DOWNLOAD PDF MODELING AND VERIFICATION OF REAL-TIME SYSTEMS

Chapter 1 : Modeling and Verification of Real-time Systems - PDF Free Download

*Modeling and Verification of Real-time Systems: Formalisms and Software Tools [Nicolas Navet, Stephan Merz] on calendrierdelascience.com *FREE* shipping on qualifying offers. This title is devoted to presenting some of the most importantconcepts and techniques for describing real-time systems andanalyzing their behavior in order to enable the designer to.*

Let us consider the net represented in Figure 1. This formula is true on the SCG, but false on the state graph of the net: In the absence of silent transitions, branching properties are captured by the concept of bisimulation; any graph of classes bisimilar with the state graph omitting temporal annotations preserves its branching properties. Rather than these constructions, we will recall that of [BER 03b], which is applicable to any TPN and generally yields smaller graphs. An algorithm is as follows [PAI 87]: Following [YON 98], let us call atomic a class stable with respect to all its following classes, i. The partition technique is explained in detail in [BER 03b]. To compute such systems, we proceed as for the calculation of strong state classes by Algorithm 1. This method is summarized by Algorithm 1. The exact implementation of Algorithm 1. Several optimizations are essential for good performances over long sequences. In particular, these systems are often constituted by independent subsystems, a property that we should exploit. Delays relative dates versus dates absolute The systems built by Algorithm 1. In this case, we will introduce an additional variable, start, which represents the date of initialization of the net. Illustration As an illustration, any schedule of support t1. The various tools constituting the environment can be combined or used independently. These tools include the following: For Petri nets untimed , tina provides the traditional constructions marking graphs, covering graphs but also some abstract state space constructions based on partial order techniques such as stubborn sets and covering steps. These constructions as described in [BER 04] preserve certain classes of properties, such as the absence of deadlocks, the properties of certain logics, or test equivalence. For Time Petri nets, tina offers all of the constructions of state class graphs discussed in this chapter. These graphs can be produced in various formats: It produces on request either a characterization of all schedules over a sequence as an inequality system , or an arbitrary solution of this system, in delays or dates, put into canonical form or not. It operates on the state space abstractions produced by tina, or, through a conversion tool, on the behavior graphs produced by other tools such as the CADP tools [FER 96]. This tool will be described at length in section 1. A screen capture of a Tina session is reproduced in Figure 1. Screen capture of a Tina session 1. Next, we need a description or abstraction of the behavior of the net preserving the properties we can express in that logic, the SCG and SSCG constructions provided by tina are adequate for this purpose. Finally, we need a tool to evaluate the truth values of the formulae expressing correctness on the behavior abstraction produced, this is the task of the selt module. The pair si , ai will be called step i. Classically, the reachability relation is assumed to be total: Preservation of LT L properties by tina constructions The graphs of state classes described in sections 1. Since the static temporal intervals associated with the transitions of a Time Petri net are not necessarily bounded, it is possible that control forever remains in certain states of the Kripke structure, even though no loop on this node materializes this arbitrary latency. This information can be retrieved from the temporal information captured by the corresponding state classes, however. When building state space abstractions with tina, the user has the choice of materializing these arbitrary latencies or not. In the event of non-satisfaction of a formula, selt can provide a sequence counter-example, either in clear or under a format loadable by the stepper simulator of Tina, so that it can be replayed. Note that the counter-example sequence obtained by the above procedure is untimed. When the input model is timed, it is thus necessary to compute from it a timed counter-example; this can be done by the module plan of Tina described in section 1. In some cases, in particular for timed systems, the counter-example sequence of a formula can be very long and thus not easily exploitable by the user. For this reason, selt may optionally produce compacted counter-examples, in a symbolic form. The selt logic The Kripke transition systems referred to here are obtained from Petri nets

or Time Petri nets. An enrichment of the language of formulae corresponds to the enrichment of Kripke transition systems. To exploit the added information, the Boolean propositional calculus parametrizing the logic will be replaced by a multi-valued calculus, and the query language of selt is extended with logico-arithmetic operators.

DOWNLOAD PDF MODELING AND VERIFICATION OF REAL-TIME SYSTEMS

Chapter 2 : Scheduling analysis real-time systems - Wikipedia

Solidly rooted in the real world, including its constant need for the highest possible reliability at the fastest possible speed, these systems require a thorough understanding of a range of models and techniques. Researchers Merz and Navet and their contributors therefore offer a variety of tools.

Classifications[edit] The criteria of a real-time can be classified as hard , firm or soft. The scheduler set the algorithms for executing tasks according to a specified order. The more challenging scheduling algorithm is found in multiprocessors, it is not always feasible to implement a uniprocessor scheduling algorithm in a multiprocessor. In general terms, in the algorithm for a real-time scheduling system, each task is assigned a description, deadline and an identifier indicating priority. The selected scheduling algorithm determines how priorities are assigned to a particular task. A real-time scheduling algorithm can be classified as static or dynamic. For a static scheduler, task priorities are determined before the system runs. A dynamic scheduler determines task priorities as it runs. An output signal indicates the processing status. It is not always possible to meet the required deadline; hence further verification of the scheduling algorithm must be conducted. Two different models can be implemented using a dynamic scheduling algorithm; a task deadline can be assigned according to the task priority earliest deadline or a completion time for each task is assigned by subtracting the processing time from the deadline least laxity. Testing and verification[edit] The performance verification and execution of a real-time scheduling algorithm is performed by the analysis of the algorithm execution times. Verification for the performance of a real-time Scheduler will require testing the scheduling algorithm under different test scenarios including the worst-case execution time. These testing scenarios include worst case and unfavorable cases to assess the algorithm performance. The time calculations required for the analysis of scheduling systems requires evaluating the algorithm at the code level. One method is by testing each input condition and performing observations of the outputs. Depending on the number of inputs this approach could result in a lot of effort. Another faster and more economical method is a risk based approach where representative critical inputs are selected for testing. This method is more economical but could result in less than optimal conclusions over the validity of the system if the incorrect approach is used. Retesting requirements after changes to the scheduling System are considered in a case by case basis.