LIMITING STATE SPACE EXPLOSION OF MODEL CHECKING USING DISCRETE EVENT SIMULATION: COMBINING DEVS AND PROMELA

Abdelhak Khemiri
Maamar El Amine Hamri
Claudia Frydman

Laboratoire d'Informatique et des Systèmes
52 Avenue Escadrille Normandie Niemen
13397 Marseille Cedex 20, France
{abdelhak.khemiri, amine.hamri,
claudia.frydman}@lis-lab.fr

Jacques Pinaton

STMicroelectronics
190 avenue Celestin Coq, ZI
Rousset, 13106, France
jacques.pinaton@st.com

ABSTRACT

In this paper we propose an approach combining discrete event simulation and model checking. Indeed, methods like model checking suffer from the state space explosion when the modeled system is complex. Consequently, we propose an approach that allows the model checking procedure to focus on a subset of the total state space that is more likely to contain an erroneous state regarding a property. To do so, a simulation run that allows us to observe some qualitative metrics is performed, and stopped when a predefined *"critical state"* is reached. This state is then projected as the initial state of the non-deterministic finite automaton used by the model checking procedure. Thus, the search will only explore states reachable from this critical state, limiting state space explosion. We finally illustrate the proposed approach through a Network-On-Chip system, and compare it with the ad hoc classical model checking approach.

Keywords: Model Checking, PROMELA, SPIN, Discrete Event Simulation, DEVS.

1 INTRODUCTION

Nowadays, hardware and software systems are becoming increasingly complex, and at the same time they are more and more prevalent in our daily lives. Failure of these systems can have serious consequences and is therefore unacceptable. The need of methods that can guarantee that a system satisfies specific properties is growing. Therefore, several methods have been developed to verify a system specification. Among them, the first method is known as testing. In this approach, observations and experiments are made directly on the real system, however, for some systems, this method is extremely expensive and can also be dangerous. Rather than interacting with the real system, the field of Modeling and Simulation (M&S) proposes to make a model, which is an representation allowing properties verification of the real system. A simulation, which is then only the execution of the model through time, is performed to generate the model behavior by acting on its state variables and outputs according to inputs and parameters.

M&S provides relevant tools and techniques for system Verification and Validation (V&V). This field, besides providing a methodology that simplifies modeling, proposes a quasi-universal formalism that makes it possible to represent a vast majority of systems: the Discrete EVent System specification (DEVS). DEVS is a hierarchical and modular dynamic formalism to design, analyze, and control both continuous and discrete event systems. Due to its real-time base and modularity, DEVS has been considered as one of the most ex-

SummerSim-AToMS, 2019 July 22-24, Berlin, Germany; ©2019 Society for Modeling & Simulation International (SCS)

pressive formalisms for the modeling and simulation of Discrete Event System. The DEVS formalism and its variations have been successfully applied in many applications (Zeigler, Kim, and Buckley 1999, Wang et al. 2015). The DEVS formalism is well adapted to build models at low levels of abstraction, and therefore allows formalizing precise, low-level description or implementation of a system, permitting different quantitative measures to be performed on the model. Nonetheless, as DEVS models are deterministic, high-level specifications cannot be modeled (Dacharry and Giambiasi 2005) and the verification of the model depends greatly on the scenarios tested.

Formal methods are one of the most efficient approaches for the V&V of software and electronic systems, or even systems in a broader sense, thanks to the use of rigorous logical reasoning and mathematical proof techniques. Formal methods are a set of formal notations and tools allowing strict and rigorous description of the system under study, with formal semantics and a proof mechanism. These methods are divided into two families: theorem proving and model checking. In this paper, we will focus on the latter. Introduced by Clarke and Emerson (1981) and Queille and Sifakis (1982), model checking techniques exhaustively explore a system's state space and demonstrate its validity regarding temporal specifications, expressed by means of logical formulas such as Linear Temporal Logic (LTL) or Computation Tree Logic (CTL). The Simple Promela INterpreter (SPIN) (Holzmann 1997, Holzmann 2003) is one of the most powerful and most widely used verification tools. In SPIN, specifications are expressed through a PROtocol MEta LAnguage (PROMELA) (Holzmann 2003), which describes the behavior of the model by defining the system specification, and the properties to verify on the model by LTL formulas.

The point is that formal methods are inefficient, and even inapplicable to complex systems, such as timed or discrete event systems. Indeed, the model checking techniques face problems of complexity and combinatorial explosion due to the complexity of the system's components interactions (Clarke et al. 2012). To overcome these problems and to allow a resolution in a reasonable time, formal methods require building models at high levels of abstraction in order to ignore some elements that do not appear indispensable in the description of the system. Therefore, these methods represent the system as a non-deterministic finite automaton (NFA) to generalize the model and remove implementation-level details (Holzmann 2003, Dacharry and Giambiasi 2005).

This paper presents a new approach to combine both methods of discrete event simulation and model checking. Indeed, simulation provides quantitative and qualitative measures of the behavior of the model that cannot be obtained via a query of the NFA. Therefore, a simulation can be used to identify — by means of conditions using quantitative measures — a subset of states where a property is more likely to be unfulfilled. Thereby, a simulation can allow the model checking procedure to focus on a subset of the total state space, thus limiting the well-known state explosion problem.

