Human-centric Computing and Information Sciences

RESEARCH Open Access

# A symbolic model checking approach in formal verification of distributed systems

Alireza Souri[1†], Amir Masoud Rahmani[1*†], Nima Jafari Navimipour[2†] and Reza Rezaei[3†]

*Correspondence: rahmani@srbiau.ac.ir
†Alireza Souri, Amir Masoud Rahmani, Nima Jafari Navimipour and Reza Rezaei contributed equally to this manuscript
[1] Department of Computer Engineering, Science and Research Branch, Islamic Azad University, Tehran, Iran
Full list of author information is available at the end of the article

**Abstract**

Model checking is an influential method to verify complex interactions, concurrent and distributed systems. Model checking constructs a behavioral model of the system using formal concepts such as operations, states, events and actions. The model checkers suffer some weaknesses such as state space explosion problem that has high memory consumption and time complexity. Also, automating temporal logic is the main challenge to define critical specification rules in the model checking. To improve the model checking weaknesses, this paper presents Graphical Symbolic Modeling Toolkit (GSMT) to design and verify the behavioral models of distributed systems. A behavioral modeling framework is presented to design the system behavior in the forms of Kripke structure (KS) and Labeled Transition System (LTS). The behavioral models are created and edited using a graphical user interface platform in four layers that include a design layer, a modeling layer, a logic layer and a symbolic code layer. The GSMT generates a graphical modeling diagram visually for creating behavioral models of the system. Also, the temporal logic formulas are constructed according to some functional properties automatically. The executable code is generated according to the symbolic model verifier that user can choose the original model or reduced model with respect to a recursive reduced model. Finally, the generated code is executed using the NuSMV model checker for evaluating the constructed temporal logic formulas. The code generation time for transforming the behavioral model is compared to other model checking platforms. The proposed GSMT platform has outperformed evaluation than other platforms.

**Keywords:** Model checking, Temporal logic, Reduced model, Kripke structure, Labeled Transition System

## Introduction

Today, distributed systems have developed complex components more and more [1, 2]. By increasing performance of complex systems such as service composition [3], task scheduling [4] and fault tolerance [5], simulation analysis cannot evaluate entire of the system levels [6, 7]. Also, the simulation results are rested to some design under test platforms [8] that omit the part of the existing state space of the system [9]. Formal verification is a mathematically correctness provable approach for the complex distributed systems which is well-suitable for NP-hard problems [10, 11]. Recent scientific studies analyzed their case studies using mathematical verification approaches such as model checking [4, 12–18], process algebra [19–24], formal concept analysis [25] and theorem proving [26–29] methods.

Among the mentioned approaches, model checking [30] is a well-known verification technique to evaluate the functional properties of a distributed system automatically [31, 32]. The main goal of the model checking is to find the property violations and limitations in the system behavior with the counterexamples [33]. However, there are some limitations for model checking such as state space explosion and temporal logic design [34]. For improving these limitations, the symbolic model checking [35, 36] with Binary Decision Diagram (BDD) has been presented by McMillan [34]. Some industrial tools such as NuSMV [37], PAT [38], Spin [39], and UPPAAL [40] are well-known for analyzing the system behavior correctness [41–43]. But, these tools have some limitations such as weak graphical user interface, the complexity of programming language and generating the automated temporal specification rules for verifying the system behavior [44–47]. To illustrate the temporal logic formulas, some model checkers such as NuSMV have supported the generated specification rules in forms of Linear Temporal Logic (LTL) and Computation Tree Logic (CTL) [17, 37, 48–50]. Also, creating a critical specification rule for checking in the generated state space of the system behavior is an important challenge for model checkers. When the state space increases exponentially, checking and discovering the critical specification rules to measure the correctness of the system is confused [51, 52]. As yet, model checkers do not guarantee automated specification rules generation [53, 54]. In addition, a model checker needs to automated formal design that supports the Kripke structure (KS) and Labeled Transition System (LTS) modeling methods. In model checking, some characteristic points consolidate an irrefrangible relationship between integrated abstract model and the concrete system behavior. The characteristic points include specifying descriptive features, designing precise model, configuring desired feature selection, and generating comprehensive specification rules. This relationship is confident that the correctness of the integrated abstract model using model checking is very reliable to evaluate concrete system behavior. If we emphasize some characteristic points for designing and modeling system behavior [55], then the accurate verification results are obtained from model checking.

This paper presents an easy to use and user-friendly Graphical Symbolic Modeling Toolkit (GSMT) to simplify model checking the system behavior. We advocate the use of fully automated designing methods to check the correctness of the system behavior. The refinement of design, modeling and verification levels lead the behavior correctness procedure to increase the accuracy. An integrated architecture is also designed for each level according to the simple relationship among the existing objects of the proposed framework. This framework not only follows the contributions of the existing model checkers but also adds some important points to verify the system behavior using model checking. The contributions of this research are as follows:

- Presenting a graphical model checking framework to facilitate the system behavior design.
- Providing a modeling platform to support the KS and LTS models.
- Generating the LTL and CTL specification rules of the system model according to the functional properties such as deadlock, reachability, and safety conditions.

- Presenting a high-level order of recursive reduced Kripke and labeled models to ameliorate the state space explosion problem.
- Facilitating the verification procedure using NuSMV.

The paper structure is organized as follows, "Related work" section illustrates a brief review of the presented related frameworks and toolsets. In "GSMT framework" section, we address a conceptual explanation of the GSMT framework. Also, this section introduces the current four layers in the automated verification approach. Moreover, the formal descriptions of the system behavior are illustrated to handle the model checking the specification rules. "Experimental analysis" depicts a descriptive case study to evaluate the verification procedure for the proposed framework with the other approaches according to some experimental results. Finally, "Conclusion and future work" provides the conclusion and some open subjects on this topic as the future works.

## Related work

In this section, some related studies are discussed briefly which contain modeling and descriptive translators and automated verification frameworks according to some important features and challenges.

Castelluccia et al. [56] presented a formal framework to design web applications according to the UML method. The key feature of this framework is based on LTS model checking and CTL formulas. First, a design of the model is generated in forms of the UML-based platform with the *XMI* format. Then, the framework translated the proposed UML-based platform to the extensible *SMV* codes.

Li et al. [54] proposed a translator framework to exchange Programmed Logic Controllers (PLC) for executable verification codes using utility block chart language. The framework presented a formal modeling approach to specifying the model structure using a Boolean explanation method. The model is translated to some modules of *SMV* codes. This translator supports just CTL formulas to embed in code generation. Designing the model structure is not automatic because the extensibility of the model checking approach is covered. Also, this framework supports a command-line authentication to avoid invalid inputs according to its powerful editor environment. The main disadvantages of this framework are as follows: the requirement patterns as the specification rules are input manually; the LTL formulas are not supported; the framework has not illustrated the correctness of the functional properties such as reachability and deadlock.

Abdelsadiq [57] presented a high-level modeling framework for Contractual Business-to-Business relations (CB2B) to apply e-contract models in the e-business management system. The CB2B models support a set of the conceptual model that includes truths, actions, responsibilities and exclusions for checking contract agreement. First, the designed model translated to Event–Condition–Action (ECA) structure according to Process Metalanguage (Promela) language. Then, a set of simple LTL formulas is generated manually. Both temporal specifications and ECA model are translated to executable codes for the Spin model checker. The main limitations of this framework are as follows: (1) the design level of the formal modeling is omitted; (2) specification rules are very simple; (3) an editable platform for user interface has not been indicated.

Caltais et al. [58] proposed a framework conversion to interact between the System Modelling Language (SysML)-based models and NuSMV symbolic model checker. The SysML-Ja is a toolset that translates the structural SysML-based models in forms of block diagrams and state diagrams to symbolic modules of *SMV* codes. This translation is retrieved from the LTS model by some events and actions. The relationships between each block/state diagram are converted to a transition command in *SMV* code. Some specification rules are input at the end of the *SMV* codes manually. There are some limitations in this framework as follows: (1) the generation of specification rules has not been considered in the structure of the framework; (2) the graphical modeling stage is omitted in this framework.

Furthermore, Deb et al. [59] have presented an inherent sequence state transition modeling transformation framework for concurrent systems. They used the Naive algorithm to handle the rise of the state space. First, requirements are translated to the LTS model with respect to a set of sequences states. In the editor environment, the LTS model is converted according to the Multi-dimensional Lattice Paths (MLP) to the *SMV* codes. The framework can add a simple CTL formula to the generated *SMV* code to verify it. However, when a large model is loaded in this framework, the state space has been increased highly. When the system behavior has a multi-tenant structure, the translated modules cannot interact with them by transition methods. In addition, the functional properties have not been verified in this framework using NuSMV.

Meenakshi et al. [60] have presented a converter environment between Simulink models and input language of model checkers automatically. The system engineers can develop the structural models in Simulink environments such as MATLAB informally. Hence, this converter tool can be useful to transform the Simulink model as the input to a formal description approach in forms of NuSMV model checker codes. The proposed tool covers all of the block diagrams that organize the structural model of the Simulink. There are some limitations in this tool compared with the other instruments: the LTL specifications are not considered in this tool to translate into *SMV* codes; a graphical modeling diagram is not illustrated to avoid the state space examination. In addition, the practical feature of this model does not support a complex industrial model for translating to the *SMV* codes.

Vinárek et al. [61] proposed a translator framework between use case models and NuSMV model checker. The authors described a formal explanation of the Formal verification of Annotated Models (FOAM) framework using a user/actor model. The use case model is converted to a textual behavior automaton based on a priority connection. The textual behavior automaton is translated to a configurable LTS model [62]. The main disadvantages of this tool are as follows: first, this translator has not the editor environment to illustrate code generation; second, this tool has not covered the LTL specifications for checking the correctness of the use case models. There is just a demo environment for this tool rather than a practical translator environment.

Szwed [63] presented a translator plugin to convert a business model to executable model checking code. This plugin specifies all of the direct elements of the business model that connect with each symbolic state in the business layer. In translation procedure, a set of the business processes are specified as the atomic states and the business tasks are specified as the events. The CTL formulas are added by the user

**Table 1 Comparison of the related frameworks according to the verification structure**

| Framework | Case study | Modeling | Design | Temporal logic/ generation | Model checker |
|---|---|---|---|---|---|
| WAVer [56] | Web applications | LTS | UML | CTL/manual | NuSMV |
| FBD [54] | Industrial controllers | LTS | Not supported | LTL/manual | NuSMV |
| PLC [57] | E-Business system | LTS | Not supported | LTL/manual | Spin |
| SysML-ja [58] | Concurrent system | LTS | UML | Not supported | NuSMV |
| IStar [59] | Concurrent system | LTS | Not supported | CTL/manual | NuSMV |
| SimuLink [60] | Concurrent system | LTS | Not supported | CTL/manual | NuSMV |
| FOAM [61] | Concurrent system | LTS | UML | CTL/manual | NuSMV |
| ArchiMate [63] | Concurrent system | LTS | UML | CTL/manual | NuSMV |
| S2Nusmv [64] | Concurrent system | LTS | Not supported | LTL/manual | NuSMV |
| Petrinet2smv [65] | Concurrent system | KS | Petri net | CTL/manual | NuMSV |
| Our approach | Concurrent system | LTS/KS | UML/graph | LTL-CTL/automatic | NuSMV |

manually. A graphical model is presented after translating *SMV* codes. Some limitations of this plugin are as follow: The verification method is executed without any correctness procedure; also, the LTL formulas are not supported. However, the execution time and reachability states are not compared with the other frameworks.

