{+}u)$$

Here, we have written t for $t(s_i)$ and t' for $t(r_k)$. If the conditions $t \in R(s)_i$ and $t' \in R(r)_k$ can be expressed by Presburger formulae then the transformed formula is also a Presburger formula.

The translation does not handle terms with quantified events. Such events appear in the definitions of the *Repetition*, *Pattern*, *Synch*, and *StrongSynch*

constraints. To translate such constraints, the quantified events have to be eliminated. For the *Repetition* constraint, if $l = u = p$ and $s = 1$ then the constraint $Repeat(e', p, p, 1)$ will uniquely determine $e'_i = e'_0 + i \cdot p$. Then, the quantification over e' can be replaced by a quantification over an arithmetic variable e'_0, and e'_i can be replaced by $e'_0 + i \cdot p$ throughout. This situation appears for the *Periodic* constraint, which calls *Repetition* through *Sporadic*.

For the synchronisation constraints, *Synch*, and *StrongSynch*, the instances e'_i of the existentially quantified event e' specify starting times of time windows of size w, where some occurrences of all events $e^1, \ldots, e^n$ belong to each window, and no occurrence of any event is outside such a time window. For these constraints, the existence of such a window can instead be expressed by the condition that for all pairs of synchronised events, all distances between the involved event occurrences are less than or equal to the size w of the window. We omit the details.

By eliminating quantified events in the manner described above, all constraints listed in Section 3 except the general *Sporadic* and *Repetition* constraints can be written in a form where (1) can be translated into a formula free from event occurrences according to above. What remains is to prove the *correctness* of the translation. We exemplify by proving the correctness for the *Delay* constraint.

Theorem 1.

$$\forall e, e'.[Per(e, t_s, p, j) \wedge Per(e', t'_s, p', j') \Rightarrow Delay(e, e', l, u)]$$
$$\Longleftrightarrow$$
$$\forall i \geq 0.\forall t \in win(t_s, p, j, i).\exists k \geq 0.\forall t' \in win(t'_s, p', j', k).[l \leq t' - t \leq u]$$

Proof. $\Leftarrow$: assume that the statement to the right of the equivalence holds. Consider any events e, e' such that $Per(e, t_s, p, j)$, and $Per(e', t'_s, p', j')$. Then it holds for any $i, k \geq 0$ that $e_i \in win(t_s, p, j, i)$, and $e'_k \in win(t'_s, p', j', k)$. Then, by instantiating $t = e_i$ and $t' = e'_k$ in the right-hand side we obtain $\forall i \geq 0.\exists k \geq 0.[l \leq e'_k - e_i \leq u]$, that is: $Delay(e, e', l, u)$.

$\Rightarrow$: assume that the statement does not hold. We show existence of e, e' where $Per(e, t_s, p, j)$, $Per(e', t'_s, p', j')$, and $\neg Delay(e, e', l, u)$. Since the statement does not hold, there exists an $i \geq 0$ and $t \in win(t_s, p, j, i)$ such that for all $k \geq 0$ there is a $t' \in win(t'_s, p', j', k)$ where $\neg(l \leq t' - t \leq u)$. Another way to express this is that there is a sequence of times $\{\, t'_k \mid k \geq 0 \,\}$ where for all elements t'_k holds that $t'_k \in win(t'_s, p', j', k)$ and $\neg(l \leq t'_k - t \leq u)$. By tightness and Lemma 1 there is an event e such that $e_i = t$ and $Per(e, t_s, p, j)$. Similarly, by tightness and Lemma 1 we can construct an event e' where $Per(e', t'_s, p', j')$, by letting $e'_k = t_k$ for each $k \geq 0$. Thus, $\exists i \geq 0.\forall k \geq 0.\neg(l \leq e'_k - e_i \leq u)$, that is: $\neg Delay(e, e', l, u)$.