The rest of paper is organized as follows: in Section 2, we establish some notations and recall some definitions pertinent to the study of DEVS simulations and model checking. In Section 3, a body of literature is reviewed and discussed. In Section 4, the approach and its underlying assumptions are fully described. The experimentation, results and a comparison between the proposed approach and classical ad hoc model checker are presented in Section 5. Finally, Section 6 concludes on the paper and highlights future works.

2 DEFINITIONS

2.1 Formal verification

In order to verify correctness properties of systems, model checking and more generally formal verification methods have to express the system and the properties as mathematical models. Usually the system model is given as a non-deterministic finite automaton.

Definition 1. *A Non-deterministic Finite Automaton (NFA) is a tuple (Q,Σ,Δ,Q_0,F) where Q is a finite set of states Σ is a finite alphabet $\Delta : Q \times \Sigma \to 2^Q$ is a transition function $Q_0 \subset Q$ is the set of initial states $F \subset Q$ is the set of final accepting states.*

As we consider dynamic systems with possible infinite runs, we give a notion of acceptance that can be applied to both infinite and finite runs of an automaton, a Büchi acceptance criterion.

Definition 2. *An infinite run of an automaton A on an infinite word of Σ^ω is a sequence $\pi = q_0 \cdot q_1, \cdots$ such that $\forall i,\ i > 0 \implies q_i \in \Delta(q_{i-1},\sigma_i)$. A run is said to be Büchi accepting if it contains infinitely many accepting states.*

The properties are given using temporal logic formulas (Pnueli 1977, Huth and Ryan 2004) such as LTL or CTL that allows reasoning on causal and temporal relations of the NFA runs.

2.2 DEVS Formalism

In the classical DEVS formalism, systems or models are described as a collection of one or more components. There are two types of components: atomic (or behavioral) components and coupled (or structural) components. An atomic component defines a simple system that has a state, accepts inputs, produces outputs, and whose behavior depends on external events, the current state, and the time that the system has already spent in the current state.

Definition 3. *A DEVS atomic D is a tuple $(X,Y,S,s_0,t_a,\delta_{ext},\delta_{int},\lambda)$ where X is a set of input events, Y is a set of output events, S is a set of states, $s_0 \in S$ is the initial state, $t_a : S \to \mathbb{R}^+_{0,\infty}$ is the time advance function, where $\mathbb{R}^+_{0,\infty} = \mathbb{R}^+_0 \cup \{\infty\}$, $\delta_{ext} : Q \times X \to S$ is the external transition function, where $Q = \{(s,e) | s \in S, e \in [0,t_a(s)]\}$ denotes the total states set and e is the elapsed time since the last event, $\delta_{int} : S \to S$ is the internal transition function, $\lambda : S \to Y^\phi$ is the output function, where $Y^\phi = Y \cup \{\emptyset\}$ and $\emptyset \notin Y$ is the unobservable event.*

Informally, the operational semantics of a DEVS atomic model is as follows: the model starts in its initial state s_0, and will remain in any given state $s \in S$ until one of the two following cases occurs: (a) an external input $x \in X$ is received, or (b) when the lifespan of s given by $ta(s)$ expires (i.e., the elapsed time e reaches $ta(s)$). In this last case, an output $y \in Y$ is generated as specified by $\lambda(s)$. Generally, the behavior of a DEVS model is given by a simulator that "implements" the model. However, to formally reason about the behavior and properties of a DEVS model, we will use the formal notion of executions or simulation runs of DEVS models, and its traces as defined in (Dacharry and Giambiasi 2005), as an alternative way to formally specify the full behavior implied by DEVS models.

Definition 4. *A finite simulation run of a DEVS model D, is a sequence $\zeta = \upsilon_0 \xrightarrow{x_1} \upsilon_1 \xrightarrow{x_2} \cdots \upsilon_{n-1} \xrightarrow{x_n} \upsilon_n$. Where each υ_i is a function from the real interval $I_i = [0,t_i]$ to the set of total states of D, such that $\forall j, j' \in I_i | j < j'$, if $\upsilon_j = (s,e)$ then $\upsilon_{j'} = (s, e + j' - j)$. Each x_i is an input or output event, and if $(s,e) = \upsilon_{i-1}(sup(I_{i-1})), (s',0) = \upsilon_i(inf(I_i))$, one of the following conditions hold: (1) $x_i \in Y_D, \delta_{int}(s) = s', ta(s) = e$, and $\lambda(s) = x_i$; (2) $x_i \in X_D, \delta_{ext}(s,e,x_i) = s'$, and $e < ta(s)$. We note $states(\zeta) = s_0 \cdot s_1 \cdots s_n$ the sequence of states taken by the finite run ζ of D, and $final(\zeta)$ the final state reached by ζ.*

3 RELATED WORKS

In this section, we will briefly discuss related works that combine formal verification and simulation and show how the proposed approach differentiates itself from these approaches. The idea of combining simulation and formal verification methods has been extensively investigated. In (Goldberg 2008), the author points out that simulation and formal verification are complementary and shows how simulation can be used for verifying sequential circuit by building a sufficient test set. The aim of this type of method is to use formal methods in order to overcome the shortcomings of simulation by building a set of scenarios that will be used to validate the system through simulations. In the opposite, our aim is to use the simulation in order to overcome the shortcomings of the model checking. In (Zeigler and Nutaro 2014), the authors compare the DEVS-based verification approaches, and investigate the combination of simulation and verification. They show that formal verification's implicit assumptions can lead to erroneous results. They then show that simulation can be used to explore the system's parameter space and thus can be used as a way to identify the boundaries beyond which any given proof fails to hold.