Jiang and Qiu [64] have proposed an Spin2NuSMV (S2N) converter framework between Spin models and NuSMV codes. This framework presents a conversion procedure for transforming a high-level model in forms of Promela language into a low-level model as a state transition system in *SMV* code. Each process in the Spin model has been translated to a state with events coverage asynchronously. However, this framework cannot support the temporal logic transformation since NuSMV covers both LTL and CTL logics and Spin just generates LTL logic in the opposite. In addition, when a complex model is transformed into the *SMV* codes, some channels connection between processes are omitted.

Szpyrka et al. [65] have presented a translator framework to convert state graph of a colored Petri-net model to an executable *SMV* code. Each net is converted to a state and each guard is transformed into an atomic proposition. The translated model is shown in forms of a Kripke model in NuSMV. A graphical reachability graph is generated after the translation procedure that is very confused and irregular. Also, the translated model is not displayed as a graphical model. This tool has a simple environment that imports a Petri-net model and translates to the *SMV* code in editor environment. The temporal logic formulas are added to end of the code manually. Also, the timed-Petri-net models cannot translate to *SMV* codes.

According to the discussed and reviewed translator frameworks in model checking approach, the comparison of the related frameworks has illustrated in Table 1. The main factors of this view include existed case study, the modeling method, design method, temporal logic provision, and model checker interaction. All of the translator frameworks added the temporal specifications to the *SMV* code manually. Our presented framework generates all of the temporal logics in forms of the embedded specification rules in *SMV* code. In addition, NuSMV supports two temporal logics to design the specification rules of the system.

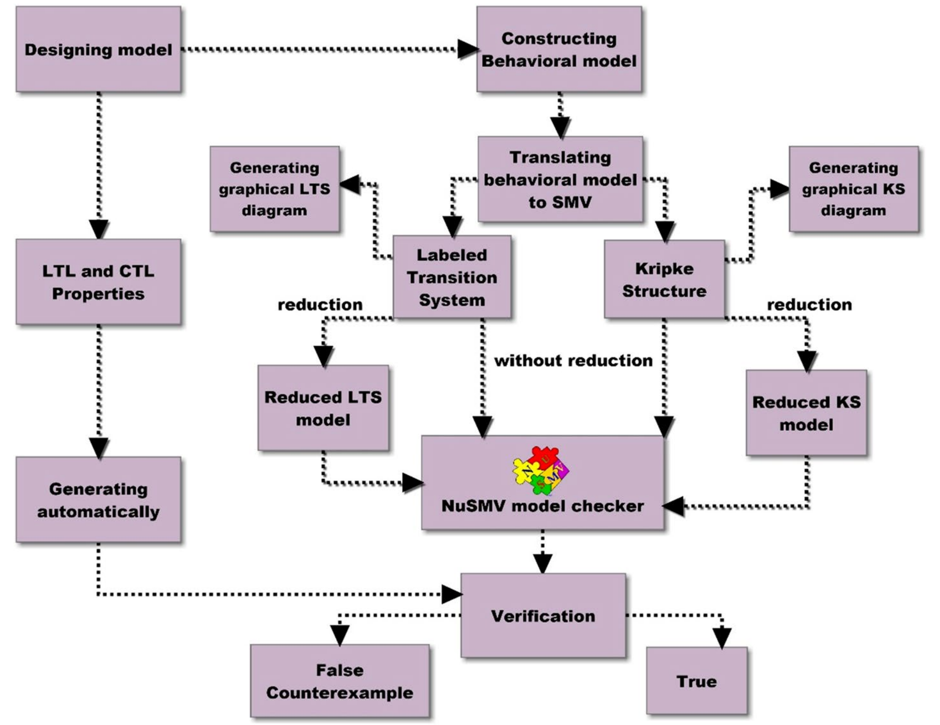


**Fig. 1** The GSMT architecture

To the best of our knowledge, all related frameworks proposed a translator to provide both code generation/execution. Also, editor platforms support just one modeling template such as LTS or KS and one temporal logic formula for the system behavioral model. At complementing with them, our GSMT framework presents (1) automated design approach for formal descriptions of the system, (2) a compositional behavioral modeling for system behavior in forms of LTS and KS models, (3) generating the visual model diagram of the designed behavior, (4) constructing detailed temporal logic formulas in terms of CTL and LTL, and (5) symbolic automated verification approach using NuSMV.

## GSMT framework

This section provides a conceptual description of the proposed framework with some key explanations. The important feature of the GSMT is its flexible modeling and checking capability that represents the common collaboration between two main steps of the formal verification approach. This flexibility is the prominent point of a translator framework that supports all technical features of the behavioral correctness of a complex system. In this section, the framework architecture is explained comprehensively. Also, the presented recursive reduction approach is illustrated in this section.

### GSMT behavioral models

The GSMT navigates the behavioral model to a complete design, actual modeling, and automated translation approach. Figure 1 displays a conceptual architecture of GSMT. The GSMT architecture includes four dependent layers as follow: design, modeling,

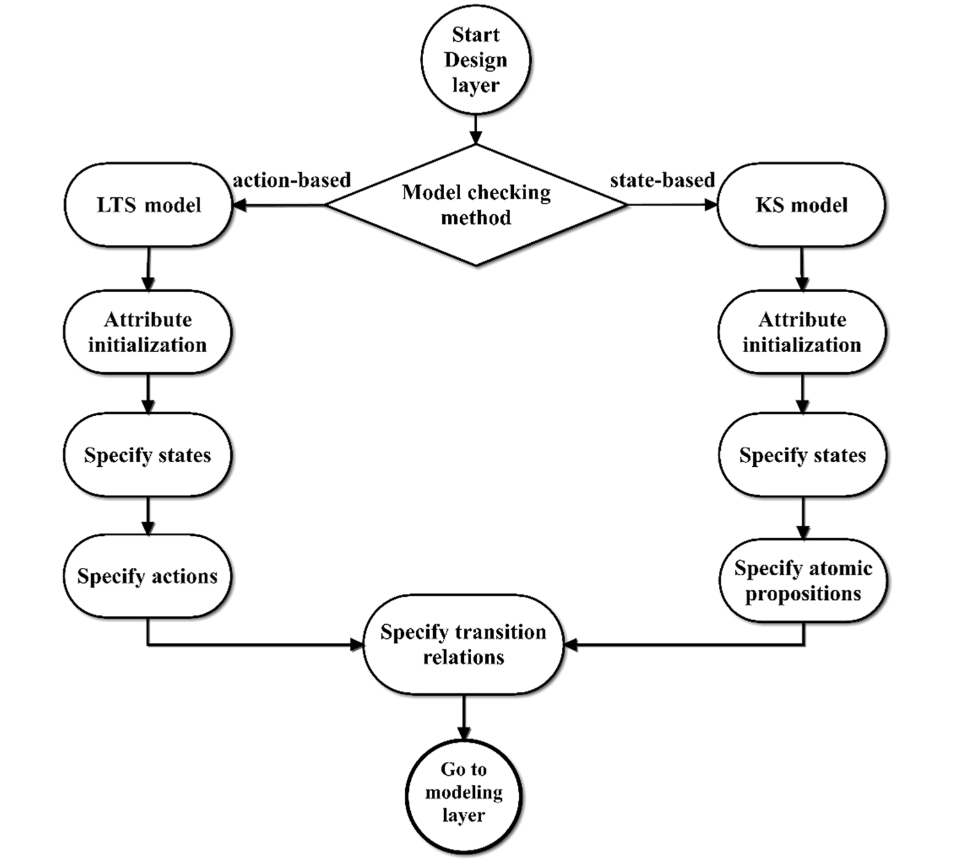


**Fig. 2** The flowchart diagram of the design layer

logic and symbolic code. After designing the proposed model, a behavioral model is constructed by the framework. The behavioral model is translated to an LTS or KS model. The translated model can get two results for converting to the final *SMV* code that includes the original model and reduced model. Concurrently in the logic layer, the specification rules are generated automatically. Then, the final generated code is executed in NuSMV to check the generated specification rules automatically.

Design layer is an interactive level to navigate the fundamental of behavioral model features. This layer has performed following three obligations:

- Specifying design type of the behavioral model in forms of KS or LTS.
- Creating the structural features of the behavioral model such as states, actions, and atomic propositions.
- Creating the system exploration according to the relationship between the features.

Figure 2 illustrates a flowchart diagram that describes the design layer in the GSMT framework. First, the design method is specified for constructing a behavioral model. Depending on the state-based or action-based model checking approaches, two methods can be chosen for this procedure in terms of KS and LTS. When the design method is specified, the basic features of the behavioral model such as states, transitions and actions should be initialized. We address a formal description of the existing methods briefly.

For the KS model, there are some features according to Kripke structure definition [66, 67]. The method is a state-based framework and the states are labeled with a name. The user can input a set of states and atomic propositions for the initialization section.

**Definition 1** A Kripke structure is a five-tuple $KS = (Q, I, P, R, L)$ where [68]:

- $Q$ is a set of states.
- $I$ is the set of initial states: $I \in Q$.
- $P$ is a set of atomic propositions.
- $R$ is a set of transition relations $R \subseteq Q \times Q$.
- $L$ is a state labeling function $\mathrm{L} : Q \rightarrow 2^{\mathrm{P}}$.

In the above definition, a path can be defined on the behavioral model as follow:

**Definition 2** A Kripke path ***KP*** is a finite sequence of the states and transitions starting from the state $\mathrm{q}_1$ and finishing at the state $q_n$ that $(q_1 \text{ and } q_n \in Q, \quad \mathrm{p} \in P)$ denoted as [69]:

$$\boldsymbol{KP} = q_1(p_1) \rightarrow q_2(p_2) \rightarrow q_3(p_3) \ldots q_{n-1}(p_{n-1}) \rightarrow q_n(p_n)$$
$$\text{such that } \forall(\mathrm{i},\mathrm{j}): (q_i, p_i) \in L \text{ and } (q_\mathrm{i}, q_\mathrm{j}) \in R.$$

In the next method, the model is constructed as an LTS model that is the event-based framework and the transitions are labeled with a name [70, 71]. The user can initialize a set of states and actions to design the behavioral model.