Theorem 1 is illustrated in Fig 4. It shows two time windows, for the events e and e', respectively, and indicates how the distance between any t, t' drawn from the respective time window must be kept between l and u. The theorem

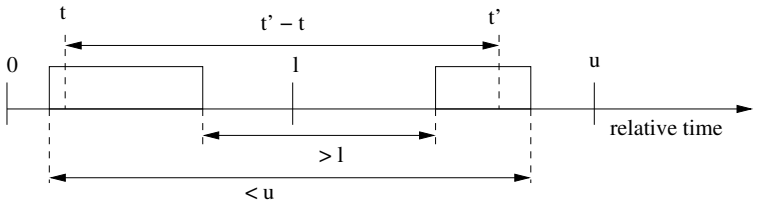

Fig. 4. An illustration of Theorem 1.

does not guarantee that the translated formula is a Presburger formula. For this to hold, the periodicities p, p' of the events e, e' must be constants.

For all the TADL2 constraints listed in Section 3, except *Sporadic* and *Repetition*, the equivalence of the translation can be proved in a similar manner as for Theorem 1. This suggests that it should be possible to prove, in a similar way, a correctness result for the translation P_R of *any* constraint in **CPExpr**. However, counterexamples can be found where the equivalence between statement and translated statement does not hold. To prove such a result we need to find a nontrivial fragment of TiCS, which is smaller than the one defined in Fig. 3, for which such a result holds. This is a topic for further research.

5 An Experiment: the Box Service Generic-External

We have tried our method by verifying parts of the timing requirements for the "Box Service Generic-External" (BSG-E) [10], which manages the fog lights in cars. It also handles the electrical protection of downstream wires, diagnostics, and the dialogue with the main car ECU over a CAN network. The total specification of the timing requirements includes ten events, five delay constraints, two periodic constraints, and one synchronisation constraint involving two events. We selected a subset of these with two events, and one delay constraint: the selection was made to provide an example with periodic events. The events are: EMA_PERM3, (filtered) voltage reading from the power supply, and CAR_CDE_BSE, the arrival of the first frame on the CAN bus from the main ECU. EMA_PERM3 is periodic, with periodicity 15 ms and zero jitter, and the nature of CAR_CDE_BSE is not specified. The delay constraint specifies a requirement that whenever a rising edge is detected on the power supply, the first frame from the CAN bus must be read within 40 ms. For this example we assume that CAR_CDE_BSE is periodic. We obtain the following timing constraints:

$$
\begin{aligned}
&AcqPerm = 5\\
&T\_init = 40\\
&Per(EMA\_PERM3, t_2, 3 \cdot AcqPerm, 0)\\
&Per(CAR\_CDE\_BSE, t_3, 3 \cdot AcqPerm, jitter)\\
&Delay(EMA\_PERM3, CAR\_CDE\_BSE, 0, T\_init)
\end{aligned}
$$

As the start time t_2 for EMA_PERM3 is not specified, we leave it open. Similarly, since CAR_CDE_BSE is not specified at all, we leave its start time t_3 and jitter open. (We cannot leave its periodicity as a parameter, as this would create a nonlinear expression in the transformed constraint.) For this example we set it to the periodicity of EMA_PERM3, but any value could be chosen.

We have verified the *Delay* constraint, transformed according to Section 4, with the `iscc`[3] calculator [22]. `iscc` can simplify sets defined by Presburger formulae using Fourier-Motzkin variable elimination. Sets defined by formulae without free variables reduce either to the empty or the universal set (that is, false or true): if the defining formula has free variables then the result will be a set defined by a simplified, quantifier-free formula in the same variables. Thus, `iscc` can be used as a Presburger solver with the ability to return parametric results.

The experiment was carried out on a Dell Optiplex 7010, with a dual-core 64 bit Intel i5-3570 processor running at 3.40 GHz, 8 GB memory, three levels of cache (256 kB, 1 MB, 6 MB) running Xubuntu linux v. 14.04.5.

```
AcqPerm = 5 and T_init = 40 and
t2 >= 0 and t3 >= 0 and jitter >= 0 and
 (not exists i : not (not (i >= 0) ||
  (not exists t : not (not (t2 + i*3*5 <= t <= t2 + i*3*5 + 0) ||
   (exists k : k >= 0 and
    (not exists t' : not (not (t3 + k*3*5 <= t' <= t3 + k*3*5 + jitter) ||
     0 <= t' - t and t' - t <= T_init)))))))
```

Fig. 5. `iscc` encoding of the Presburger formula for the *Delay* constraint.