Many authors have studied the translation of DEVS models into verifiable models. Previous studies have developed the translation of DEVS model into Timed Automata (Dacharry and Giambiasi 2005; Dacharry and Giambiasi 2007; Inostrosa-Psijas et al. 2016). Other studies (Hwang and Zeigler 2009) reduce the DEVS model into a finite and deterministic model and apply model checking techniques. In (Trojet, Frydman, and Hamri 2009), the authors propose an approach to verify properties like determinism and completeness by encoding some structural properties of the DEVS model into Z specifications. In a similar way, recent work (Yacoub et al. 2017) combines formal methods and simulation by conceiving a language, DEv-PROMELA, for system specification which is able to model the concept described in DEVS and PROMELA. It is then possible to extract both a DEVS model and a PROMELA model from the DEv-PROMELA model. The interesting point of this study is that the authors gave proof about the relation of the two obtained models.

In quite a similar way to our approach, Stuart et al. (2001) define a new analysis methodology called simulation verification. In this approach, the system under study is specified with one formalism, Modechart, a hierarchical graphical specification language for real-time systems. The obtained model is then simulated to generate a simulation prefix. This simulation prefix is then used to constrain the generation of the model computation graph. When deciding which transitions can give rise to successors of a point in the graph, the computation prefix is used to exclude all of the events other than the next transition in the prefix. Thus, only the path corresponding to the simulation prefix is explored. Then, the computation graph is expanded from the end of the simulation path to a frontier, which limits the size of the graph generated. Finally, queries are performed against the generated graph using a Real Time Logic formula. Although this seems quite similar to the proposed approach, there are some differences. In (Stuart et al. 2001), the approach is considered using one formalism, Modechart, which is used to perform both the simulation and the verification phase. On the contrary, we use two formalisms, DEVS for the simulation and PROMELA for the verification. In the proposed approach, simulation is performed in order to lead the system to a so-called critical state from which all reachable states are more likely to contain an erroneous state with regard to the property to be verified. Moreover, the type of property that we seek to guarantee is not the same as the one proposed by the simulation-verification approach. Indeed, they seek to verify timing properties of real-time systems, whereas we want to guarantee some liveness and safety properties.

4 COMBINED APPROACH: DEVS AND MODEL-CHECKING

In this section, we will introduce in detail the different parts of the developed approach. The novelty presented in the paper does not lie in the fact of combining simulation and model checking but in the way of combining them through DEVS and PROMELA. Moreover, as DEVS is a pivot formalism of discrete event simulation, it can extends the scope of our method to other discrete event simulation formalism. As a simulation provides quantitative or qualitative measures of the model's behavior that cannot be obtained via a query of the NFA, it can enable the model checking procedure to focus on a subset of the state space, thus limiting the state explosion problem. In the rest of the paper, we will consider the verifiable class of DEVS unless mentioned otherwise.

To guarantee that a system meets a given property, the ad hoc model checking approach models the system using an NFA, then specifies the property using a logical formula and finally checks whether the property is satisfied for all possible executions of the modeled system. As previously mentioned, the model checking objective is to explore the total state space of a given model. Nevertheless, as this state space grows exponentially with the number of components, the widespread approach is to make abstractions and then to explore the reduced state space. The search stops when a path $\pi = \sigma_0 \cdot \sigma_1 \cdot \sigma_2 \cdots \sigma_n$ does not satisfy the property, or when the full state space has been explored. However, with this approach, the combinatorial state explosion problem arises for realistic complex systems. Thus, the full state space exploration will be unmanageable due to the considerable size of the state space and the model checker will exhaust all its resources (e.g., memory, time) before providing any answer. However, to verify a property, the designer can make the following assumption: among the set of all possible executions of the system, there is a subset for which the likelihood of the property being unsatisfied is more significant. This idea that a system has better chances of encountering an error in the boundaries of its operating domains is well known in software system verification and testing (Whittaker 2000, Myers, Sandler, and Badgett 2011). However, the difference with our approach lies in the fact that after having targeted a subspace where an error is more likely, a test set will only increase our confidence in this subspace, without guaranteeing its absence. We argue that, after having restricted the search for a subspace, it is more interesting to guarantee that this subspace is without errors. This leads to a more conservative approach in exploring state spaces. An important question is how to identify this set of states. Assigning a value to every state variable in such a way that this valuation corresponds to an actual possible configuration of the real system is a complicated task. Obviously, in the case of a simple system, the modeler can directly initialize the model in one of these "critical" states. However, when considering a complex system with a number important of variables evolving in a dependent way, this task can be extremely difficult or even impossible. It is precisely here that the simulation can play an advantageous role. Indeed, if the system to be verified is modeled and simulated, then the user does not need to know all the variable values of the system when it is in a critical state, but only a scenario that leads the system to a critical state. In other words, the critical state is only partially defined through threshold values for certain variables. That is, we do not know all the values of the variables a priori, but only the values of a part of the model's variables, which makes the state of the system critical. Therefore, it is thanks to the simulation that we will reach a state that violates these threshold values and that we will obtain a value for any variable of the model. The overall approach can be summarized in four phases: (*a*) the modeling phase, in which both the NFA and the DEVS model are specified; (*b*) the simulation phase, where a critical state is identified; (*c*) the projection phase, where the identified critical state is "projected" as the initial state of the NFA; and (*d*) the verification phase, in which model checking is performed on the NFA resulting from the previous phase.