**Definition 3** A Labeled Transition System $LT$ is a 4-tuple $LT = (S, M, A, T)$ where:

- $S$ is a set of states.
- $M$ is the set of initial state: $M \in S$.
- $A$ is a set of actions.
- $T$ is a total transition relation: $T \subseteq S \times A \times S$.

This means, the relation $s_1 \overset{a}{\rightarrow} s_2 (s_1,\ s_2 \in S \text{ and } \mathrm{a} \in A)$ is used for stating that $(s_1,\ \mathrm{a},\ s_2) \in T$.

Also, in the second method, a path on the behavioral model is described as follow:

**Definition 4** A Labeled path ***LP*** in the second method is a finite sequence of the events and actions starting from the state $\mathrm{s}_1$ and finishing at the state $s_n (s_1 \text{ and } s_n \in S)$ denoted as [72]:

$$\boldsymbol{LP} = s_1 \overset{a1}{\rightarrow} s_2 \overset{a2}{\rightarrow} s_3 \ldots s_{n-1} \overset{an-1}{\rightarrow} s_n \quad \text{such that } \forall(\mathrm{k},\mathrm{v}) : (s_k,\ a_v,\ s_{k+1}) \in T.$$

By using the Kripke Path ***KP*** and the Labeled Path ***LP***, we create the state space exploration for the proposed behavioral models in model checking.

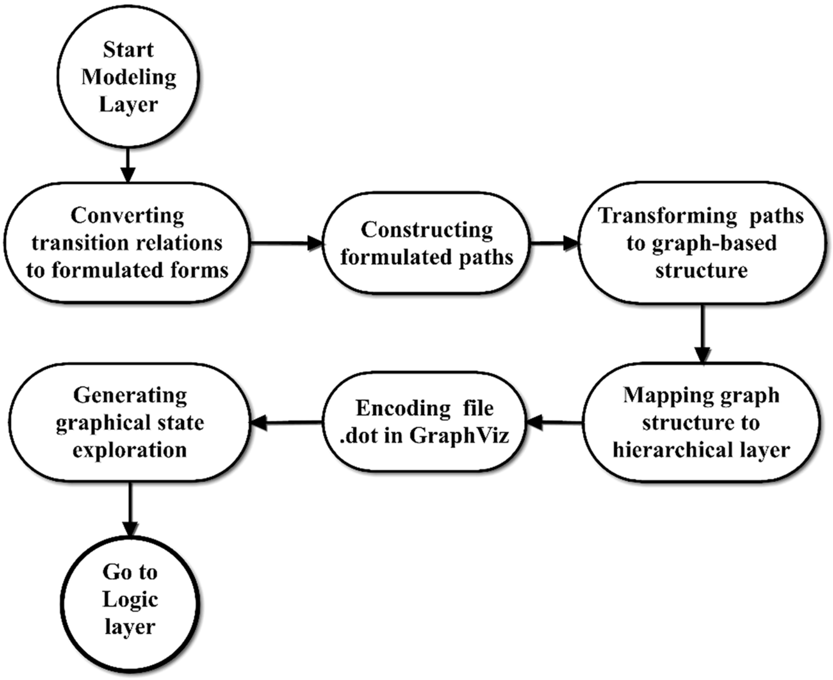

Start Modeling Layer
Converting transition relations to formulated forms
Constructing formulated paths
Transforming paths to graph-based structure
Mapping graph structure to hierarchical layer
Encoding file .dot in GraphViz
Generating graphical state exploration
Go to Logic layer

**Fig. 3** The modeling layer architecture of the GSMT framework

Modeling layer is a visual interaction level illustrating the graphical models of the behavioral model. This layer is classified to the following three steps:

- Configuring the transition relations between the expected attributes in the behavioral model.
- Translating the configurable relations to the graph-based relation machine.
- Generating a graphical state exploration diagram according to the graph-based relation machine.

Figure 3 shows the modeling layer architecture for generating a visual state exploration diagram in the GSMT framework. First, each transition relation in the behavioral model is constructed according to the formulated paths in the above definitions. Due to the importance of the relation handling between expected states, transmitting the states by each event or an atomic proposition is done automatically. In this situation, any transition relation is not omitted in a complex behavioral model. After configuring the formal transition relations, a graph-based relation machine is translated for mapping on the state space exploration. This translation is based on the GraphViz[1] tool as a visual modeling software. Finally, a graphical state exploration diagram for the designed behavioral model has generated automatically. The generated output model is produced in form of *dot* format that has a hierarchical drawing architecture for modeling the system behavior. We prepare the editable version of the modeling format for the user that can save it to the other viewable formats like an image. Due to having the simple language structure in GraphViz, this platform is chosen for increasing the flexibility.

[1] http://www.graphviz.org/.

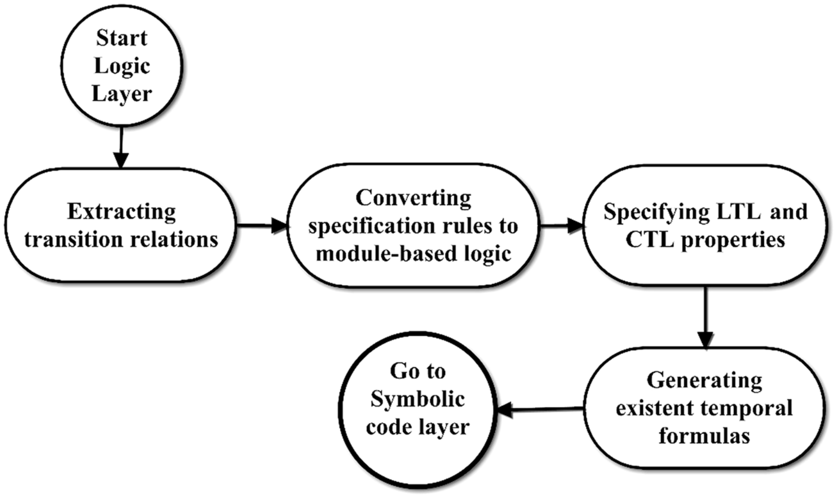

Start Logic Layer
Extracting transition relations
Converting specification rules to module-based logic
Specifying LTL and CTL properties
Generating existent temporal formulas
Go to Symbolic code layer

**Fig. 4** The logic layer architecture of the GSMT framework

The logic layer is a formal descriptive level to demonstrate the temporal logic formulas in verification of the behavioral model. This layer has the following features:

- Extracting the transition relations as a set of specification rules.
- Converting the specification rules to a formula-based platform in forms of the LTL and CTL.
- Generating the existent permutation temporal formulas for all of the specification rules.

Figure 4 shows the logic layer architecture to the automated construction of the temporal logic specifications in terms of reachability, deadlock, liveness, and safety conditions. Initially, the set of states, atomic propositions and actions are extracted to the permutation of the transition relations in a Finite State Machine (FSM). According to the following descriptions of the temporal logics, the conversion procedure is done for each property checking which includes deadlock condition, reachability asset, safeness property and liveness condition. For showing the specification properties, we explain CTL and LTL briefly.

The CTL syntax is described as follows [16]:

$$\alpha ::= \mathrm{True} | p | \neg\alpha | \alpha \wedge \alpha' | \alpha \vee \alpha' | AX\alpha(p) | AG\alpha(p) | AF\alpha(p) | EG\alpha(p) | EX\alpha(p) | EF\alpha(p)$$

- True is a true proposition.
- The $p$ is an atomic proposition where the $\alpha$ formula can hold atomic proposition $p$ with a sentence or statement according to following syntax $\alpha$ ($p$) which is both true or false value.
- The $\alpha$ is ranged over CTL formulas.
- The $\neg\alpha$ (not), $\alpha \wedge \alpha'$ (and) and $\alpha \vee \alpha'$ (or) are logical syntaxes on the formulas.
- $A$ (always) and $E$ (eventually) are the general quantifiers on all of the paths.

- $G$ (globally), $X$ (next state) and $F$ (in the future) are contracted in the entire of each path.

Also, LTL syntax is explained as follow [73, 74]:

$$\beta ::= \text{True}|q|\neg\beta|\beta \wedge \beta'|\beta \vee \beta'|G\beta(q)|F\beta(q)|X\beta(q)|\beta U\beta'$$

- True is a true proposition.
- The $q$ is an atomic proposition where a $\beta$ formula gets atomic proposition $q$ with a declarative statement according to following syntax $\beta$ $(q)$ which is both true or false value.
- The $\beta$ is a range over LTL formulas.
- The $\neg\beta$ (not), $\beta \wedge \beta'$ (and) and $\beta \vee \beta'$ (or) are logical syntaxes on the formulas.
- $G$ (globally), $X$ (next state) and $F$ (in the future) are contracted in the entire of each path.
- The $\beta U\beta'$ means that $\beta$ is true and enabled until $\beta'$ is activated.

According to the specified temporal syntaxes, three categorizations are performed to generate all of the expected specification rules in the system behavior automatically. The user can select each property according to the model analysis. The generated temporal properties are added to the end of the code. For example, we have the simple template of some specification properties for the LTS model as follows:

- (Deadlock freedom) AG !(*state* & *action*);
- (Liveness) AG (*state* & *action*) → AF (*state* & *action*);
- (Reachability) AG (EF(*state* & *action* → *state* & *action*));

Symbolic code layer is a fully automated verification approach for executing the generated symbolic codes in the NuSMV interactive model checker. This layer navigates the following tasks:

- Translating the expected attributes to the hierarchically structured programming platform.
- Transforming the hierarchically structured platform to the *SMV* codes.
- Adding the generated specification formulas to the end of the code.
- Reducing the expected attributes to ameliorate the state space explosion.
- Confirming the reduced behavioral model as the optimally generated *SMV* code.
- Generating the executable *SMV* code for automated verification in NuSMV.

