Science of Computer Programming 80 (2014) 264–287

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Specifying safety-critical systems with a decidable duration logic

Savas Konur

Department of Computer Science, University of Liverpool, Liverpool, L69 3BX, UK

ARTICLE INFO

Article history:
Received 18 April 2012
Received in revised form 15 May 2013
Accepted 9 July 2013
Available online 13 August 2013

Keywords:
Safety-critical systems
Punctuality timing constraints
Temporal logics
Decidability
Tableau systems

ABSTRACT

Punctual timing constraints are important in formal modelling of safety-critical real-time systems. But they are very expensive to express in dense time. In most cases, punctuality and dense-time lead to undecidability. Efforts have been successful to obtain decidability; but the results are either non-primitive recursive or nonelementary. In this paper we propose a duration logic which can express quantitative temporal constraints and *punctuality timing constraints* over *continuous* intervals and has a reasonable complexity. Our logic allows most specifications that are interesting in practice, and retains punctuality. It can capture the semantics of both *events* and *states*, and incorporates the notions *duration* and *accumulation*. We call this logic *ESDL* (the acronym stands for *Event- and State-based Duration Logic*). We show that the satisfiability problem is *decidable*, and the complexity of the satisfiability problem is *NEXPTIME*. ESDL is one of a few decidable interval temporal logics with metric operators. Through some case studies, we also show that ESDL can specify many safety-critical real-time system properties which were previously specified by undecidable interval logics or their decidable reductions based on some abstractions.

1. Introduction

Formal modelling of real-time requirements is crucial in analysing safety-critical systems. Among these, *punctuality*[1] properties constitute an important part of the real-time requirements. There have been various modelling paradigms to make this analysis mathematically possible. Two main approaches for modelling time are *dense/continuous* modelling of time and *discrete* time models [1]. Discrete time points provide an advantage in terms of obtaining decidability and reasonable complexity; but they are not natural ways of modelling real-time systems. On the other hand, dense/continuous time domains mainly lead to undecidability for punctuality properties [2]. Indeed, until recently researchers thought that any linear-time temporal logic which can express punctuality properties is undecidable [3]. There have been recent attempts to obtain decidability; but the results are either non-primitive recursive or nonelementary [4,5,3]. In a recent study an EXPSPACE complexity is found [3] for a fragment of ML; but the resulting logic has some restrictions.

These reported results are important, but they are not sufficient because complexities of these logics are still very high to use them in the formal analysis of real-time systems. An alternative approach is to relax density/continuity, and use feasible time models (e.g. discrete); but in this approach we cannot capture exact system behaviour. This is an important limitation in real-time system study. In particular, safety-critical systems should be designed and analysed according to exact specifications; but we cannot achieve this using simple time models. Therefore, a realistic (i.e. dense or continuous) modelling of time should be used.

E-mail address: konur@liverpool.ac.uk.

[1] A *punctuality property* states, for example, that an event/state B follows an event/state A in exactly 3 seconds.

0167-6423/$ – see front matter
http://dx.doi.org/10.1016/j.scico.2013.07.012

One aim of this paper is to tackle these issues. We propose a logic, called *ESDL* (the acronym stands for *Event- and State-based Duration Logic*), which can express punctuality timing constraints over continuous-time intervals. The proposed logic is decidable, and has a reasonable complexity.

Another design choice is to choose between *points* (instants) or *intervals* for modelling time. The interval-based scheme provides a richer representation formalism than the point-based scheme to specify continuous processes and temporal statements; for example, intervals can support uncertain temporal relations between real world events, which the point-base scheme cannot support [6].

Quite to the contrary, interval temporal logics, such as, e.g. ITL [7], RTIL [8], TILCO [9], DC [10], NL [11], PNL [12], are more likely to be undecidable. In the literature, various methods have been proposed to achieve decidability for interval logics. However, most of these methods, such as translating interval logics into discrete or sampled time, cause some syntactic and semantic restrictions. For this reason, we use intervals as primitive objects of time models. Unlike many other interval logics, we do not apply any abstraction to time models, and we therefore try to minimise semantic restrictions. Yet, we can still acquire decidability and a reasonable complexity.