4.1 Modeling Phase

The first phase of the approach is the modeling phase. In this phase, the discrete event simulation model and the model used for model checking model of the system are built. Here, the simulation model is specified with the DEVS formalism and the model used for checking model is specified with the modeling language of the SPIN model checker, PROMELA. However, the approach can be considered with any discrete event simulation and any model checker. A strong assumption of the approach is that the PROMELA model satisfies the same LTL properties than the DEVS model. We do not give here a systematic procedure to guarantee that both models meets this assumption. Nonetheless, some authors have examined the proof of equivalence between DEVS and PROMELA models (Yacoub et al. 2017, Yacoub, Hamri, and Frydman 2017) and more generally between two formalisms (Zeigler 2018, Zeigler et al. 2019).

Now, let S be a real system, P be the PROMELA model of S and φ a property that all states of P have to meet. We then denote D as the DEVS model of S. As we consider dynamic systems, simulation runs can be infinite, therefore we want the simulation to stop on the first critical state reached during a run.

Definition 5. *Let D be a DEVS model. A stop condition $C_{critical} : Q_D \to \{true, false\}$, is a function that maps each element of Q_D to a truth value, where Q_D is the total state set of D.*

Giving the stop condition, we then define the set of critical states regarding a condition C as follows:

Definition 6. *The set of critical states of a DEVS model D and a condition $C_{critical}$, is given by $critical(D, C_{critical}) = \{s \mid s \in Q_D, C_{critical}(s) = true\}$, where Q_D is the total state set of D.*

The state set S_D of the DEVS model D can be defined as $\times_{i=0}^{n-1}(dom(v_i))$, where n is the number of variables defining a state, and $dom(v_i)$ denotes the domain of the variable v_i, the condition $C_{critical}$ is then:

$$C_{critical}(s) = \begin{cases} true, & \text{if } v_i > threshold \\ false, & \text{otherwise.} \end{cases}$$

Where v_i is a variable defining the system state and $threshold \in dom(v_i)$ is a value in the domain of v_i. One can also express a more elaborate condition by defining it over a set of variable. However, considerable attention must be paid when defining the condition. Indeed, the variable v_i is not chosen randomly. The variable is selected because it has a direct or indirect impact on the property φ that we want to verify later on the PROMELA model, in the verification phase. It is interesting to note that, as the observed variable v_i is part of the simulation model, v_i can be dependent on the time which cannot be the case in the untimed formal verification model. The utility of such information available during the simulation is therefore highlighted, as it can serve the model checking. Also note that, $|critical(D, C_{critical})| > 0$. Otherwise, if there is no state identified as critical, the simulation continues indefinitely. This modeling phase sets the stage for the next step, the simulation phase.

4.2 Simulation Phase

In this phase, the simulation of the DEVS models D is executed by a simulator in order to reach a critical state $s \in critical(D, C_{critical})$. In order to run the simulation, an important question is: how to get the scenario that will lead the simulation run to a critical state? Indeed, if the scenario does not lead to a critical state, the simulation could continue indefinitely and the model checking will never run. Two possibilities can be considered. The first one is that when the expert expresses the condition $C_{critical}$ that defines the set of critical states, he also gives scenarios that will lead to a critical state. The other interesting possibility can be considered in the case of systems for which historical data are available. For these systems, one could use the recorded data that has previously led the system to a state that does not meet the condition $C_{critical}$ and use one of these scenarios to perform the simulation. During the simulation run, the system state is checked before each event transition (internal and external) and stopped when the simulation run reaches a critical state. Thus, the simulation results in a run $\zeta = \upsilon_0 \xrightarrow{x_1} \upsilon_1 \xrightarrow{x_2} \cdots \upsilon_{n-1} \xrightarrow{x_n} \upsilon_n$, where $final(\zeta) \in critical(D, C_{critical})$ is the first critical state in $states(\zeta)$. As soon as these phases have been carried out, the approach proceeds with the projection phase.

4.3 Projection Phase

The aim of this phase is to *project* the critical state $final(\zeta)$ of the simulation run ζ as the initial state of the PROMELA model P. This phase is closely related to the modeling phase where the two models were built. Indeed, the assumption was that the two models satisfy the same properties. However, as the DEVS model is more detailed than the PROMELA model, for a given DEVS model state there is only one equivalent state in the NFA represented by the PROMELA model. Note that the reverse is not true.