Figure 5 displays the symbolic code layer architecture to automated verification of the behavioral model. First, the modeled structure is translated to a hierarchical-based platform to preserve the expected transition relations. Then, the hierarchical-based platform is transformed into the *SMV* code configuration. In this position, the user has two methods for producing final code. The original *SMV* code of the behavioral model via the expected specification rules are generated automatically. Also, the user can request the reduced behavioral model to ameliorate the state space explosion in a complex system. The GSMT generates a reduced *SMV* code for executing in the NuSMV. In the

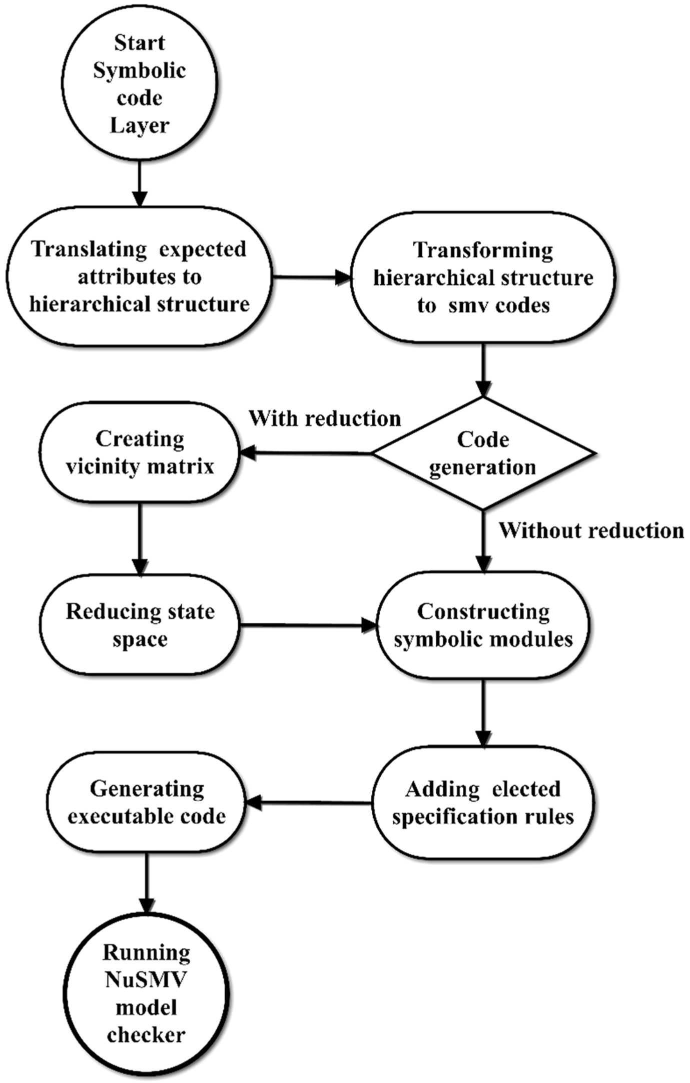

Start Symbolic code Layer
Translating expected attributes to hierarchical structure
Transforming hierarchical structure to smv codes
Code generation
With reduction
Without reduction
Creating vicinity matrix
Reducing state space
Constructing symbolic modules
Adding elected specification rules
Generating executable code
Running NuSMV model checker

**Fig. 5** The symbolic code layer architecture of the GSMT framework

verification phase, the NuSMV reads the generated code and transforms it into a flat hierarchical model. Then, the existing variables are encoded for constructing Ordered Binary Decision Diagram (OBDD) [75] platform. Finally, the constructed model is built for checking the behavioral correctness of the system.

After describing the GSMT architecture, we present the recursive reduction approach for the GSMT.

## Recursive reduced model

The reduced model generally is based on a linear reduction in some related approaches [70, 76–78]. The complex systems have a set of impermissible states that are composed of the parallel relational processes. The similarity of the attributes and transition relations increase the number of state space size. Whatever the number of states and transitions are decreased, the state space is compacted because the size of

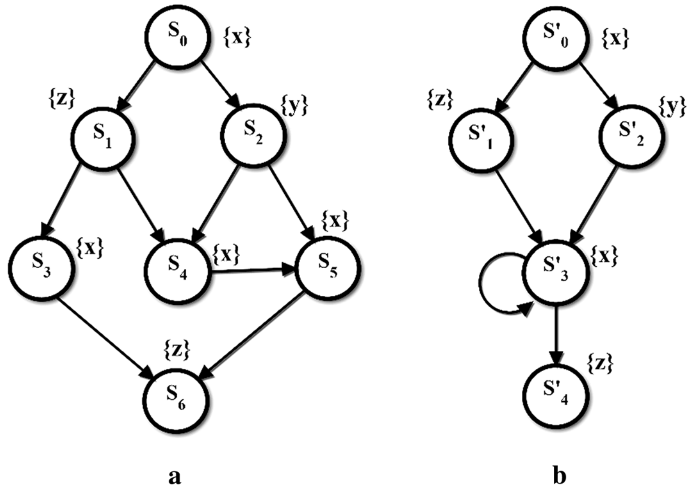


**Fig. 6** **a** The original model. **b** The reduced model of the GSMT reduction algorithm

the state space has been increased exponentially. We use a vicinity matrix for recursive reduced model. To describe the state space reduction, the first step is ordering the vicinity matrix of the state space according to the transition relations. After generating the vicinity matrix, a recursive reduced algorithm is executed for refining the state space. According to the reduction algorithms [76, 77], we have a minimization equivalence method that the model size is defined for comparing the minimality and reduced model [76].

**Definition 5** Model size is shown by $|S_m|$ with the number of states and transitions. In other words, we conclude $|\mathrm{S_m}| \leq \left|\mathrm{S'_m}\right|$ if and only if the number of all attributes (states and transitions) of the $\mathrm{S_m}$ is smaller than $\mathrm{S'_m}$ [79].

**Definition 6** The $\mathrm{S_m}$ is an original model and $\mathrm{S_R}$ is its reduced model. A minimal equivalence $M_e$ is an equal relation, when $|\mathrm{S_R}| \leq |\mathrm{S_m}|$ (the size of the reduced model is smaller than the original model), then $\mathrm{S_m} \equiv \mathrm{S_R}$ (the original model is equivalence with reduced model) if and only if the minimal equivalence $\mathrm{S_R} \approx M_e \approx \mathrm{S_m}$ is established. Consequently, the reduced model $\mathrm{S_R}$ is replaced on the original model $\mathrm{S_m}$ for ameliorating the state space explosion [67].

Figure 6a is the original KS model by a set of states ($S_0$, $S_1$, $S_2$, $S_3$, $S_4$, $S_5$, $S_6$) and Fig. 6b is a reduced KS model. In the original Kripke model, there are three states $S_3$, $S_4$ and $S_5$ by same atomic proposition $\{x\}$ in the KS model that are merged together in set of labeling functions (($S_0$, $\{x\}$), ($S_1$, $\{z\}$), ($S_2$, $\{y\}$), ($S_3$, $S_4$, $S_5$, $\{x\}$), ($S_6$, $\{z\}$)). First, a vicinity matrix is created for the original Kripke model.

Figure 7 depicts the design of the vicinity matrix for the original and reduced models. For a sample, in the original matrix (Fig. 7a), there are two neighborhood values according to the transition relation method. When the value of $\mathrm{S_1S_3}$ is equal to the value of $\mathrm{S_1S_4}$ ($P\mathrm{S_{i,\,j}} = P\mathrm{S_{i,\,j+1}}$) that means there is a same proposition for the proposed states, then the reduced approach is applied. Initially, the $S_4$ and $S_5$ are transmitted to

$$\begin{array}{c|ccccccc|}
 & S_0 & S_1 & S_2 & S_3 & S_4 & S_5 & S_6 \\
S_0 & 0 & z & y & 0 & 0 & 0 & 0 \\
S_1 & 0 & 0 & 0 & \boxed{x} & \boxed{x} & 0 & 0 \\
S_2 & 0 & 0 & 0 & 0 & \boxed{x} & \boxed{x} & 0 \\
S_3 & 0 & 0 & 0 & 0 & 0 & 0 & z \\
S_4 & 0 & 0 & 0 & 0 & 0 & x & 0 \\
S_5 & 0 & 0 & 0 & 0 & 0 & 0 & z \\
S_6 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}
\qquad
\begin{array}{c|ccccc|}
 & S'_0 & S'_1 & S'_2 & S'_3 & S'_4 \\
S'_0 & 0 & z & y & 0 & 0 \\
S'_1 & 0 & 0 & 0 & x & 0 \\
S'_2 & 0 & 0 & 0 & x & 0 \\
S'_3 & 0 & 0 & 0 & x & z \\
S'_4 & 0 & 0 & 0 & 0 & 0
\end{array}$$

**a** **b**

**Fig. 7** The vicinity matrix of **a** original Kripke model, **b** reduced Kripke model

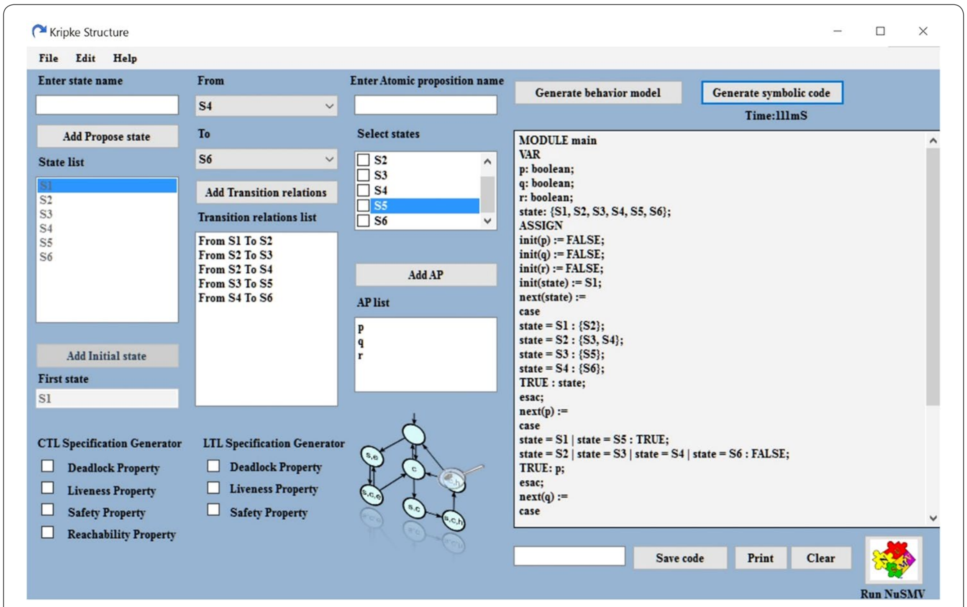
```
For all of the matrix n-arrays (i=1; i =<n ; i++);
{
  For the matrix n-arrays (j=1; j=<n ; j++);
    {
      If ($S_{ij}$=$S_{i(j+1)}$) then
      Delete array $S_{i\,(j+1)}$
      Each relation (x, $S_{j+1}$) transmitted to (x, $S_j$) where x is the existing atomic
proposition of $S_{j+1}$.
      Each relation ($S_{j+1}$, y) transmitted to ($S_j$, y) where y is the existing atomic
proposition of $S_{j+1}$.
      The $S_j$ replaced to deleted $S_{j+1}$
       If ($S_{ij}$=$S_{ji}$)
       Delete array $S_{ji}$
       Each relation ($S_j$, x) transmitted to ($S_j$, x)
       Each relation (y, $S_j$) transmitted to (y, $S_i$)
       The $S_{ii}$ replaced to deleted $S_j$
     }
}
```

**Fig. 8** The recursive reduction algorithm of the GSMT

the $S'_3$ and the proposition $\{x\}$ is omitted for them. Second, each inputted edge to the $S_4$ and $S_5$ is inputted to the $S'_3$ and each outputted edge from the $S_4$ and $S_5$ is outputted from the $S'_3$. Then, the remaining Kripke model is mapped to the new Kripke model as a reduced model (Fig. 7b) by set of states ($S'_0$, $S'_1$, $S'_2$, $S'_3$, $S'_4$) and set of labeling functions (($S'_0$, $\{x\}$), ($S'_1$, $\{z\}$), ($S'_2$, $\{y\}$), ($S'_3$, $\{x\}$), ($S'_4$, $\{z\}$)). The number of two states and three edges are deleted from the original Kripke model. Finally, the relation of minimal equivalence between $\mathrm{K}_{\mathrm{Original}}$ and $\mathrm{K}'_{\mathrm{Reduced}}$ is established as follows:

The size of the reduced Kripke model is lower than original Kripke model $|\ \mathrm{K}'_{\mathrm{Reduced}}\ | \leq |\ \mathrm{K}_{\mathrm{Original}}\ |$ and the original Kripke model is equivalence with reduced Kripke model $\mathrm{K}_{\mathrm{Original}} \equiv \mathrm{K}'_{\mathrm{Reduced}}$.

