Information Processing Letters 122 (2017) 8–16

Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

On minimising the maximum expected verification time

Toni Mancini, Federico Mari, Annalisa Massini *, Igor Melatti, Ivano Salvo, Enrico Tronci *

Computer Science Department, Sapienza University of Rome, Italy

ARTICLE INFO

Article history:
Received 18 February 2016
Received in revised form 2 November 2016
Accepted 1 February 2017
Available online 4 February 2017
Communicated by Krishnendu Chatterjee

Keywords:
Formal verification
Explicit model checking
System-level formal verification
Formal methods
Software engineering

ABSTRACT

Cyber Physical Systems (CPSs) consist of hardware and software components. To verify that the *whole* (i.e., software + hardware) system meets the given specifications, *exhaustive* simulation-based approaches (Hardware In the Loop Simulation, HILS) can be effectively used by first generating *all* relevant simulation scenarios (i.e., sequences of *disturbances*) and then actually simulating all of them (*verification phase*). When considering the whole verification activity, we see that the above mentioned verification phase is repeated until no error is found. Accordingly, in order to minimise the time taken by the whole verification activity, in each verification phase we should, ideally, start by simulating scenarios witnessing errors (*counterexamples*). Of course, to know beforehand the set of such scenarios is not feasible. In this paper we show how to select scenarios so as to minimise the Worst Case Expected Verification Time.

1. Introduction

A CPS consists of hardware (e.g., engines, electrical circuits, etc.) and software components. Thus, in order to verify a CPS design, we need methods and tools that can model and effectively support analysis of hardware as well as software components.

From a formal point of view, CPS can be modelled as hybrid systems (see, e.g., [8,32,31] and citations thereof). Many *Model-Based Design* software tools offer support for modelling and simulation of CPSs. Well known examples are Simulink, VisSim, Open Modelica, JModelica, Dymola. All such tools take as input a (mathematical) model of the behaviour of the CPS along with a simulation scenario and provide as output the time evolution (*trace* or *simulation run*) of the system.

System Level Verification of CPSs aims at verifying that the *whole* (i.e., software + hardware) system meets the given specifications. *System Level Formal Verification (SLFV)* has the goal of *exhaustively* verifying that the above holds for *all* possible operational scenarios.

For digital circuits, formal verification is usually carried out using symbolic model checking techniques (see, e.g., [13,12]). Unfortunately, model checkers for hybrid systems cannot handle SLFV of real world CPSs because of state explosion. Thus, HILS is currently the main workhorse for system-level verification of CPSs, and is supported by model-based design tools.

In HILS, the *control software* (see, e.g., [30,4,5]) reads/sends values from/to mathematical models (*simulation*) of the physical systems (e.g., mechanical or electrical systems) it will be interacting with. Simulation can be very time consuming. Accordingly, in order to reduce system design time, there are tools providing modelling and simulation software along with FPGA-based hardware to support real-time simulation. Examples are Opal-RT and dSpace.

Finally, model-based design of CPSs often refers to the activity of synthesising control software from system re-

* Corresponding authors.
E-mail address: massini@di.uniroma1.it (A. Massini).

http://dx.doi.org/10.1016/j.ipl.2017.02.001
0020-0190/

quirements (see, e.g., [7,6] and citations thereof). Here, instead, we assume that a model for the whole system (software + hardware) is given, and we are only interested in CPS SLFV.

1.1. Motivations

Simulation-based approaches to the analysis of hybrid systems have been proven very effective in application domains as diverse as CPSs (see, e.g., [24,28,17,11,41,1,42]), smart grids (see, e.g., [40,29,20]) and biological systems (see, e.g., [22,38]). The goal of all such approaches is to show that, notwithstanding the possible presence of *disturbances* (i.e., uncontrollable events such as faults, variations in system parameters, etc.) from the environment, the system meets its requirements. This is done by using HILS to show that for all *simulation scenarios* (i.e., time sequences of disturbances) in a given set, the system meets its requirements. HILS, in turn, is carried out using a simulator (e.g., Simulink, Open Modelica, JModelica, Dymola) able to model and simulate both hardware (e.g., mechanical or electrical systems) as well as software components. Simulation-based verification can be carried out using two approaches: *online* and *offline*. The *online* approach typically selects the next disturbance to be simulated using a *Monte Carlo* strategy. The verification activity then consists of a sequence of disturbance generation and simulation steps. The *offline* approach *first* generates the whole (ordered) set of scenarios to be simulated (*scenario generation phase*) and *then* simulates all of them (*verification phase*).