Definition 7. *A projection from a state of a DEVS model D to a state of a PROMELA model P is a function* $projection : Q_D \rightarrow Q_P$*, such that* $\forall s \in Q_D, \exists q \in Q_P, projection(s) = q$

The projection maps a state of the DEVS model to a state of the NFA represented by the PROMELA model. An idealized illustration of the projection phase is given in Figure 1, where a one-to-one mapping between the state space of the DEVS model and the PROMELA model is assumed. However, in order to perform the projection, several points must be considered. The first point is that we have to take into account that variables defined in PROMELA models must take their values in a predefined restricted range and some variable cannot be represented. For example, PROMELA does not allow to take into account the real numbers. This can be seen as a limitation in the sense that systems with real variables cannot be represented in PROMELA and therefore cannot be verified. Another point is that all variables that are time-dependent in the simulation model are abstracted away when performing the projection. For all other variables, a one-to-one variable mapping is performed. At the end of the projection step, the initial state q_0 of the PROMELA model is defined as $q_0 = projection(final(\zeta))$.

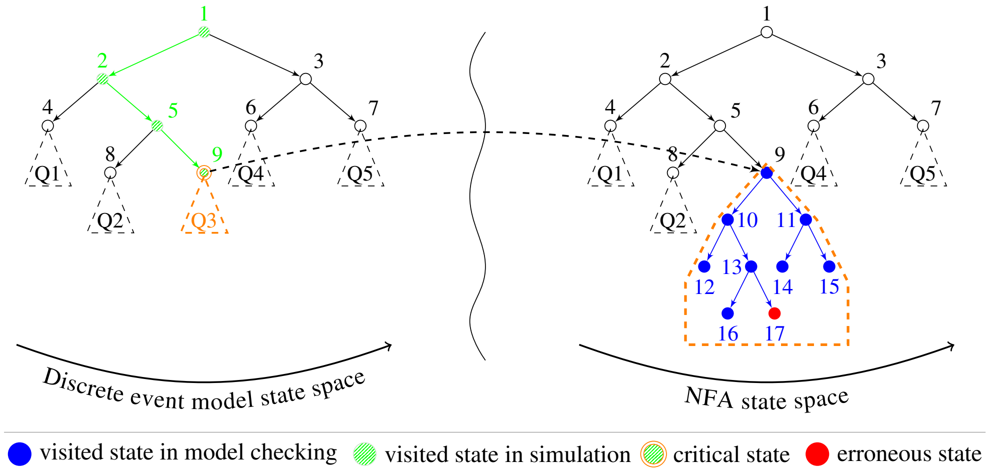

Figure 1: Illustration of the state space exploration with the combined approach.

4.4 Verification Phase

In the last step of the approach, the obtained PROMELA model from the previous phase, and the property φ are provided as inputs to the model checker. From there, three possibilities must be considered: (a) the model checker finds an execution of the system for which the property is not verified; (b) the formal verification ends without finding a state that does not respect the property, which means that the system meets the property; and, (c) the model checking exhausts all the resources allocated to it without concluding, in which case the approach does not allow to conclude on the fact that the property is always satisfied. In the first two cases, with the proposed approach, the model checker is able to terminate, which may not have been the case if the ad hoc model checking method was performed directly. Even though the ad hoc model checking would have been performed directly and terminated, we argue that, in the best case, the proposed approach would have explored fewer states and transitions than the ad hoc model checking approach, and the same number of states and transitions in the worst case. Indeed, when considering dynamic systems with infinite runs, it is possible that the model checking procedure will eventually explore the total state space, due to the presence of cycles. For example, if the system eventually returns to the initial state of the DEVS model, the entire state space will be explored, and the performance of the proposed approach will be the same as the classical model checking approach. However, with the proposed approach, the exploration will have started on a subset that is more likely to contain an error, and if no error was found on this subset, the model checking procedure will continue, as usual, to explore the state space.

The previous sections have introduced the related works and detailed the developed approach. In the next section, we will introduce a case study using the developed approach and compare it to the classical ad hoc model checking approach based on the SPIN model checker.

5 NETWORK-ON-CHIP, A USE CASE

In this section, we will consider a Network-On-Chip (NoC) system in order to illustrate the proposed approach. We will first present some NoC concepts. Then, we will compare the proposed approach to the classical ad hoc model checking procedure. Lastly, we will make a conclusion and provide some opportunities for future research.

5.1 The NoC Concepts

A NoC is a communication subsystem on an integrated circuit (chip), usually between Intellectual Property (IP) cores in a system on a chip. In order to overcome typical bus limitations (e.g., scalability), NoCs have been developed to be more scalable as they provide a much larger amount of communication resources. The NoCs typically consist of nodes (IP cores), routers, network interfaces and connections. Routers direct the data according to the protocol selected. Connections are channels of data transmission between the elements to the network. Finally, network interfaces make the logic connection between the IP cores and the network. The way in which routers, network adapters and connections are organized is determined by a given topology, the most commonly used one is the 2D mesh topology. Interested readers may refer to (Cota et al. 2012) for a more detailed and comprehensive description of NoC concepts. Numerous properties can be verified using a formal model, like performance evaluation or communication correctness. Here, we will consider NoC correctness properties such as deadlock freedom.