Figure 8 depicts the recursively reduced algorithm based on the vicinity matrix of labeling functions. This algorithm provides two conditions for the reduced model for



State list: S1, S2, S3, S4, S5, S6

First state: S1

From: S4

To: S6

Transition relations list:

From S1 To S2
From S2 To S3
From S2 To S4
From S3 To S5
From S4 To S6

Select states: □ S2, □ S3, □ S4, □ S5, □ S6

AP list: p, q, r

```
MODULE main
VAR
p: boolean;
q: boolean;
r: boolean;
state: {S1, S2, S3, S4, S5, S6};
ASSIGN
init(p) := FALSE;
init(q) := FALSE;
init(r) := FALSE;
init(state) := S1;
next(state) :=
case
state = S1 : {S2};
state = S2 : {S3, S4};
state = S3 : {S5};
state = S4 : {S6};
TRUE : state;
esac;
next(p) :=
case
state = S1 | state = S5 : TRUE;
state = S2 | state = S3 | state = S4 | state = S6 : FALSE;
TRUE: p;
esac;
next(q) :=
case
```

**Fig. 9** The KS window in the GSMT framework

searching each matrix array. First, both neighbor values by a vicinity condition are specified if $S(i,\ j) = S(i,\ j+1)$, then the reduction procedure is applied. Second, each loop condition occurs for two states if $S(i, j) = S(j,\ i)$, then the reduction procedure is applied. Searching matrix arrays are done until there is no array that applies in two conditions.

## Experimental analysis

This section illustrates some experimental case studies to evaluate the GSMT framework. First, a brief exploration of the GSMT environment is presented. Then, some case studies are illustrated to demonstrate the performance evaluation of the framework. Finally, the verification results are shown in this section.

### User interface of GSMT

The framework consists of three main windows that include modeling method selection, KS model window, and LTS model.

In Fig. 9, a Kripke model platform is shown for creating Example 1 as a case study. Following sections illustrate the important regions in KS platform. At the first stage, the designer can input initial information for the behavioral model. The reduction method, generating temporal logics and generating *SMV* codes are done automatically. In the main text, all the existing layers have been illustrated with manual or automatic conditions.

- Add propose state: the user inputs a set of existing states on *Add propose state* button. The defined states are listed in the *state list* manually.
- Add initial state: the user should select an initial state from the state list which the initial state is displayed in the *First state* box manually.

- Add transition relation: it shows the transition relations that are constructed in forms of *From/To* structure. All of the transition relations are listed in *transition relation list* box. The interaction simplicity is a key point for users and engineers to design and model a complex system using GSMT manually.
- Add AP: it specifies the atomic propositions of each state using *Add AP* button manually.
- Generate behavioral model: it consists of a button which generates a graphical state transition diagram in form of the GraphViz output based on own structure codes automatically.
- Generate symbolic code: it is a symbolic code generation for constructing the final *SMV* code in the following textbox. This textbox is an editable platform for copying and modifying the *SMV* code. When the checkbox *reduce* is checked, then the model is reduced according to the reduction approach and the reduced final *SMV* code is generated. Also, the framework generates the new graphical diagram for reduced model automatically.
- Specification generators: by selecting the specification rules, the GSMT produces the temporal formulas automatically. In the column of CTL specification generator, there are 4 specification rules for adding to the end of the *SMV* code automatically.
- For example, the deadlock and reachability properties are selected to generate and add in the code. In addition, the LTL specification generator column has three specification rules. In Fig. 9, all of the properties are selected to add the end of the code.

*Example 1* This example illustrates a translation procedure for a Kripke model to the *SMV* code. A verification approach is done based on the NuSMV model checker automatically. According to Definition 1, the formal description of the Kripke structure of Example 1 is as follow:

- Set of the states $Q = (S1, S2, S3, S4, S5, S6)$.
- The initial state $I = S1$.
- The set of atomic propositions $P = (p, q, r)$.
- The set of transition relations $R = \{(S1, S2), (S2, S3), (S2, S4), (S3, S5), (S4, S6)\}$.
- The state labelling functions $L = ((S1, \{p\}), (S2, \{q\}), (S3, \{q\}), (S4, \{p, q\}), (S5, \{p, r\}), (S6, \{p, r\})$.

Figure 10 shows the graphical state transition diagram of Example 1 that is generated automatically using GSMT. After modeling the proposed behavioral model of the Example 1, the final *SMV* code is generated according to the symbolic code platform. The verification results of the Example 1 are as follows:

- The execution time of this model is 158.5 ms,
- Generating 18 deadlock-free properties,
- Generating 180 reachability properties,
- Generating 180 liveness properties,
- Generating 720 safety properties.

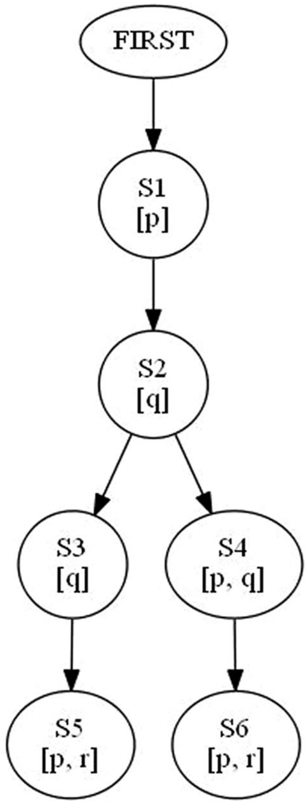
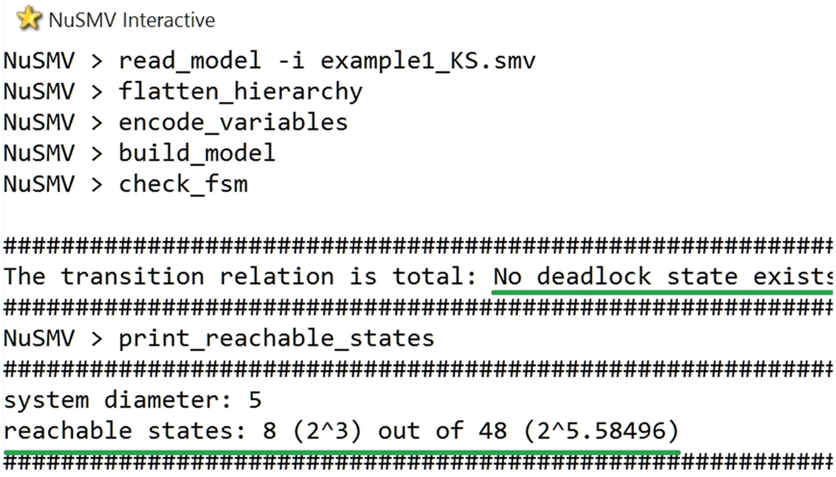


**Fig. 10** The generated graphical transition diagram of Example 1 in the GSMT framework



```
NuSMV > read_model -i example1_KS.smv
NuSMV > flatten_hierarchy
NuSMV > encode_variables
NuSMV > build_model
NuSMV > check_fsm

##########################################################
The transition relation is total: No deadlock state exists
##########################################################
NuSMV > print_reachable_states
##########################################################
system diameter: 5
reachable states: 8 (2^3) out of 48 (2^5.58496)
##########################################################
```

**Fig. 11** The automated model checking environment of the Example 1

Figure 11 illustrates the executed *SMV* code in NuSMV for Example 1 automatically. In this figure, there is no deadlock problem in the example. The existing reachable states of the proposed model is 64 with system diameter 5. The numbers of allocated OBDD states are 297. After checking the CTL specifications, the 50% of the generated

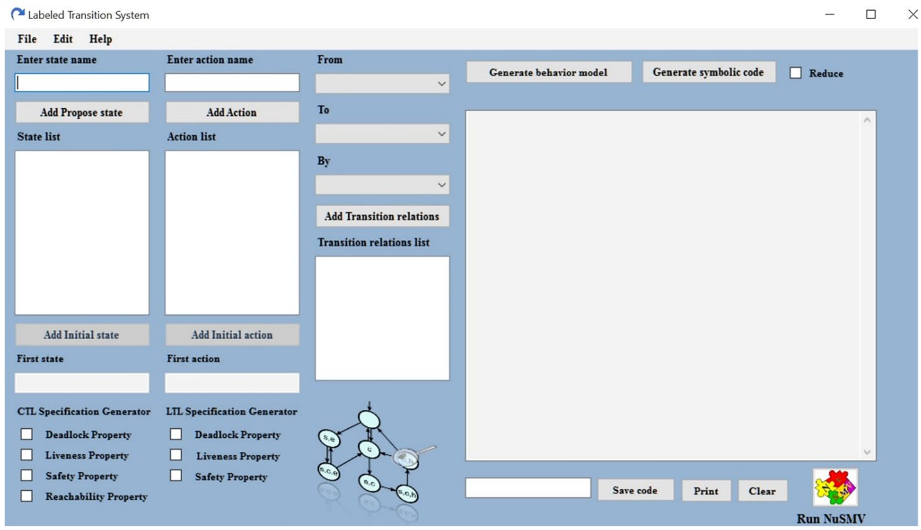


**Fig. 12** The LTS window in the GSMT framework

deadlock properties is true, the 77% of the reachability properties is true, the 100% of the liveness properties is true, and the 92% of the safety properties is true. The total number of the generated CTL properties of the Example 1 is 1098.

Figure 12 shows the LTS window for constructing the Example 2 as the suggested case study. According to the specified numbers of Fig. 12, the translation procedure of the LTS to the final *SMV* code is shown.

