Model Checking: Verification or Debugging?

Theo C. Ruys and Ed Brinksma
Faculty of Computer Science, University of Twente.
P.O. Box 217, 7500 AE Enschede, The Netherlands.

Abstract  Model checking tools are increasingly being used for the validation of real-life systems in an industrial context. This paper discusses two validation approaches with respect to the application of model checkers. The verification approach tries to ascertain the correctness of a formal model of a system, whereas the debugging approach tries to find errors in the model. This paper discusses the differences between the two complementing approaches and shows for each approach its advantages and disadvantages.

Keywords: model checking, verification, debugging

1. Introduction

Model checking [2, 15] is an automated technique that, given a finite-state model $M$ of a system and a property $\phi$ stated in some formal notation (e.g. temporal logic), systematically checks the validity of the property. In other words, model checking tools verify whether $M \models \phi$ holds. Model checkers are being put forward as ‘press-on-the-button’ tools, which automatically check whether the property $\phi$ is satisfied or not. The reduced level of user interaction is seen as an advantage for industrial applications, as it has better chances of being used by non-experts.

Before ’pressing-the-button’, however, the user still needs to come up with

- a model $M$ of the system under verification;
- the set of properties $S$ which the system should satisfy [4, 16].

Due to the infamous ‘state space explosion’, the model $M$ will generally be too big to be checked exhaustively by the model checker. As a result, the user is often also forced to

- make abstractions of the model $M$ [1, 17];
- exploit the optimising options and settings of the model checking tool to ‘tune’ the validation process.

The success and popularity of model checking tools is largely based on the bugs and errors that those verification tools have exposed in (existing) systems and standards [11]. Until recently, complete verification has only been feasible for small, toy-like systems. Now that model checking tools are becoming more powerful and widespread, the application of model checkers is slowly shifting from debugging to verification. In this paper we discuss the implications of this shift on the model checking process.

In the past years our group has been involved in several industrial projects concerning the modelling and validation of (communication) protocols [3, 14, 23]. In these projects we used modelling languages and tools – like Promela, Spin [8, 9, 11] and Uppaal [19] – to specify and verify the protocols and their properties. Consequently, our experiences with model checking tools are mainly based on the analysis of (software) protocols. The issues raised in this paper, however, also apply to hardware model checking.

In the next section, we briefly discuss the characteristics of the verification and debugging approach to model checking. In Sect. 3 we focus on the most important differences between the two approaches. In Sect. 4 we conclude the paper.
2. Verification and Debugging

To put it bluntly: the verification approach tries to ascertain the correctness of a detailed model \( M \), whereas the debugging approach tries to find errors in a model \( M \). One might say that the verification approach is the same as the debugging approach where no errors are being found, but there is more to it; both approaches differ in several ways with respect to the use of the model checking tool. In this section we try to give the characteristics of both approaches.

**Life-cycle of the system** Before describing both approaches, we will first discuss the place in the life-cycle of the system where both approaches are generally and preferably applied. Figure 1 (from [20]) shows the classic “waterfall model”, the life-cycle paradigm to software development.

The verification approach is mostly used after the design of the system has stabilised. The model \( M \) of the system is based on the (functional) design of the system. The user requirements as captured during the system engineering phase form the basis for the set of properties \( S \) that should be satisfied by the model \( M \).

The “classic” debugging approach is applied much earlier in the life-cycle. During the analysis or design phase, a prototype model \( M \) of the system is constructed and during these phases this model is validated against the user requirements. The model checking activities are performed by the analysis and design team itself. In general, several prototype models \( M \) are validated.

The debugging arrow after the code phase depicts the recent attempts (e.g. [5, 7, 13]) to use model checking tools directly on the implementation of the system. Although interesting and promising, this type of debugging is not of interest for this paper; we stick to the application of model checking technology in the earlier stages of the design.

From the above it will be clear that, to industry, an external verification project can sometimes be an attractive way to validate the design of a system:

- An external party formally checks the final design of the system. The only input to this external verification team is the (functional) design of the system and the user requirements.
- No knowledge of the verification technology is required and no (financial or educational) investment into the model checking tools is needed.
- The analysis phase of the company does not have to be altered. The debugging approach, on the other hand, might induce changes in the development process.
2.1 Verification

The purpose of the verification approach is to come up with a correct model on a certain level of abstraction. The verification approach is characterised by the following:

- During the validation\(^1\) phase the model \(M_v\) of the system is fixed at a certain level of abstraction.
- All properties \(\phi_i \in S\) are systematically validated.
- During the validation of a property \(\phi_i\), abstractions have to be made of the model \(M_v\).