The encoding of the Presburger formula for the *Delay* constraint, in the textual language of `iscc`, is shown in Fig. 5. The formula includes constraints on the auxiliary variables as well as non-negativity constraints on the starting times and the jitter. When run with this input, `iscc` will instantaneously return the following, simplified set expression:

```
{ [AcqPerm = 5, T_init = 40, t2, t3, jitter] :
   t2 >= 0 and t3 >= 0 and 0 <= jitter <= 40 + t2 - t3 and
   15*floor((14t2 + t3)/15) >= -40 + 14t2 + t3 + jitter }
```

We thus obtain constraints on the unknown parameters of the periodic events. We can also make runs with different periodicities of CAR_CDE_BSE to explore how this parameter affects the ability to find a solution that satisfies the constraint. In all, the ability to produce symbolic solutions facilitates a design space exploration where the system is designed as to meet its timing constraints.

[3] Our version was built using version 0.40 of the `barvinok` library.

6 Related Work

Logics for expressing and reasoning about real-time properties are mostly expressed as modal logics [1, 3, 11]. Decidable such logics allow for verification of real-time properties of systems by model-checking. In particular Timed Automata [2] have become much used for this purpose. UPPAAL [16] is a well-known tool for modeling and verification using timed automata. Jagadish [21] used UPPAAL to verify some TADL timing constraints, transformed into timed automata, for periodic events. Examples include the delay constraint. UPPAAL has also been used for verifying timing constraints expressed in the AUTOSAR Timing Extensions [7, 9], which build on the TADL timing constraints.

Our approach, using translations into Presburger formulae, is quite different from model checking. It allows the use of decision procedures that eliminate quantifiers through projections, which allows parametric solutions in the form of quantifier-free formulas. For timing verification this can be valuable in early design phases where system parameters are not yet fixed. Possibly parametric model checking of timed automata can be used for the same purpose [14], but this remains to be investigated. Furthermore our approach can be directly extended to continuous time. For this kind of time the use of projection-based decision procedures will yield a lower complexity, whereas the opposite holds for model checking.

We know few other attempts to verify timing properties by deciding Presburger formulae. Amon et al. [4] capture the logic of timing diagrams in a form that resembles our constraint language but without event variables. This sub-language corresponds to Presburger formulas.

CCSL [5] is a timing constraint language in the UML profile MARTE [20] for modeling and analysis of real-time systems. CCSL can specify *clocks*, and relations between them. Yin et. al. [23] describe how to translate a specification in a subset of CCSL into a Promela model for verification in the model checker SPIN. Ning and Pantel [13] proposed a framework for verifying timing properties of MARTE models through model checking over Timed Petri Nets.

7 Conclusions and Further Research

We have shown how to verify a number of TADL2 timing constraints under the condition that the constrained events are periodic with jitter. The TADL2 constraints then express timing requirements, whereas the periodicity and jitter of the events may stem from an implementation where tasks are triggered with some fixed periodicity. Such implementations are common in safety-critical systems, in domains like automotive and avionics. TADL2 has been developed within the automotive domain, with its typical timing requirements mind, and it forms the basis for the AUTOSAR Timing Extensions. It can therefore be of great practical interest to have methods to verify whether or not such periodically scheduled systems will meet their timing requirements expressed as TADL2 constraints. Our work provides a step towards this goal.

The translations to Presburger formulas, and their proofs of correctness, follow a common pattern. An obvious topic for future research is to find a larger fragment of TiCS for which the translation is correct.

We believe that the ability to obtain symbolic results, constraining design parameters, can be very useful. This is convenient in early design phases where all parts of the system are not yet fixed, and it facilitates a design space exploration where the system is optimised under the given timing constraints.

A possible concern is the extremely high worst-case complexity for solving Presburger formulas. However, our experience is that timing constraints in domains like automotive often are gathered into a conjunction of rather simple constraints, like delay or synchronisation constraints. The translated Presburger formula will then be a conjunction of simple constraints, each involving only a few variables. The solver can then solve these individually and intersect the results, which is likely to be much faster than the worst case. A topic for future research is to make larger case studies to see whether this is indeed the case.

8 Acknowledgments

This work was partially supported by VINNOVA through the ITEA2 TIMMO-2-USE and ITEA3 ASSUME projects. We would also like to thank Johan Nordlander for interesting discussions.