- Add propose state: the user inputs a set of existing states on *Add propose state* button. The defined states are listed in the *state list* manually.
- Add initial state: the user should select an initial state from the state list which the initial state is displayed in the *First state* box manually.
- Add action: the user inputs a set of existing actions on *Add propose action* button manually.
- Add initial action: the user selects an initial action from the action list which the initial action is shown in the *First action* box manually.
- Add transition relation: it shows the transition relations that are constructed in forms of *From*/*To*/*By* structure. All of the transition relations are listed in *transition relation list* box manually.
- Generate a behavioral model: it consists of a button which generates a graphical state transition diagram in form of the GraphViz output automatically.
- Generate symbolic code: it is a symbolic code generation for constructing the final *SMV* code in the following textbox. This textbox is an editable platform for copying and modifying the *SMV* code. When the checkbox *reduce* is checked, then the model is reduced according to the reduction approach and the reduced final *SMV* code is generated. Also, the framework generates the new graphical diagram for reduced model automatically.

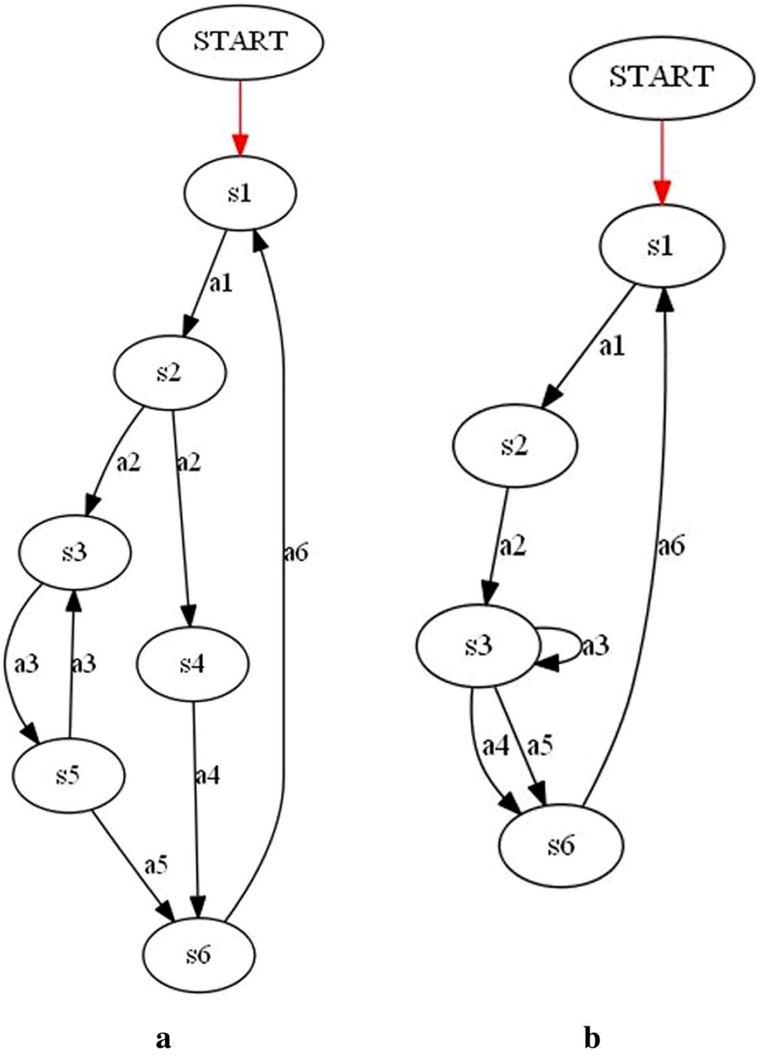
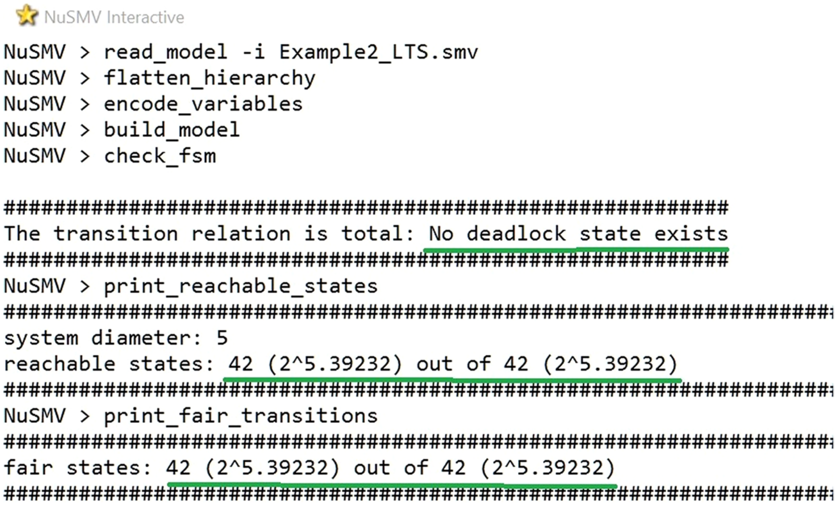


**Fig. 13** The generated graphical transition diagram of Example 2 in **a** original and **b** reduced model

- Specification generators: by selecting the specification rules, the GSMT produces the temporal formulas automatically. In the column of CTL specification generator, there are 4 specification rules for adding to the end of the *SMV* code automatically.

*Example 2* This example illustrates a translation procedure for a labeled model to the *SMV* code. A modeling and verification approach is done based on the NuSMV model checker automatically. According to Definition 3, the formal description of the LTS of Example 2 is as follow:

- Set of the state $S = (\mathrm{S1, S2, S3, S4, S5, S6})$.
- The initial state $M = \mathrm{S1}$.
- The set of atomic propositoins $A = (\mathrm{a1, a2, a3, a4, a5, a6})$.



```
NuSMV > read_model -i Example2_LTS.smv
NuSMV > flatten_hierarchy
NuSMV > encode_variables
NuSMV > build_model
NuSMV > check_fsm

##########################################################
The transition relation is total: No deadlock state exists
##########################################################
NuSMV > print_reachable_states
##################################################################
system diameter: 5
reachable states: 42 (2^5.39232) out of 42 (2^5.39232)
##################################################################
NuSMV > print_fair_transitions
##################################################################
fair states: 42 (2^5.39232) out of 42 (2^5.39232)
##################################################################
```

**Fig. 14** The automated model checking environment of the Example 2

- The set of transition relations $T$ = {(S1, a1, S2), (S2, a2, S3), (S2, a2, S4), (S3, a3, S5), (S4, a4, S6), (S5, a5, S6), (S5, a3, S3), (S6, a6, S1)}.

Figure 13 shows the graphical state transition diagrams for Example 2 that is generated automatically using GSMT. Figure 13a shows the original LTS model and Fig. 13b depicts the reduced LTS model after applying the reduce approach on the original model. After modeling the proposed behavioral model of the Example 2, the final *SMV* code is generated according to the symbolic code platform. The verification results of the Example 2 are as follows:

- The execution time of this model is 328.9 ms;
- Generating 42 deadlock-free properties;
- Generating 1260 reachability properties;
- Generating 1260 liveness properties;
- Generating 3450 safety properties.

Figure 14 illustrates the executed *SMV* code of Example 2 in the NuSMV automatically. In this figure, there is no deadlock problem. The existing reachable states of the proposed model is 2680 with system diameter 5. The numbers of allocated OBDD states are 296. After checking the CTL specifications, the 55% of the generated deadlock properties is true, the 75% of the reachability properties is true, the 100% of the liveness properties is true, and the 94% of the safety properties is true. The total number of the generated CTL properties of the Example 2 is 6012.

## Performance evaluation

For comparing the performance of GSMT and the other translator frameworks, some test case examples are analyzed. In this experiment, an Intel® Core™ i5-6200U @ 2.30 GHz CPU, and 8 GB memory in Windows 10 have been used.

**Table 2 Test cases of the GSMT analysis**

| Test case | No. states | No. Trans |
|---|---|---|
| Case 1 | 10 | 8 |
| Case 2 | 50 | 70 |
| Case 3 | 100 | 150 |
| Case 4 | 500 | 650 |
| Case 5 | 1000 | 1250 |
| Case 6 | 5000 | 5700 |
| Case 7 | 10,000 | 13,500 |
| Case 8 | 20,000 | 27,000 |
| Case 9 | 50,000 | 65,000 |
| Case 10 | 100,000 | 155,000 |

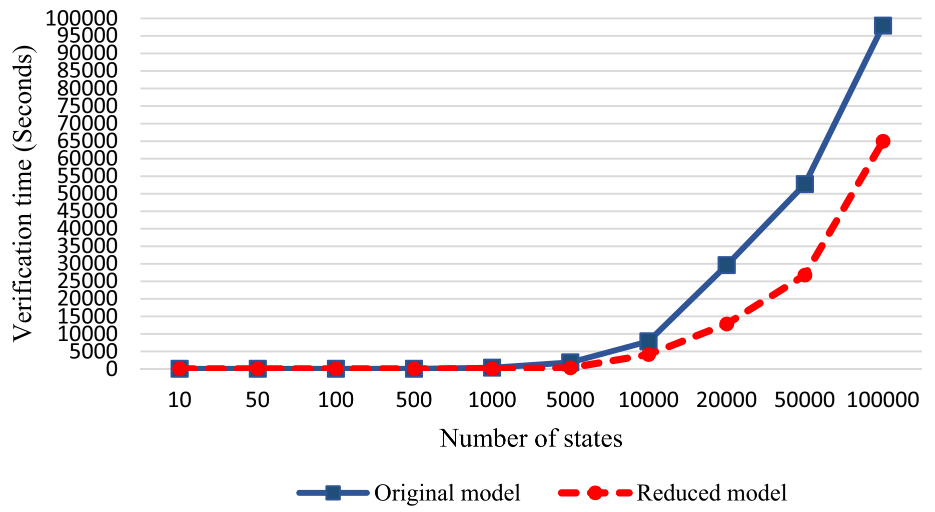


**Fig. 15** Verification time of test cases verification in original and reduced models

The first level of the performance evaluation is analyzing the verification time of the original and reduced models that we perform some test cases to analyze the GSMT framework. The details of these test cases are illustrated in Table 2. These case studies are generated randomly.

Figure 15 demonstrates the verification time for ten test cases (10 to 100,000 state explorations) in forms of original and reduced models. This result specifies that the reduced model of the GSMT provides a substantial performance in the verification time. When the number of the state space attributes are increased, the verification time of the state exploration is grown. In this situation, the reduced model can significantly decrease the verification time of system verification.

Also, Table 3 shows the number of states and transitions of 10 test cases of Table 2 in order to the percentage of state space reduction using the proposed recursive reduced model of the GSMT framework. The reduction average of the state space using GSMT is 18.54%. The reduced models have minimal equivalency relations with the original models.