The verification activity simulates the SUV until either a scenario (*counterexample*) whose simulation returns *FAIL* is found, or all scenarios have been simulated and simulation returns *PASS*. If the verification activity returns *FAIL*, then the SUV design is revised, by exploiting the counterexample, and a new verification activity is performed. We note the following points.

First, with an *offline* approach to CPS verification, more than 99% of the overall verification time is spent in the verification phase (see, e.g., [24]). Namely, for CPSs, simulating a single scenario may take from several seconds to several minutes (see, e.g., [24,26,28]) depending on the complexity of the system model (since typically a system of ordinary differential equations has to be solved in order to simulate the SUV dynamics). For example, for the SUV considered in [24], we see that generating a simulation scenario takes on average 0.45 ms (thus generating 4 million simulation scenarios takes about 30 minutes), whereas the Simulink simulation of a single scenario takes on average about 16.8 seconds (and the sequential simulation of all scenarios would take more than 700 days!). This is in contrast with, e.g., digital hardware simulation, where the time needed to generate a scenario and to simulate it are comparable. Accordingly, within an *offline* framework, we can afford to increase (e.g., doubling) the time spent in the generation phase if that can decrease (even slightly) the expected time for the verification phase. Note however that the *offline* approach makes sense only for CPSs, whereas the *online* approach can always be used and is indeed the approach always used in digital hardware as well as in software verification.

Second, whenever an error is found (and the SUV revised accordingly), the verification activity needs to simulate again *all* scenarios, including those already been simulated in previous verification activities (since revising the SUV design may have introduced new errors).

Third, in the *offline* approach, the scenario generation phase is performed only once, at the beginning of the verification activity. This is possible because the scenario generation phase depends only on the environment the SUV will be interacting with, and not on the SUV model itself. Thus, revising the SUV design, after an error has been found, does not change the set of simulation scenarios to be considered in the verification phase.

From the above points, it follows that simulating scenarios preceding a counterexample is indeed a waste of time, since those scenarios will have to be simulated again anyway. In order to minimise such a waste of time, one would like to order the simulation scenarios in such a way that those witnessing errors (counterexamples) are simulated at the very beginning in each verification phase. Of course, to know beforehand the set of counterexamples is not feasible (it is indeed the purpose of the verification activity). Furthermore, while reordering of the set of scenarios to be simulated can be effectively done within an *offline* framework (and has been done indeed in [25,28]), this is not possible within an *online* framework where SUV simulation starts before the whole set of scenarios is known. Indeed, from [10] we see that no strategy to select the next disturbance in an *online* strategy can be optimal for all SUVs.

The above considerations motivate investigation on effective algorithms that can order the set of simulation scenarios so as to minimise the *worst case expected time* for the verification activity within an *offline* CPS verification approach.

Of course our techniques could be applied to simulation-based verification of any system (be it software or hardware) with a finite set of scenarios. However, from a practical point of view, it only makes sense when scenario generation takes much less than scenario simulation (see discussion above). Presently, to the best of our knowledge, this is only the case for CPSs and this is why we focus on them.

1.2. Main contributions

From the previous discussion we see that the *computation time* (defined as the number of scenarios to be simulated before hitting a *counterexample*, if any) of a verification phase (*off-line* approach) depends on the order in which scenarios are simulated and on *where* in such an order counterexamples are.

Accordingly, the generic verification phase (also simply called *verification* in the following) can be modelled as a two-player zero-sum game as follows. First, player 1 (*verifier*) chooses the (possibly probabilistic) ordering strategy in which scenarios will be simulated. Second, player 2 (*adversary*) chooses which scenarios will be counterexamples (that is, will witness an error). Finally, the verifier simulates the scenarios in the chosen order.