References

1. Abadi, M., Lamport, L.: An old-fashioned recipe for real time. ACM Transactions on Programming Languages and Systems 16(5), 1543–1571 (Sep 1994)
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. AUTOSAR: Homepage of the AUTOSAR project (2009), `www.autosar.org/`
7. AUTOSAR: Specification of timing extensions (2011), `www.autosar.org/`
8. Baruah, S., Buttazzo, G., Gorinsky, S., Lipari, G.: Scheduling periodic task systems to minimize output jitter. In: Proc. Sixth International Conference on Real-Time Computing Systems and Applications (RTCSA '99). pp. 62–69 (1999)
9. Beringer, S., Wehrheim, H.: Verification of AUTOSAR software architectures with timed automata. In: ter Beek, M.H., Gnesi, S., Knapp, A. (eds.) Proc. Joint 21st International Workshop on Formal Methods for Industrial Critical Systems and 16th International Workshop on Automated Verification of Critical Systems, FMICS-AVoCS 2016. Lecture Notes in Comput. Sci., vol. 9933, pp. 189–204. Springer International Publishing, Pisa, Italy (Sep 2016)

10. Blom, H., Feng, L., Lönn, H., Nordlander, J., Kuntz, S., Lisper, B., Quinton, S., Hanke, M., Peraldi-Frati, M.A., Goknil, A., Deantoni, J., Defo, G.B., Klobedanz, K., Özhan, M., Honcharova, O.: D11 language syntax, semantics, metamodel v2. Technical report (Aug 2012), https://itea3.org/project/timmo-2-use.html
11. Chaochen, Z., Hoare, C.A.R., Ravn, A.P.: A calculus of durations. Inf. Process. Lett. 40(5), 269–276 (1991)
12. 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)
13. Ge, N., Pantel, M.: Time properties verification framework for UML-MARTE safety critical real-time systems. In: Vallecillo, A., Tolvanen, J.P., Kindler, E., Störrle, H., Kolovos, D. (eds.) Modelling Foundations and Applications, Lecture Notes in Comput. Sci., vol. 7349, pp. 352–367. Springer Berlin Heidelberg (2012), http://dx.doi.org/10.1007/978-3-642-31491-9_27
14. Hune, T., Romijn, J., Stoelinga, M., Vaandrager, F.W.: Linear parametric model checking of timed automata. J. Log. Algebr. Program. 52-53, 183–220 (2002), http://dx.doi.org/10.1016/S1567-8326(02)00037-1
15. 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)
16. Larsen, K.G., Pettersson, P., Yi, W.: UPPAAL in a nutshell. Int. Journal on Software Tools for Technology Transfer 1, 134–152 (1997)
17. Lisper, B., Nordlander, J.: A simple and flexible timing constraint logic. In: Margaria, T., Steffen, B. (eds.) Proc. 5^{th} International Symposium on Leveraging Applications of Formal Methods (ISOLA'12). Lecture Notes in Computer Science, vol. 7610, pp. 80–95. Springer Berlin Heidelberg, Heraclion, Crete (Oct 2012), http://dx.doi.org/10.1007/978-3-642-34032-1_12
18. Liu, C., Layland, J.: Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. J. ACM 20(1), 46–61 (1973)
19. Mok, A.K.: Fundamental design problems of distributed systems for the hard-real-time environment. Ph.D. thesis, Massachusetts Institute of Technology (May 1983), http://www.lcs.mit.edu/publications/pubs/pdf/MIT-LCS-TR-297.pdf
20. UML profile for MARTE: Modeling and analysis of real-time embedded systems. Tech. rep., OMG (Nov 2009), www.omg.org/spec/MARTE/1.0
21. Suryadevara, J.: Validating EAST-ADL timing constraints using UPPAAL. In: Proc. 39th Euromicro Conference on Software Engineering and Advanced Applications (SEAA) (Sep 2013), http://www.es.mdh.se/publications/2988-
22. Verdoolaege, S.: barvinok: User guide. Technical report (Jan 2016), barvinok.gforge.inria.fr/barvinok.pdf
23. Yin, L., Mallet, F., Liu, J.: Verification of MARTE/CCSL time requirements in Promela/SPIN. In: Proc. 16th IEEE Int. Conf. on Engineering of Complex Computer Systems (ICECCS 2011). pp. 65–74 (Apr 2011)