Science of Computer Programming 77 (2012) 779–798

Contents lists available at SciVerse ScienceDirect

Science of Computer Programming

journal homepage: www.elsevier.com/locate/scico

SAT-based verification for timed component connectors

S. Kemper

Centrum Wiskunde en Informatica, Amsterdam, The Netherlands

ARTICLE INFO

Article history:
Received 30 November 2009
Received in revised form 3 February 2011
Accepted 8 February 2011
Available online 24 February 2011

Keywords:
Timed constraint automata
Abstraction refinement
Model checking
SAT
Component-based software engineering

ABSTRACT

Component-based software construction relies on suitable models underlying components, and in particular the coordinators which orchestrate component behaviour. Verifying correctness and safety of such systems amounts to model checking the underlying system model. The model checking techniques not only need to be correct (since system sizes increase), but also scalable and efficient.

In this paper, we present a SAT-based approach for bounded model checking of Timed Constraint Automata, which permits true concurrency in the timed orchestration of components. We present an embedding of bounded model checking into propositional logic with linear arithmetic. We define a product that is linear in the size of the system, and in this way overcome the state explosion problem to deal with larger systems. To further improve model checking performance, we show how to embed our approach into an extension of counterexample guided abstraction refinement with Craig interpolants.

1. Introduction

Component-based software engineering supports constructing large systems by composing individual components. The correctness and safety of these concurrent systems depend on *inter-component* communication actions which happen at the *right time*. Components are often only available as black boxes; therefore, there is a need for component connectors that provide exogenous coordination, i.e., coordination from *without* [2]. As such, these component connectors require true concurrency in time, which combines synchrony and asynchrony, to express complex coordination patters.

The computational complexity introduced by the infinite state space of these *real-time systems* leads to severe limitations in scalability even within very well-established model checkers like UPPAAL (http://www.uppaal.com). Aside from the omnipresent state explosion problem [14] already present in finite state model checking, current model checking techniques for real-time systems are still limited in the number of concurrent quantitative temporal observations (measured by clocks). A particularly dramatic cause of the state explosion problem is the exponential blow-up obtained by forming the cross product for parallel composition. To avoid this, we define a linear-size parallel composition for the logical representation of TCA. By providing an initial valuation for step 0, typically only a reduced part of the full parallel composition has to be expanded from our representation during satisfiability checking (SAT solving).

Very sophisticated and well-optimised techniques (e.g., [28]) guide high-end SAT solvers to explore only a comparably narrow fragment around the part of the state space relevant for the particular safety property. We build upon this development by choosing a linear arithmetic/propositional encoding, a philosophy that has successfully proven its great potential in finite state systems [13]. With this basis, we exploit the particularities of transition systems induced by TCA using abstraction refinement to deal with the challenges of infinite states.

E-mail address: s.kemper@cwi.nl.

0167-6423/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.scico.2011.02.003

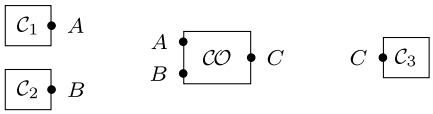

Fig. 1. TCA, conceptual view.

1.1. Timed constraint automata

Timed Constraint Automata [4] (TCA) are a combination of constraint automata [5] (CA) and timed automata (TA) with location invariants [1]. Originally defined as a semantical model for the channel based coordination language $\mathcal{R}$eo [3] (and so far having been used for this single purpose only), they offer a powerful coordination mechanism for channel based coordination languages in general. In this work, we exploit the full modelling spectrum of the formalism, by *directly* using TCA to implement coordinating connectors in networks where timed components communicate by exchanging data through multiple channels. The behaviour of the network is given by synchronisation between channel ends (ports).

While the functionality of channels is often limited to synchrony and (FIFO) buffering, TCA allow connectors with arbitrary behaviour. These connectors provide exogenous coordination, by imposing a certain communication pattern – for example reordering or delays – on associated components. TCA are compositional, which allows to easily build complex connectors out of simpler ones.

Most action-based (coordination) models, like e.g. finite state machines, I/O automata or TA, permit only a single action per transition. As a consequence, synchrony, and concurrent execution of actions in the parallel composition, is reduced to arbitrary interleavings plus nondeterminism. Especially for timed systems (like TA) – aside from being unintuitive – this does not correctly capture the nature of distributed systems, since it imposes a sequential order on actions which conceptually happen at the same time. Moreover, from a technical point of view, the presence of all possible interleavings amplifies the state explosion problem. In contrast, TCA allow sets of actions on each transition, which permits *true concurrency*, as this directly models (truly atomic) synchronous communication through different ports.

In this way, TCA provide a coordination model to implement component connectors, which combines the notions of real-time and true concurrency, and allows for complex coordination patterns including both synchrony and asynchrony.

Example 1.1 (*Introduction*)**.** Fig. 1 shows our conceptual notion of networks of TCA. The network presented here consists of three components $\mathcal{C}_1$, $\mathcal{C}_2$ and $\mathcal{C}_3$, and a central connector $\mathcal{CO}$, which is connected to the components through ports A, B and C, respectively. Throughout the paper, we will use the connector $\mathcal{CO}$ as a running (toy) example. The basic idea of $\mathcal{CO}$ is to repeatedly receive data from components $\mathcal{C}_1$ (through port A) and $\mathcal{C}_2$ (through B), and to send this data to $\mathcal{C}_3$ (through C). Depending on constraints on the received data, either component $\mathcal{C}_1$ or $\mathcal{C}_2$ is connected to – i.e., data is transmitted between – component $\mathcal{C}_3$ (dynamic reconfiguration). Note that this is just a toy example, used to illustrate the concepts introduced in this paper. For a more meaningful example, consider Section 6.

1.2. Abstraction refinement

Abstraction refinement [14,19] is a promising direction of research to overcome the challenges of the state explosion problem and infinite state model checking, while preserving correctness of verification results. Abstraction techniques based on over-approximation (also called *conservative* approximation) enrich the system behaviour, by removing constraints that are considered irrelevant for verifying a particular property. These techniques may yield false negatives: a safety property is rejected in the abstract system, though it holds in the original system. If, however, the abstract system is safe (no error state is reachable) then, by over-approximation, so is the original.

Based on the representation of TCA in propositional logic with linear arithmetic, iterative abstraction refinement consists of the following steps: applying the abstraction function to the representation, we automatically produce a simpler *abstract* version of it. After *unfolding* the resulting transition formula k times, a *satisfiability check* solves the bounded reachability question in the abstract system. Depending on the outcome, the system has either been proven safe (error state is unreachable) within bound k, or needs to be analysed with respect to an abstract counterexample (*concretised*), again using SAT solving. If the abstract counterexample has a counterpart in the non-abstracted system, then the system is unsafe. Otherwise, the counterexample is *spurious* and results from an inappropriate choice of abstraction. Analysing the counterexample (with Craig interpolants derived by the SAT solver, e.g. FOCI (based on [25]) or MathSAT [24]) then helps to *refine* the abstraction and start over until the system is proven safe (within bound k) or unsafe.

1.3. Contributions

The main contributions of this paper can be summarised as follows: we extend the (expressiveness of the) basic definition of TCA [4], by generalising from finite data domains to countably infinite data domains.

We then develop a formal framework for direct investigation and verification of TCA: we define a constraint-based representation of TCA in propositional logic with linear arithmetic. These constraints capture the current state of connectors in the network, and possible synchronisations of connectors with each other and the environment. In this way, we can