Part (a) of Fig. 2 presents the global pseudo-algorithm\(^2\) for the verification approach. The verification approach starts with a detailed model \(M_v\) of the system. Before starting the actual validation loop in the verification approach, the detailed model \(M_v\) is simulated to obtain an initial degree of correctness. In general, the state space of this model \(M_v\) will be too large for an exhaustive search by a model checker [8]. Therefore, in the validation loop of the verification approach, one has to make abstractions of parts of the complete model \(M_v\). These abstractions are guided by the property \(\phi_i\). The validation phase is ended when the model \(M_v\) has been checked against all properties \(\phi_i \in S\).

2.2 Debugging

The debugging approach aims at finding errors and weaknesses in the (initial) design of a system. The debugging approach focuses its attention on those parts of the system where flaws are most likely to occur. This approach is characterised by the following:

- The validation phase is started with a model \(M_d\) on a high level of abstraction. During the validation phase, this level is not fixed.
- During the validation phase, one zooms in at certain aspects of the model \(M_d\) using local refinement techniques.
- There is no fixed set of properties \(S\) which the model \(M_d\) should satisfy. Properties are added and checked “on-the-fly” of the validation process.
- Only a limited part of the system is validated and no information is obtained about the non-validated components. In this respect, the debugging approach resembles the nature of testing (e.g. see [21]): testing can only show the presence of errors, not their absence.

Part (b) of Fig. 2 presents a pseudo-algorithm for the debugging approach. The debugging approach starts with an abstract model \(M_d\) of the system. In the validation loop, the debugging approach tries to find errors by adding details to model \(M_d\). The validation phase of the debugging approach is ended when (enough) errors have been exposed or when resources (e.g. time, money) have run out.

Please note that both approaches prescribe methods at the extreme ends of the validation spectrum. In practice, one usually adopts a combination of both approaches. We have used both approaches in our validation work.

3. Comparison

In the previous section we already saw that the place in the life-cycle of the development of a system is the first apparent difference between both approaches. Below we discuss other important aspects of the application of model checking tools in relation to the two approaches.

Model The initial model of the system that is used at the start of the validation trajectory is different for both approaches.
The verification approach is centralised around the model \( M_v \) that is being verified. This model \( M_v \) is supposed to be a sensible and correct abstraction of the system under verification. Ideally, this model has a fixed level of abstraction. A structured and readable model is important for verification purposes. In [22] we proposed to use literate techniques [18] to enhance the readability and accessibility of the verification model \( M_v \). An advantage of the verification approach is that after a successful verification trajectory, a validated, detailed and executable specification of the system (i.e. model \( M_v \)) is obtained. This specification can be used in later phases of the design. A drawback of the verification approach is that the modelling phase normally takes more time because of the required level of detail in the initial model \( M_v \).

For debugging purposes, the model is not important as a means of communication and is used only as a vehicle to find errors and weaknesses in the system as soon as possible. The debugging approach will usually not result in a complete specification of the design. The abstraction level of the final model may be unbalanced in the sense that the debugging engineer has only zoomed into parts of the model \( M_d \) where he suspected to find errors.

Another aspect of the model \( M \) which is being checked is the efficiency of \( M \) with respect to the corresponding state space. Because of the aforementioned state space explosion, minimisation of the number of states and the state vector (i.e. the information to be stored for each state) is an important aspect of the activities of the validation engineer. Experience has shown that there is generally a big difference in efficiency in the models developed by a ‘casual’ user and the models developed by an ‘expert’ user. The expert user exploits ‘assembler programming’-like tricks to make the model as minimal as possible. Furthermore, the expert user takes advantage of the optimising options of the model checker to ‘tune’ the verification process even further. Naturally, ‘assembler programming’-like tricks are less problematic for the debugging approach than for the verification approach, as the readability of the model is of less importance for the debugging approach.

Abstractions As mentioned above, the state space of a model \( M \) together with the property \( \phi \) to be checked, is generally too big to be checked exhaustively. In order to reduce the state space of the model \( M \), abstractions have to be made.

The verification approach requires a strong relation between the abstract model \( M_{abs} \) and the original model \( M_v \): if the abstract model \( M_{abs} \) is proven to be correct with respect to the (safety) property \( \phi \), this should imply that the original model \( M_v \) is correct with respect to \( \phi \) as well. Strong abstraction relations are usually obtained by replacing explicit choices in \( M_v \) by non-determinism in \( M_{abs} \).

