Science of Computer Programming 85 (2014) 137–165

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Deriving real-time action systems with multiple time bands using algebraic reasoning

Brijesh Dongol[a,∗,1], Ian J. Hayes[b,2], John Derrick[a,1]

[a] Department of Computer Science, The University of Sheffield, S1 4DP, UK
[b] School of Information Technology and Electrical Engineering, The University of Queensland, Australia

HIGHLIGHTS

- An interval-based logic for deriving action systems via algebraic reasoning is developed.
- Temporal operators are given an algebraic semantics, avoiding the need for a separate temporal logic semantic layer.
- True concurrency between an action system and its environment is addressed.
- Compositional methods for reasoning over multiple time bands and sampling is developed.
- A controller for a two-pump, two-tank system, taking into account various real-world timing constraints is derived.

ARTICLE INFO

Article history:
Received 7 January 2013
Received in revised form 5 August 2013
Accepted 13 August 2013
Available online 26 August 2013

Keywords:
Sampling logic
Action systems
Time bands
Algebraic reasoning
Program refinement

ABSTRACT

The verify-while-develop paradigm allows one to incrementally develop programs from their specifications using a series of calculations against the remaining proof obligations. This paper presents a derivation method for real-time systems with realistic constraints on their behaviour. We develop a high-level interval-based logic that provides flexibility in an implementation, yet allows algebraic reasoning over multiple granularities and sampling multiple sensors with delay. The semantics of an action system is given in terms of interval predicates and algebraic operators to unify the logics for an action system and its properties, which in turn simplifies the calculations and derivations.

1. Introduction

Modern cyber-physical systems are implemented using a digital controller that executes by sampling the various system sensors, performing some computation, then signalling the components being controlled to change their behaviour in accordance with the system requirements. This paper presents methods to formally derive controllers from the system specifications, where the controllers periodically sample the environment and signal various components when necessary. We use a logic for reasoning about complex systems with events in different time granularities, e.g., sampling events for different components may occur at different rates, and these rates can also depend on the properties of the system being measured. The components being controlled can also operate at different time granularities, e.g., the effect of a motor reaching operating speed may occur in a different time band than the effect of a switch that powers on a motor.

∗ Corresponding author.
E-mail addresses: B.Dongol@sheffield.ac.uk (B. Dongol), Ian.Hayes@itee.uq.edu.au (I.J. Hayes), J.Derrick@dcs.shef.ac.uk (J. Derrick).

1 Brijesh Dongol and John Derrick are supported by EPSRC Grant EP/J003727/1.
2 Ian Hayes is supported by ARC Discovery Grant DP130102901.

0167-6423/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.scico.2013.08.009

The derivation method builds on our method of enforced properties [15–17,19], which uses the verify-while-develop paradigm to incrementally obtain program code from the underlying specifications. Our framework incorporates a logic of time bands [9,10,49], which allows one to formalise properties at different time granularities and define relationships between these properties. Behaviours of components at fine levels of granularity often involve interactions that may not necessarily be observed when assuming a coarse level of atomicity. Development of a system assuming coarse-grained atomicity can be problematic if the atomicity assumptions cannot be realised by the system under development, causing the developed system to become invalid. On the other hand, consideration of fine-grained interactions results in an increase in the complexity of the reasoning. In this paper, we use a high-level logic that allows one to describe the observable states that may occur when sampling variables at finer time bands [10,19,21,30].

We present our methods using the action systems framework, which has been used as a basis for several theories of program refinement [3,7,4,5]. In its simplest form, an action system consists of a set of actions (i.e., guarded statements) and a loop that at each iteration non-deterministically chooses then executes an enabled action from the set of actions. Thus, periodic sampling is naturally supported by the framework.

To model the behaviour of a program in an environment one may include actions corresponding to the program and its environment within a single action system [7,17] so that the actions corresponding to the controller and its environment are interleaved with each other. However, in the context of real-time reactive systems, this model turns out to be problematic because for example it is unable to properly address *transient properties* [19,20]. Such properties only hold for a brief amount of time, say an attosecond, and hence, a real-world implementation would never be able to reliably detect the property. A theoretical model that considers interleaving of controller and environment actions would require that the implementation does detect a transient property, which is unrealistic. Instead, an implementation should be allowed to ignore transient properties because they cannot be reliably detected. In this paper, like [19,20], we modify the semantics so that an action system executes with its environment in a truly concurrent manner. This allows one to develop a theoretical model that properly addresses transient properties — an implementation is only required to handle non-transient properties.

This paper adds to our series of papers on program derivation [16,17,19,20,24,25]. Of these, [16,24,25] consider concurrent program derivation and [17,19,20] consider real-time programs. Our papers [17,19,20] increasingly consider more realistic assumptions in concurrent real-time systems and the most advanced of these [19] allows one to consider sampling issues and components that operate over multiple time granularities. However, the framework itself has become increasingly complex and becomes a bottleneck to achieving scalable derivations because program properties are expressed in an interval-based LTL-style [40] logic, whereas the requirements are expressed using interval predicates. As a result, a derivation step is required to transform the interval predicate requirements to the level of the program.

In this paper, we remove this bottleneck by defining a semantics for action systems using algebraic operators in the style of Back and von Wright [6]. However, unlike Back and von Wright, we address real-time issues by basing our semantics on an algebra for interval predicates [22,35,36]. This allows one to improve uniformity across the model by enabling one to use interval predicates to express system requirements as well as program behaviour. Thus, the additional interval-based LTL logic from [19,20] is completely avoided, and all proofs are carried out at the level of interval predicates.

To enable compositionality, we use rely/guarantee-style reasoning [12,31,38,39], where the *rely* condition is an interval predicate that specifies the behaviour of the environment. Unlike Jones, [12,38] who defines rely-guarantee reasoning in a relational setting, we assume that rely conditions are interval predicates that may specify real-time behaviour [21]. The underlying theory also includes methods for reasoning about delays and feedback.

This paper is structured as follows. In Section 2, we present a motivating example consisting of two pumps and two water tanks. In Section 3, we present our background theory and in Section 4, we present methods for reasoning about multiple time bands and sampling of multiple sensors. In Section 5 we present a novel algebraic semantics for action systems, which includes constructs for reasoning about enforced properties and rely conditions. We also discuss action system refinement. We use the theory from the earlier sections to derive a controller for our motivating example in Section 6. We consider some related work in Section 7 and present some concluding thoughts in Section 8.

2. Example: A two-pump system

Throughout this paper, we consider a system consisting of two water tanks $Tank_1$, $Tank_2$ and two pumps $Pump_1$, $Pump_2$ depicted in Fig. 1 (also see [1]). The environment (of the system) adds water to $Tank_1$ and does not affect $Tank_2$. We assume that $Tank_1$ is allowed to overflow, but $Tank_2$ is not. $Pump_1$ removes water from $Tank_1$ and fills $Tank_2$. $Pump_2$ only operates if *Button* is pressed and removes water from $Tank_2$. Aichernig et al. [1] describe the following requirements. We have adapted their informal specification to clarify the input/output behaviours of the pumps and to better distinguish safety (**S1**, **S2** and **S3**) and progress (**P1**, **P2** and **P3**) properties. Note that a progress property to turn $Pump_1$ off is not needed because it is implied by safety properties **S1** and **S2**.

S1. Whenever $water_1$ (the water level in $Tank_1$) is $empty_1$ or below, $Pump_1$ must be stopped.
S2. Whenever $water_2$ (the water level in $Tank_2$) is $full_2$ or above, $Pump_1$ must be stopped.
S3. Whenever $water_2$ is $empty_2$ or below, $Pump_2$ must be stopped.