**Table 3 Comparison of the state space reduction for the test cases in GSMT**

| Case study | Model | | | | State space reduction % |
|---|---|---|---|---|---|
| | Original model | | Reduced model | | |
| | No. states | No. Trans | No. states | No. Trans | |
| Case 1 | 10 | 8 | 8 | 13 | 17.65 |
| Case 2 | 50 | 70 | 41 | 57 | 18.3 |
| Case 3 | 100 | 150 | 77 | 120 | 23 |
| Case 4 | 500 | 650 | 410 | 554 | 18.5 |
| Case 5 | 1000 | 1250 | 832 | 1021 | 18.25 |
| Case 6 | 5000 | 5700 | 4078 | 5287 | 18.1 |
| Case 7 | 10,000 | 13,500 | 8268 | 11,752 | 18.9 |
| Case 8 | 20,000 | 27,000 | 16,378 | 26,274 | 17.5 |
| Case 9 | 50,000 | 65,000 | 42,680 | 61,723 | 18.2 |
| Case 10 | 100,000 | 155,000 | 84,727 | 148,316 | 18.1 |

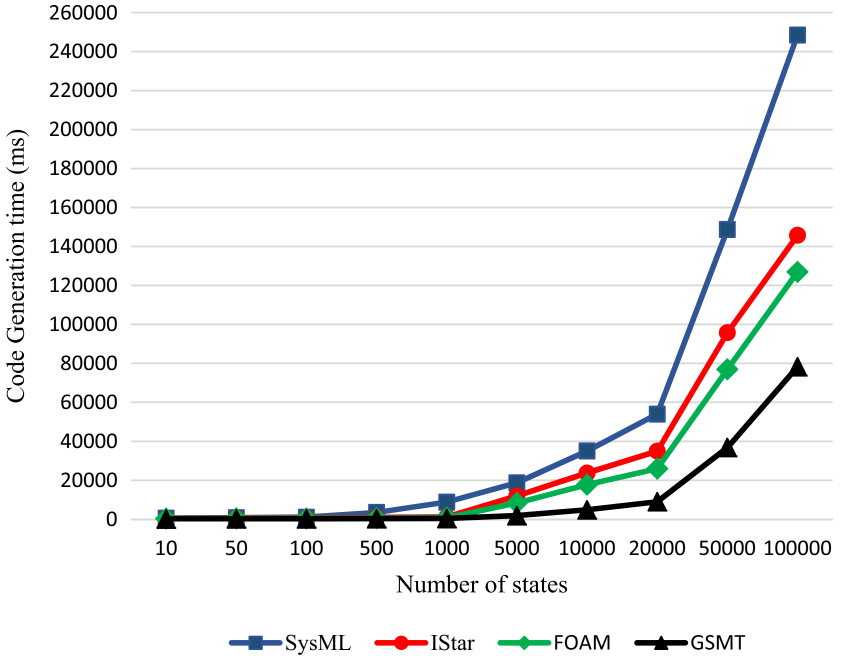


**Fig. 16** Generation time of final code in translator frameworks

The second level of the performance analysis is to compare the code generation time for ten test cases (10 to 100,000 state explorations). We implemented the existing case studies in three famous translator frameworks SysML-ja [58], IStar [59], and FOAM [61] to compare with the performance of the GSMT framework. Since the selected framework supports just the LTS model, the structure of existing examples has been considered in forms of LTS for a fair measurement. Figure 16 depicts the code generation time for specifying the case studies. By increasing the number of the states and transitions in each example, the generation time is grown exponentially. As the result, the GSMT framework generates the final code by minimum time.

## Discussion

Some model checking converters follow up a standard translation architecture such as a design code structure, the specification properties definition, and an executive verifiable code. The proposed framework not only supports the standard platforms but also represents a specification rules generator, behavioral model generation, and space reduction approach. However, interconnecting some verification approaches such as process algebraic methods and theorem proving tools is a key challenge in complex software and hardware development. The experimental results acquired via some individual test cases obviously demonstrate that the recursive reduction method improves significantly the execution and verification time. Nevertheless, the increasing of the generated specification rules can influence code generation complexity and rise to check the time of the properties negatively, and enhancement of the system correctness positively. The limitations of the GSMT framework can be improved with applying the evolutionary algorithms in the model checking approach. For example, the reduction time of the reduced model can significantly be decreased using greedy algorithms. For analyzing the completeness and soundness of a complex system, the model checking approach is time-consuming and the theorem proving frameworks such as Isabelle[2] and SPASSD[3] tools can influence to prove these problems. Also, middleware converters between the concrete and the verifiable model are very useful to correctness evaluation of the complex systems. The important challenge of these converters is the approximation of the verifiable model to the implementation model. Table 4 shows the comparison of the related frameworks and the GSMT according to the verification environment factors in terms of code generation mode, editor layer, graphical modeling creation, property generation section, and reduction approach. In this evaluation, the GSMT support all of the verification environment factors automatically.

## Conclusion and future work

In this research, a GSMT is presented with respect to simplifying the behavioral modeling software systems. It consists of the behavioral modeling in form of the LTS and the KS, generating a graphical state exploration diagram of the behavioral model, generating the expected specification rules automatically, translating the behavioral model to the *SMV* codes, and reduction of the state space. The important functionality of the GSMT is the implementation of the syntactic reduced approach that ameliorates the state space explosion. Also, the framework generates the specification rules for proofing the correctness of the model automatically. In order to use the NuSMV, the GSMT supports both the LTL and CTL formulas to add the final code for execution in the interactive environment. The experimental results of the GSMT shown that this framework has usability and simplicity for behavioral modeling software and hardware systems. In comparison analysis, the reduction approach can significantly decrease the execution time for model verification. In addition, the framework has a sufficient execution time for generating final executable *SMV* code rather than the other translation model checking frameworks. In checking the generated specification properties for each model in average, the

[2] http://isabelle.in.tum.de/.

[3] http://www.spass-prover.org/.

**Table 4 Assessment principles for related frameworks in the verification environment**

| Factor | Tool | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | GSMT | WaVer [56] | FBD [54] | PLC [57] | SysML-ja [58] | IStar [59] | SimuLink [60] | FOAM [61] | ArchiMate [63] | S2N [64] | Petrinet2smv [65] |
| Code generation | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Editor layer | ✓ | ✓ | ✓ | – | ✓ | ✓ | ✓ | – | ✓ | – | ✓ |
| Graphical model generation | ✓ | – | ✓ | – | – | – | – | – | ✓ | – | – |
| Property generation | ✓ | – | – | – | – | – | – | – | – | – | – |
| Reduction approach | ✓ | – | – | – | – | – | – | – | – | – | – |

55% of the generated deadlock properties is true, the 73.5% of the reachability properties is true, the 100% of the liveness properties is true, and the 93% of the safety properties is true. In the future work, we will add some key features such as contracting the formal specification using pi-calculus and model checking in an integrated framework, improving the specification rules generation according to behavioral model satisfactory, refining the state space reduction percentage for the complex systems, and applying the multi-action transition associations for decreasing the state space complexity.

**Authors' contributions**

All authors read and approved the final manuscript.

**Author details**

[1] Department of Computer Engineering, Science and Research Branch, Islamic Azad University, Tehran, Iran. [2] Department of Computer Engineering, Tabriz Branch, Islamic Azad University, Tabriz, Iran. [3] Department of Computer Engineering, Saveh Branch, Islamic Azad University, Saveh, Iran.

**Acknowledgements**

Not applicable.

**Competing interests**

The authors declare that they have no competing interests.

**Availability of data and materials**

Not applicable.

**Funding**

No funding was received.

## Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Received: 27 September 2018 Accepted: 5 January 2019
Published online: 28 January 2019

**References**

1. Mitsch S, Passmore GO, Platzer A (2014) Collaborative verification-driven engineering of hybrid systems. Math Comput Sci 8:71–97
2. Li Y, Tao F, Cheng Y, Zhang X, Nee AYC (2017) Complex networks in advanced manufacturing systems. J Manuf Syst 43:409–421
3. Vakili A, Navimipour NJ (2017) Comprehensive and systematic review of the service composition mechanisms in the cloud environments. J Netw Comput Appl 81:24–36
4. Keshanchi B, Souri A, Navimipour NJ (2017) An improved genetic algorithm for task scheduling in the cloud environments using the priority queues: formal verification, simulation, and statistical testing. J Syst Softw 124:1–21
5. Glaßer C, Pavan A, Travers S (2011) The fault tolerance of NP-hard problems. Inf Comput 209:443–455
6. Higashino WA, Capretz MAM, Bittencourt LF (2016) CEPSim: modelling and simulation of complex event processing systems in cloud environments. Future Gener Comput Syst 65:122–139
7. Suh Y-K, Lee KY (2018) A survey of simulation provenance systems: modeling, capturing, querying, visualization, and advanced utilization. Hum Centric Comput Inf Sci 8:27
8. Dill DL (1998) What's between simulation and formal verification? (extended abstract). In: Presented at the proceedings of the 35th annual design automation conference, San Francisco, California, USA
9. Li K, Liu L, Zhai J, Kosgoftaar TM, Shao M, Liu W (2017) Reliability evaluation model of component-based software based on complex network theory. Qual Reliab Eng Int 33(3):543–550
10. Khan W, Ullah H, Ahmad A, Sultan K, Alzahrani AJ, Khan SD et al (2018) CrashSafe: a formal model for proving crash-safety of Android applications. Hum Centric Comput Inf Sci 8:21
11. Kim J, Won Y (2017) Patch integrity verification method using dual electronic signatures. J Inf Process Syst 13
12. Hu K, Lei L, Tsai W-T (2016) Multi-tenant verification-as-a-service (VaaS) in a cloud. Simul Model Pract Theory 60:122–143
13. Jafari Navimipour N (2015) A formal approach for the specification and verification of a Trustworthy Human Resource Discovery mechanism in the Expert Cloud. Expert Syst Appl 42:6112–6131
14. Jafari Navimipour N, Habibizad Navin A, Rahmani AM, Hosseinzadeh M (2015) Behavioral modeling and automated verification of a Cloud-based framework to share the knowledge and skills of human resources. Comput Ind 68:65–77
15. Souri A (2016) Formal specification and verification of a data replication approach in distributed systems. Int J Next Gener Comput 7(1):18–37
16. Souri A, Jafari Navimipour N (2014) Behavioral modeling and formal verification of a resource discovery approach in Grid computing. Expert Syst Appl 41:3831–3849