For the debugging approach, weak relations between the abstract model \( M_{abs} \) and the model \( M_d \) are sufficient: if an error is found in the abstract model \( M_{abs} \), it is certain that the error

(a) verification approach

1. procedure verification
2. Start with detailed model \( M_v \)
3. Simulation of \( M_v \): sanity check
4. while not all properties \( \phi_i \in S \) checked do
5. Focus on particular property \( \phi_i \)
6. Make abstractions of model \( M_v \) as needed
7. Model check \( M_{abs} \) against \( \phi_i \)
8. od verification

(b) debugging approach

1. procedure debugging
2. Start with abstract model \( M_d \)
3. Simulate and model check \( M_d \)
4. while not errors found and resources available do
5. Zoom in on certain aspects of the model \( M_d \)
6. Simulate and model check \( M_d \)
7. od debugging

Figure 2: Pseudo-algorithms for both approaches.
also appears in the original model $M_d$. Weak abstractions are usually obtained by removing behaviour (e.g. statements) from the original model $M_d$.

**Partial search** For state spaces that are too big to be checked exhaustively, model checking tools often support options to partially search the state space. Spin, for example, provides the so-called “bitstate” hashing or supertrace technique [12], that can perform verifications with a relatively high coverage within a memory area that may be orders of magnitude smaller than required for exhaustive verifications.

In general, the user of a model checker should try to exhaustively analyse the state space of the model $M$ and the property $\phi$. This especially holds for the verification approach. However, there may be cases where reduction of the state space by abstraction is too costly (i.e. time consuming) and where a partial search becomes a serious option.

For the debugging approach the partial search mode even seems to be a natural choice. Because a partial search of the state space is in general much faster than its exhaustive counterpart, this mode is convenient when results are needed as fast as possible.

**Management of results** The verification approach requires the complete verification trajectory to be carefully controlled and managed: all verification results should be reproducible. This involves the management of all versions of the models, the properties, the verification runs, the verification results, etc. Without tool support, the quality of the verification process depends on the accuracy of the persons who conducted the verification.

The debugging approach is only interested in the errors found in the model and the corresponding error traces. This information should be automatically saved. Other aspects of the validation trajectory, however, are of less importance to the debugging approach.

**Switching between the two approaches** When the verification approach is being used and several errors are being exposed during the model checking process, it is of course possible to switch to the debugging approach to find as many bugs as possible. The other way around is more problematic. If the debugging approach does not reveal any errors, in general, the model $M_d$ has to be changed considerably to be used as a model $M_v$ for the verification approach. The model $M_v$ has a fixed level of abstraction and is optimised to be readable and accessible, all properties that the model $M_d$ does not have in general. On the other hand, the properties that have been checked during the debugging approach may serve as the starting set of properties that the new $M_v$ should satisfy.

**Marketing** Industry is interested in methods and tools that eliminate errors from their systems as soon as possible in the development of these systems. Research into computer aided verification, on the other hand, often stresses the fact that model checking tools are formal “verification” tools. If model checkers would be sold as “smart and fast debugging” tools, the acceptance of model checking technology would probably increase.

4. Conclusions

In this paper we have discussed two extreme approaches with respect to the use of model checking tools. The purpose of the verification approach is to come up with a correct model on a certain level of abstraction. The debugging approach aims at finding errors and weaknesses in the (initial) design of a system. Both approaches prescribe extreme methods for validation. In practice, one usually adopts a combination of both approaches.

There is a clear tension between both approaches. The debugging approach tries to use all modelling and verification tricks to zoom into the weaknesses of a system. The model $M_d$ only serves as an efficient vehicle to find errors as soon as possible. The verification approach resembles the classic (manual) formal verification of a system (e.g. see [6]) in the sense that the correctness of a formal description is proven
correct with respect to another formal description.

Currently, model checking is clearly most effective in combination with the debugging approach. The number of situations were the verification approach is required is growing though. To close the gap between debugging and verification, we propose the following improvements:

- The education of users of model checkers should be improved. Especially the technology transfer towards industry should be further enhanced.
- The difference in effectiveness between a casual user and an expert user is currently too big. Now that model checking tools in general, and Spin in particular, are becoming more widespread in use [10], these tools are starting to be used by people that only want to press the button and that do not know precisely what is ‘under the hood’ of such verification tools. Model checkers should incorporate techniques from optimising compilation technology: inefficient verification models should be translated to optimal state spaces.
- Currently, a lot of manual effort has to be put into the abstractions of a model $M$ to fight the state space explosion. Ideally, these abstractions should be executed automatically. In the future, manual abstractions and local refinements may be verified automatically using theorem proving technology.
- Verification tools should be equipped with ‘software configuration management’ functionality such that the complete validation trajectory will always be reproducible.
- The message about model checking technology should change: automatic verification is not about proving correctness, but about finding bugs much earlier in the development of a system.

Acknowledgements

The authors want to thank Pedro D’Argenio and Joost-Pieter Katoen for their useful suggestions to improve the readability of the paper.

References