One important design paradigm is to choose between *events* and *states*.[2] A formal method should cover both event-based and state-based views in order to model the behaviour of real-time systems more accurately. In early phases of system development, requirements are usually provided less formally; it is therefore more convenient to specify them with event-based formalisms; on the other hand, in later and implementation stages, state-based methods are more convenient, because programming languages are generally state-based [13]. Most temporal logics employed for the specification of real-time systems, however, do not support both methods. They are either event-based or state-based. For this reason, incorporating both views in a temporal logic in a way that it can be used in specifying real-time system properties is an important research subject in this area. In this paper, we consider both views, and therefore introduce both event and state types in the syntax of ESDL to have more accurate representation of real-time systems.

Another important feature of ESDL is that it incorporates the notion of *duration*, denoting the length of a state (or an event), and *accumulation*, denoting the total duration of a state. These notions are useful for reasoning about time durations of dynamic systems.

We show that the satisfiability problem of ESDL is decidable. We provide a complexity bound for satisfiability, showing that this problem can be solved in NEXPTIME. We also propose a tableau based decision procedure.

Due to its unique syntax ESDL is different from known logics in the literature, which makes this logic interesting to explore (see Section 3.3 for a more detailed discussion). In addition to its computational feasibility, ESDL can properly specify important real-time system properties, including punctuality. Actually, we apply ESDL to well-known mine pump [14] and gas burner systems [10], and introduce two verification techniques to check the correctness of ESDL properties. Both systems were studied with some decidable variants of DC [14–16]; but these variants are based on some abstractions. Since ESDL is defined over genuine continuous intervals, it can capture the system behaviour more accurately.

The plan of this paper is as follows. In Section 2 we will discuss intervals and their relations. In Section 3 the syntax and semantics of ESDL will be presented. In Section 4 we will present the tableau method devised to analyse the complexity of ESDL. In Section 5 we apply the logic ESDL to some safety-critical systems, and analyse the verification problem. In Section 6 we will give some concluding remarks, and discuss some future work.

2. Intervals and their relations

In the rest of the paper we represent intervals as pairs of interval bounds which take real numbers. More formally we say that an *interval* is a pair $[t_1, t_2]$ such that $t_1, t_2 \in \mathbb{R}$ and $t_1 \leqslant t_2$. We denote the set of all intervals $\{[t_1, t_2]\colon\ t_1 \leqslant t_2 \wedge t_1, t_2 \in \mathbb{R}\}$ by $\mathcal{I}$, and we use letters $I, J, \ldots,$ as intervals. It can be simply observed that intervals may be punctual. Intervals can be considered as a closed, bounded, convex and non-empty subset of the real line.

Here, we also define the functions ***beg*** and ***end*** return the beginning and end points of an interval, respectively. For example, $beg([t_1, t_2])$ returns t_1 and $end(J)$ returns t_2.

In [6] Allen introduced thirteen well-known different binary relations between intervals on a linear ordering, which are *before*, *meets*, *overlaps*, *starts*, *during*, *finishes*, *equals*, *finished by*, *includes*, *started by*, *overlapped by*, *met by* and *after*. Given two time intervals J and J', their relative positions can be described by exactly one of the elements of the set of thirteen basic interval relations. Namely, J before J' is denoted by $R^b(J, J')$; J after J' is denoted by $R^{\bar{b}}(J, J')$; J meets J' is denoted by $R^m(J, J')$; J met-by J' is denoted by $R^{\bar{m}}(J, J')$; J overlaps J' is denoted by $R^o(J, J')$; J overlapped-by J' is denoted by $R^{\bar{o}}(J, J')$; J during J' is denoted by $R^d(J, J')$; J includes J' is denoted by $R^{\bar{d}}(J, J')$; J starts J' is denoted by $R^s(J, J')$; J started-by J' is denoted by $R^{\bar{s}}(J, J')$; J finishes J' is denoted by $R^f(J, J')$; J finished-by J' is denoted by $R^{\bar{f}}(J, J')$; and J equals J' is denoted by $R^e(J, J')$.

[2] Consider the sentences "John wrote the letter", and "John worked on the letter". The former reports an event, while the latter reports an ongoing process or state-types. The sentences which describe state-types have the property that whenever they are true over an interval I, they are true over every subinterval of I. Other sentences describing completed events have the property that whenever they are true over an interval I, they are false over some subinterval of I.