17. Souri A, Norouzi M, Safarkhanlou A, Sardroud SHEH (2016) A dynamic data replication with consistency approach in data grids: modeling and verification. Balt J Mod Comput 4:546
18. Shen VRL, Wang Y-Y, Yu L-Y (2016) A novel blood pressure verification system for home care. Comput Stand Interfaces 44:42–53
19. Rezaee A, Rahmani AM, Movaghar A, Teshnehlab M (2014) Formal process algebraic modeling, verification, and analysis of an abstract Fuzzy Inference Cloud Service. J Supercomput 67:345–383
20. Ruiz MC, Cazorla D, Pérez D, Conejero J (2016) Formal performance evaluation of the Map/Reduce framework within cloud computing. J Supercomput 72:3136–3155
21. Hermanns H, Herzog U, Katoen J-P (2002) Process algebra for performance evaluation. Theoret Comput Sci 274:43–87
22. Tini S, Larsen KG, Gebler D (2017) Compositional bisimulation metric reasoning with probabilistic process calculi. Log Methods Comput Sci 12(4):2627
23. Chen X, Wang L (2017) Exploring fog computing based adaptive vehicular data scheduling policies through a compositional formal method-PEPA. IEEE Commun Lett. 2017
24. Challenger M, Mernik M, Kardas G, Kosar T (2016) Declarative specifications for the development of multi-agent systems. Comput Stand Interfaces 43:91–115
25. Hao F, Sim D-S, Park D-S, Seo H-S (2017) Similarity evaluation between graphs: a formal concept analysis approach. JIPS 13:1158–1167
26. Sardar MU, Hasan O, Shafique M, Henkel J (2017) Theorem proving based formal verification of distributed dynamic thermal management schemes. J Parallel Distrib Comput 100:157–171
27. Srikanth A, Sahin B, Harris WR (2017) Complexity verification using guided theorem enumeration. In: Proceedings of the 44th ACM SIGPLAN symposium on principles of programming languages, pp 639–652
28. Xue T, Ying S, Wu Q, Jia X, Hu X, Zhai X et al (2017) Verifying integrity of exception handling in service-oriented software. Int J Grid Util Comput 8:7–21
29. Copet PB, Marchetto G, Sisto R, Costa L (2017) Formal verification of LTE-UMTS and LTE–LTE handover procedures. Comput Stand Interfaces 50:92–106
30. Edmund J, Clarke M, Grumberg O, Peled DA (1999) Model checking. MIT Press, Cambridge
31. Leitner-Fischer F, Leue S (2013) Causality checking for complex system models. In: Giacobazzi R, Berdine J, Mastroeni I (eds) Proceedings of verification, model checking, and abstract interpretation: 14th international conference, VMCAI 2013, Rome, Italy, January 20–22, 2013. Springer Berlin Heidelberg, Berlin, pp 248–267
32. Merelli E, Paoletti N, Tesei L (2017) Adaptability checking in complex systems. Sci Comput Program 115–116:23–46
33. Baier C, Katoen J-P (2008) Principles of model checking (representation and mind series). The MIT Press, Cambridge
34. McMillan KL (1993) Symbolic model checking. Kluwer Academic Publishers, Norwell
35. Burch JR, Clarke EM, McMillan KL, Dill DL, Hwang LJ (1992) Symbolic model checking: 1020 states and beyond. Inf Comput 98:142–170
36. Souri A, Norouzi M (2015) A new probable decision making approach for verification of probabilistic real-time systems. In: 2015 6th IEEE international conference on software engineering and service science (ICSESS), pp 44–47
37. Cimatti A, Clarke E, Giunchiglia F, Roveri M (2000) NuSMV: a new symbolic model checker. Int J Softw Tools Technol Transfer 2:410–425
38. Sun J, Liu Y, Dong JS (2008) Model checking CSP revisited: introducing a process analysis toolkit. In: International symposium on leveraging applications of formal methods, verification and validation, pp 307–322
39. Holzmann GJ (1997) The model checker SPIN. IEEE Trans Softw Eng 23:279–295
40. Bengtsson J, Larsen K, Larsson F, Pettersson P, Yi W (1995) UPPAAL—a tool suite for automatic verification of real-time systems. In: International hybrid systems workshop, pp 232–243
41. Podivinsky J, Cekan O, Lojda J, Zachariasova M, Krcma M, Kotasek Z (2017) Functional verification based platform for evaluating fault tolerance properties. Microprocess Microsyst 52:145–159
42. Wang S, Huang K (2016) Improving the efficiency of functional verification based on test prioritization. Microprocess Microsyst 41:1–11
43. Balasubramaniyan S, Srinivasan S, Buonopane F, Subathra B, Vain J, Ramaswamy S (2016) Design and verification of Cyber-Physical Systems using TrueTime, evolutionary optimization and UPPAAL. Microprocess Microsyst 42:37–48
44. Kaufmann P, Kronegger M, Pfandler A, Seidl M, Widl M (2015) Intra- and interdiagram consistency checking of behavioral multiview models. Comput Lang Syst Struct 44(Part A):72–88
45. López-Fernández JJ, Guerra E, de Lara J (2016) Combining unit and specification-based testing for meta-model validation and verification. Inf Syst 62:104–135
46. Amálio N, Glodt C (2015) A tool for visual and formal modelling of software designs. Sci Comput Program 98(Part 1):52–79
47. Holzmann GJ, Joshi R, Groce A (2008) New challenges in model checking. In: Grumberg O, Veith H (eds) 25 years of model checking: history, achievements, perspectives, Springer Berlin Heidelberg, Berlin, pp 65–76
48. Bozzano M, Villafiorita A (2006) The FSAP/NuSMV-SA safety analysis platform. Int J Softw Tools Technol Transfer 9:5
49. Głuchowski P (2016) NuSMV model verification of an airport traffic control system with deontic rules. In: Zamojski W, Mazurkiewicz J, Sugier J, Walkowiak T, Kacprzykj (eds) Dependability engineering and complex systems: proceedings of the eleventh international conference on dependability and complex systems DepCoS-RELCOMEX. June 27–July 1, 2016, Brunów, Poland, Springer International Publishing, Cham, pp 195–206
50. Safarkhanlou A, Souri A, Norouzi M, Sardroud SEH (2015) Formalizing and verification of an antivirus protection service using model checking. Procedia Comput Sci 57:1324–1331
51. Ngo VC, Legay A (2018) Formal verification of probabilistic SystemC models with statistical model checking. J Softw Evol Process 30:e1890
52. Li W, Hayes JH, Antoniol G, Guéhéneuc Y-G, Adams B (2016) Error leakage and wasted time: sensitivity and effort analysis of a requirements consistency checking process. J Softw Evol Process 28:1061–1080
53. Mercorio F (2013) Model checking for universal planning in deterministic and non-deterministic domains. AI Commun 26:257–259

54. Li J, Qeriqi A, Steffen M, Yu IC. Automatic translation from FBD-PLC-programs to NuSMV for model checking safety-critical control systems. 2016
55. Sharma PK, Ryu JH, Park KY, Park JH, Park JH (2018) Li-Fi based on security cloud framework for future IT environment. Hum Centric Comput Inf Sci 8:23
56. Castelluccia D, Mongiello M, Ruta M, Totaro R (2006) WAVer: a model checking-based tool to verify web application design. Electron Notes Theor Comput Sci 157:61–76
57. Abdelsadiq A (2013) A toolkit for model checking of electronic contracts
58. Caltais G, Leitner-Fischer F, Leue S, Weiser J (2016) SysML to NuSMV model transformation via object-orientation
59. Deb N, Chaki N, Ghose A (2016) Extracting finite state models from i* models. J Syst Softw 121:265–280
60. Meenakshi B, Bhatnagar A, Roy S (2006) Tool for translating Simulink models into input language of a model checker
61. Vinárek J, Ŝimko V, Hnětynka P (2015) Verification of use-cases with FOAM tool in context of cloud providers. In: 2015 41st euromicro conference on software engineering and advanced applications, pp 151–158
62. Simko V, Hauzar D, Hnetynka P, Bures T, Plasil F (2015) Formal verification of annotated textual use-cases. Comput J 58:1495–1529
63. Szwed P (2015) Verification of ArchiMate behavioral elements by model checking. In: Saeed K, Homenda W (eds) Computer information systems and industrial management: 14th IFIP TC 8 international conference, CISIM 2015, Warsaw, Poland, September 24–26, 2015, proceedings, Springer International Publishing, Cham, pp 132–144
64. Jiang Y, Qiu Z (2012) S2N: model transformation from SPIN to NuSMV. In: Presented at the PROCEEDINGS of the 19th international conference on Model Checking Software, Oxford, UK
65. Szpyrka M, Biernacka A, Biernacki J (2014) Methods of translation of petri nets to NuSMV language. In: CS&P, pp 245–256
66. Browne MC, Clarke EM, Grümberg O (1987) Characterizing Kripke structures in temporal logic. In: presented at the The International Joint Conference on theory and practice of software development on TAPSOFT '87, Pisa, Italy
67. Reniers MA, Willemse TAC (2011) Folk theorems on the correspondence between state-based and event-based systems. In: Černá I, Gyimóthy T, Hromkovič J, Jefferey K, Královič R, Vukolić M, et al. (eds) SOFSEM 2011: theory and practice of computer science: 37th conference on current trends in theory and practice of computer science, Nový Smokovec, Slovakia, January 22–28, 2011. Proceedings, Springer Berlin Heidelberg, Berlin, pp 494–505
68. Ghobaei-Arani M, Rahmanian AA, Souri A, Rahmani AM (2018) A moth-flame optimization algorithm for web service composition in cloud computing: simulation and verification. Softw Pract Exp 48:1865–1892
69. Souri A, Nourozi M, Rahmani AM, Navimipour NJ (2018) A model checking approach for user relationship management in the social network. Kybernetes. https://doi.org/10.1108/K-02-2018-0092092
70. Bouneb M, Saidouni DE, Ilie JM (2015) A reduced maximality labeled transition system generation for recursive Petri nets. Formal Aspects Comput 27:951–973
71. Sibay GE, Braberman V, Uchitel S, Kramer J (2013) Synthesizing modal transition systems from triggered scenarios. IEEE Trans Softw Eng 39:975–1001
72. Souri A, Rahmani AM, Jafari Navimipour N (2018) Formal verification approaches in the web service composition: a comprehensive analysis of the current challenges for future research. Int J Commun Syst 31:1–27
73. Rozier KY (2011) Linear temporal logic symbolic model checking. Comput Sci Rev 5:163–203
74. Zhao Y, Rozier KY (2014) Formal specification and verification of a coordination protocol for an automated air traffic control system. Sci Comput Program 96(Part 3):337–353
75. Bollig B (2016) On the minimization of (complete) ordered binary decision diagrams. Theory Comput Syst 59:532–559
76. Sharma A (2012) A two step perspective for Kripke structure reduction. arXiv preprint arXiv:1210.0408
77. Gradara S, Santone A, Villani ML, Vaglini G (2004) Model checking multithreaded programs by means of reduced models. Electron Notes Theor Comput Sci 110:55–74
78. Flanagan C, Godefroid P (2005) Dynamic partial-order reduction for model checking software. In: Presented at the proceedings of the 32nd ACM SIGPLAN-SIGACT symposium on principles of programming languages, Long Beach, California, USA
79. Reniers MA, Schoren R, Willemse TAC (2014) Results on embeddings between state-based and event-based systems. Comput. J 57:73–92