The NoCs used in this example are a 2×2 and a 3×3, mesh network as in (Ahmadinejad, Refan, and Sarjoughian 2011). The routing strategy is a deterministic X-Y algorithm. Each router input port has a dedicated buffer. A round-robin policy is performed to select the next input port. The simulator used to perform the simulation is the FwkDEVS, and we used the model checker SPIN Version 6.4.8 for deadlock safety property. In order to test the correctness of the approach, a fault was added in both of the PROMELA and DEVS models by modeling defective IP. The objective is to find if, given a set of messages, it exists an execution of the system leading to a deadlock situation and thus to a complete system failure. Two (resp. three) messages are send in the 2×2 (resp. 3×3) NoC. Kindly note that the source code and all the models used in this study can be found at https://github.com/khemiriabdelhak/NetworkOnChip

The PROMELA model M_P is obtained by the following rules: Each router is modeled with a process. Each process consists of two variables : flit which holds the current flit, and cur_id that corresponds to the current channel index from which flits are received. The process begins in a check state, where the current input channel is checked for incoming flits. If there are no incoming flits in the current channel, the next input channel is selected. Otherwise, if there are an input flits, the first (i.e., the header) flit is routed with an X-Y routing algorithm to the appropriated channel, then all the subsequent incoming flits from the channel are routed, until the last flit (i.e., the tail) is routed. Links between nodes are modeled with channels. As IPs are just passive components, they are abstracted away. A defective IP is modeled as a channel that is never read. Flits are represented by a user-defined data type (i.e., msg_flit) that hold the type of the flit (header, payload, tail), a value that represents the data, and the destination IP. Note that a flit stands for FLow control unIT, a part of a packet. In the PROMELA model M_P, for each message, a cyclic process is added to send the message's flits over the network and terminates when all the message's flits are sent. In the DEVS model M_D, for each message, a generator is added to send the message's flits over the network.

To set the condition $C_{critical}$ that will define the set of all critical states it is interesting to note that for this example, the more the network is congested, the higher are the odds of being in deadlock situations. Hence, it would be interesting to check with the model checker only the subset of executions where the network is congested. In order to define the set of critical state, two conditions are possible: (a) a weak condition, which identifies the states where there is one buffer that contains $N-1$ elements; and (b) a strong condition, which corresponds to the states where all buffers contains $N-1$ element. Where N is the capacity of the buffer. In this example we will only consider the weak condition. $C_{critical}$ is then defined as:

$$C_{critical}(s) = \begin{cases} true, & \text{if } \bigvee_{i=0}^{k-1} (size(buf_i) == capacity(buf_i) - 1) \\ false, & \text{otherwise.} \end{cases}$$

Where, k is the total number of buffers, buf_i is the i^{th} buffer and $\bigvee$ the logical *or* operator. At the end of the simulation of the DEVS model M_D, the resulting simulation run is $\zeta = q_0 \xrightarrow{t_0} q_1 \xrightarrow{t_1} q_1 \cdots \xrightarrow{t_n} q_c$ such that, the state $final(\zeta) \in critical(D, C_{critical})$ is a critical state, (i.e., a buffer contains $N-1$ elements). The state $final(\zeta)$ is "projected" as the initial state of the NFA used by the model checking procedure. The obtained PROMELA model is then given to the SPIN model checker and the results are compared to the ad hoc model checking procedure of the initial PROMELA model.

5.2 Results

We performed a comparison between the proposed approach and the ad hoc classical model checking approach with the SPIN model checker. The experiments were performed on a computer with 2.6 GHz Intel Core processor, and 8 GB RAM. The two approaches are compared regarding the number of visited states, the number of transitions performed, the memory used and the total time of the search. For each model, the experiments were performed with a depth-first-search exhaustive exploration, and the partial order reduction enabled. The physical memory available was set to 5120 MB, the maximum search depth was set to 10^6 and it was never reached. The results are summarized in Table 1.

In the two considered NoCs, the ad hoc model checking was unable to conclude whether the property P holds or not, and exhausted all resources. However, with the proposed approach, the model checking was able to detect a deadlock situation. Indeed, with the ad hoc model checking, the model checker has to consider all possible interleaving between the processes and find which one lead to a situation where all processes are blocked. For the 2×2 NoC, the ad hoc model checking explored more than 4 million unique states, performed about 7 million transitions in 29 seconds and reached the limit of 5GB of memory without finding the error. In the combined approach, a simulation is performed and stopped when a buffer size is equal to $N-1$. The state of all buffers and variables are then set in the PROMELA model as initial state and the model checking is performed. With the combined approach, a deadlock was found in less than one second with less than 1000 unique states explored, less than 1000 transitions performed and 129

Table 1: Comparison of the combined approach with the ad hoc Model Checking (MC).

	2×2 NoC		3×3 NoC	
	ad hoc MC	Combined approach	ad hoc MC	Combined approach
Error Found	None(Not finished)	Deadlock Found	None (Not Finished)	Deadlock Found
States	4 293 012	606	1 655 692	1 870
Transitions	7 870 814	618	3 362 026	1 936
Memory Usage (MB)	5 119.940	129.511	5 119.687	188.047
Simulation time (s)		2.16		4.57
MC Time (s)	29.3	~ 0	14.3	0.01

MB of used memory. For the 3×3 NoC, the ad hoc model checking exhausted all resources and did not find an error before stopping the search. The search was stopped after 14.3 seconds and 5GB of memory used, corresponding to 1 655 692 unique states explored and 3 362 026 transitions performed. The ad hoc model checking is outperformed by the combined approach that did find the deadlock looked for. Indeed, the combined approach used less than 190 MB of memory, performed 1936 transitions and reached 1870 unique states in less than one second. As expected, the experiments show that in both considered models, the combined approach limited the state space explosion of the ad hoc model checking and thus successfully detected a deadlock situation.

6 CONCLUSION AND DISCUSSIONS

In this paper, we proposed an approach combining model checking and simulation in order to mitigate the well-known state space explosion shortcoming of model checking. Simulation, which provides quantitative and qualitative measures, can be used to identify a subset of states where a given property is more likely to be unfulfilled, thus allowing the model checking procedure to focus on a subset of the total state space. Results so far have been very promising and show that the approach can lead to large savings in terms of computation time, number of transitions performed, visited states and memory used. Besides, it is interesting to note that the verification of models that are close to the implementation is a difficult task for the model checking method. Although considerable progress has been made in order to overcome the shortcomings of model checking, we believe that the proposed approach could be helpful for verifying complex systems where expert knowledge can be used to identify from which states of the system a specific property is more likely to be unfulfilled.

This study has potential limitations. Indeed, the proposed approach stands or falls with the skills of the designer to create at the same time a PROMELA model and a DEVS model. One could argue that we could have reduced the development effort by using DEv-PROMELA to derive the DEVS and the PROMELA models. By using DEv-PROMELA, the generated PROMELA model risks being too detailed for the model checking. Indeed, the PROMELA model must be generated in such a way that it is abstract enough to allow verification of the desired property while ensuring that the abstraction preserves the property that one wishes to verify. Another solution would be to model the system in an abstract way (i.e., omitting details of the real system). However, the derived DEVS model would be too abstract for a simulation that should be faithful to the real system. This eventually pushes us to have two models and implies that the abstraction must be done by the modeler according to the property to check. An option could be to use DEv-PROMELA to obtain a first PROMELA model and let the modeler abstract it according to the property to be verified. Another limitation is the absence of guidance to highlight critical states of the system to check. Nevertheless, some solutions exist in the literature, like Bayesian networks, stochastic models, etc., which may facilitate the identification of such states.

ACKNOWLEDGMENTS

We would like to thank the reviewers for their constructive and insightful feedback. This work is supported by STMicroelectronics Rousset, France.

REFERENCES

Ahmadinejad, H., F. Refan, and H. Sarjoughian. 2011. “NoC simulation modeling in DEVS-suite”. In *Simulation Series*, Volume 43, pp. 134–139. Society for Computer Simulation International.

Clarke, E. M., and E. A. Emerson. 1981. “Design and synthesis of synchronization skeletons using branching time temporal logic”. In *Workshop on Logic of Programs*, pp. 52–71. Springer.

Clarke, E. M., W. Klieber, M. Nováček, and P. Zuliani. 2012. *Model Checking and the State Explosion Problem*, pp. 1–30. Berlin, Heidelberg, Springer Berlin Heidelberg.

Cota, É., A. de Morais Amory, and M. S. Lubaszewski. 2012. *NoC Basics*, Chapter 2. Boston, MA, Springer US.

Dacharry, H. P., and N. Giambiasi. 2005. “Formal Verification with Timed Automata and DEVS Models: a case study”. In *Proceedings of Argentine Symposium on Software Engineering*, pp. 251–265.

Dacharry, H. P., and N. Giambiasi. 2007. “A Formal Verification Approach for DEVS”. In *Proceedings of the 2007 Summer Computer Simulation Conference*, SCSC ’07, pp. 312–319, Society for Computer Simulation International.

Goldberg, E. 2008. “On Bridging Simulation and Formal Verification”. In *Verification, Model Checking, and Abstract Interpretation*, edited by F. Logozzo, D. A. Peled, and L. D. Zuck, pp. 127–141. Berlin, Heidelberg, Springer Berlin Heidelberg.

Holzmann, G. 2003. *The Spin Model Checker: Primer and Reference Manual*. First ed. Addison-Wesley Professional.

Holzmann, G. J. 1997. “The model checker SPIN”. *IEEE Transactions on software engineering* vol. 23 (5), pp. 279–295.

Huth, M., and M. Ryan. 2004. *Logic in Computer Science: Modelling and Reasoning About Systems*. New York, NY, USA, Cambridge University Press.

Hwang, M. H., and B. P. Zeigler. 2009. “Reachability Graph of Finite and Deterministic DEVS Networks”. *IEEE Transactions on Automation Science and Engineering* vol. 6 (3), pp. 468–478.

Inostrosa-Psijas, A., V. Gil-Costa, G. Wainer, and M. Marín. 2016. “Formal Verification of DEVS Simulation: Web Search Engine Model Case Study”. In *Proceedings of the Summer Computer Simulation Conference*, SCSC ’16, pp. 1–8, Society for Computer Simulation International.

Myers, G. J., C. Sandler, and T. Badgett. 2011. *The art of software testing*. Wiley Publishing.

Pnueli, A. 1977. “The Temporal Logic of Programs”. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, SFCS’77, pp. 46–57. Washington DC, USA, IEEE Computer Society.

Queille, J.-P., and J. Sifakis. 1982. “Specification and verification of concurrent systems in CESAR”. In *International Symposium on programming*, pp. 337–351. Springer.

Stuart, D. A., M. Brockmeyer, A. K. Mok, and F. Jahanian. 2001. “Simulation-verification: biting at the state explosion problem”. *IEEE Transactions on Software Engineering* vol. 27 (7), pp. 599–617.

Trojet, M. W., C. Frydman, and M. E.-A. Hamri. 2009. “Practical Application of "Lightweight" Z in DEVS Framework”. In *Proceedings of the 2009 Spring Simulation Multiconference*, SpringSim ’09, pp. 1–8. San Diego, CA, USA, Society for Computer Simulation International.

Wang, Y., G. Zacharewicz, D. Chen, and M. K. Traoré. 2015. "A proposal of using DEVS model for process mining". In *27th European Modeling & Simulation Symposium (Simulation in Industry)*, pp. 403–409.

Whittaker, J. A. 2000. "What is software testing? And why is it so hard?". *IEEE Software* vol. 17 (1), pp. 70–79.

Yacoub, A., M. e. A. Hamri, and C. Frydman. 2017. "Restricting DEv-PROMELA with a Hierarchy of Simulation Formalisms". In *Proceedings of the Symposium on Theory of Modeling & Simulation*, TMS/DEVS '17, pp. 1–11. Society for Computer Simulation International.

Yacoub, A., M. E. A. Hamri, C. Frydman, C. Seo, and B. P. Zeigler. 2017. "DEv-PROMELA: an extension of PROMELA for the modelling, simulation and verification of discrete-event systems". *International Journal of Simulation and Process Modelling* vol. 12 (3-4), pp. 313–327.

Zeigler, B., and J. J. Nutaro. 2014, January. "Combining devs and model-checking: Using systems morphisms for integrating simulation and analysis in model engineering". *26th European Modeling and Simulation Symposium, EMSS 2014*, pp. 350–356.

Zeigler, B. P. 2018. "The role of approximate morphisms in multiresolution modeling: Can we relax the strict lumpability requirements?". *Journal of Defense Modeling and Simulation* vol. 15 (4), pp. 495–498.

Zeigler, B. P., D. Kim, and S. J. Buckley. 1999. "Distributed Supply Chain Simulation in a DEVS/CORBA Execution Environment". In *Proceedings of the 31st Conference on Winter Simulation: Simulation—a Bridge to the Future - Volume 2*, WSC '99, pp. 1333–1340. New York, NY, USA, ACM.

Zeigler, B. P., A. Muzy, and E. Kofman. 2019. "Chapter 17 - Verification, Validation, Approximate Morphisms: Living With Error". In *Theory of Modeling and Simulation* (Third Edition ed.). Academic Press.

AUTHOR BIOGRAPHIES

ABDELHAK KHEMIRI is a Ph.D. student at Aix Marseille Université and a member of the Laboratoire d'Informatique et des Systèmes (LIS), Marseille, France. He is also a member of the Technical Staff with STMicroelectronics Process Control group, Rousset. His email address is abdelhak.khemiri@lis-lab.fr.

MAAMAR EL AMINE HAMRI is an Associate Professor at Aix-Marseille Université. He is also a member of Laboratoire d'Informatique et des Systèmes (LIS), Marseille, France. He has been active for many years in Modeling and Simulation research area. He has participated in various international conferences on modeling and simulation. His email address is amine.hamri@lis-lab.fr.

CLAUDIA FRYDMAN is a full Professor at Aix-Marseille Université. She is also a member of the Laboratoire d'Informatique et des Systèmes (LIS), she has been a referee for several scientific journals and a member of the program committee in various international conferences. She has been active for many years in knowledge management and currently her research is focusing especially on researches on knowledge based simulation. Her email address is claudia.frydman@lis-lab.fr.

JACQUES PINATON is the manager of Process Control System group at STMicroelectronics Rousset, France. He is an engineer in metallurgy from the Conservatoire National des Arts et Métiers d'Aix en Provence. He joined ST in 1984. After 5 years in the process engineering group, he joined the device department to implement SPC and Process Control methodology and tools. He participated in the startup of 3 fab generations. He is leading various Rousset R&D programs on manufacturing science including programs on automation, APC, and diagnostics. His email address is jacques.pinaton@st.com.