# Using Abstraction in Model Checking Z Specifications

M. U. Siregar, J. Derrick
*Verification and Testing Lab., Department of Computer Science, Faculty of Engineering, University of Sheffield.*

**Abstract**

Z notation is a language used for writing formal specifications of a system. However, tool support for this language is lacking. One such tool that is not generally available is a model checker. Model checking is a method used to verify that a system has certain properties; this is important since it can provide full verification of a finite state system without the user having sophisticated knowledge. Originally applied in hardware systems, it is now commonly available for application in software systems. One of the drawbacks of model checking is that it applies to finite state systems, since it works by performing a complete state space exploration. However, the size of the systems that model checkers can now cope with has increased rapidly. In this paper, the use of abstraction as a means to make model checking feasible for arbitrary Z specifications is investigated. Several experiments have shown that the abstract models have fewer states than the concrete ones or have the same number of states as the concrete one.

**Keywords** Abstraction; Model Checking; SAL Model Checker; Z Specification; Z2SAL

## 1. INTRODUCTION

Z is a language that can be used to write formal specifications. The aim of Z is that it can be used to define a specification of a system. As a formal language, Z could make a specification free from ambiguities.

Whilst there has been increasing interest in the use of Z, the tool support for Z is limited. There are many aspects to this, but one of the deficiencies in the tools available is that there is no support for model checking Z specifications. Although the Community Z Tools (CZT) project has continuously developed a set of open source tools for Z, its progress is slow [1]. In particular there is currently no model checker available for Z either in CZT or elsewhere.

It is well recognised that to build a model checker directly for a Z specification would take considerable effort due to the abstraction of the language. Therefore, alternative methods of providing support have been explored, such as translating the input of a Z specification into a language that a model checker tool accepts. It is quicker to do such an adaptation than write a model checker for the language from scratch.

As part of this work, researchers at the University of Sheffield implemented a translation tool which takes a Z specification and translates it into the input for model checkers found in the SAL tool-kit. Specifically, they used a LaTeX mark up of a Z specification as the input to the translation tool, which they called Z2SAL, and translated that into a representation that SAL could use [2].

In another case, Smith and Winter reported that a Z specification can consist of complex predicates as well as large number of or even infinite state spaces [3]. In contrast, state space explosion has been showed in many references as the most challenging problem on model checking [4]. As a result, Smith and Winter have proposed the approach of abstraction to Z specification systematically [5].

Clarke *et al.* have used several methods of counterexamples guided abstraction in order to solve the explosion problem of model checking, such as using SAT solver to simulate the counterexample and using ILP and machine learning to refine the abstract model [5, 6], and using symbolic algorithm to refine the abstract model [7].

Regarding abstraction on Z specification, Jackson has built a prototype which integrates abstraction on Z and VDM specification and has shown its functionalities during verification of some examples in those languages [8].

In this paper, we investigate the use of abstraction in model checking Z specifications by conducting several experiments. At the moment, this investigation is done manually for the generation of the abstract model, detection of false counterexample and refinement of that abstract. However, the first abstract model is built by modifying the SAL file of the concrete one generated by Z2SAL. Verification is also done automatically by using the SAL model checker.

## 2. METHOD

The abstraction process refers to the work of Smith and Winter as follows [3]:

Step 1: Abstract generation

To avoid initial value of output by Z2SAL, that output is initialised to 0 for numbered types. Generating the abstract model involves several steps as follow:

1.1. Define “monomials” [9] from atomic predicates of properties (LTL theorems).

1.2. Create state schema whose number of states is equal to the number of monomials.

1.3. Create abstract function which is a mapping from monomials to state.

1.4. Derive initial schema.

1.5. Derive operational schemas.

1.6. Derive abstract version of properties.

Step 2: False counterexample detection

If the property can be proved or the counterexample belongs to the original specification, then go to Step 4. If this is not the case, refinement takes place. This brings us to Step 3. The approach that is used to detect a false counterexample is based on Clarke, *et al.* [10].

Step 3: Refine the abstract model

This step involves a number of refinements that are called if the refined property cannot be proved. This is needed to avoid the false counterexample during processing with the model checker. Smith and Winter used the approach developed by Clarke, *et al.* [10].

## 3. RESULTS AND DISCUSSIONS

Several experiments have been done. However, due to page limitation, only two are considered here. The first is based on the example given by Smith and Winter [3]. The second by using the counterMod4 specification.

For the first example, the original model can have many states which can request a number or send an allocated number. A LTL theorem was added to prove that the system can never send the same number more than once. This will be proved by SAL model checker. Based on that theorem, there is one atomic property which then generates two monomials representing two states. Thus, in the first abstraction, this system will be modelled by only two states. However, the same property cannot be proved, and a counterexample was generated instead. After four refinement cycles, the property embedded on the abstract model can be proved. This latest abstract model has four states with the biggest number of states being 10.

For the second, the original model can have four states to do modulo four operation. A theorem was added to prove that generally from number one, next will be two. The first abstraction generated four states, but one state is not possible and was later deleted. Moreover, another state was too complex and was split into two states instead. Thus, one can do further checking before defining the abstract function. This abstraction could not be proved by the SAL model checker. After one further refinement, it was proved. Although the last abstract model also has four states, the number of states, if the states are not deleted, is seven.

Based on these experiments, systems with large state spaces will get more advantages from this abstraction. On the other hand, the simple ones might have the same number of states with their abstract models.

## 4. CONCLUSIONS

As a conclusion, this abstraction can reduce the number of states of complex systems. Therefore, it can be utilised further to gain a beneficial effect in verification of large systems, and even infinite systems.

## ACKNOWLEDGEMENTS

This work was initially based on work of John Derrick, Siobhan North and Anthony Simons on their Z2SAL. Furthermore, last summer, this work has been extended to take into account other significant fields of interest offered by Graeme Smith and Kirsten Winter on their Abstraction paper. This study has been supported financially by ISIHEMORA the Republic of Indonesia.

## REFERENCES

1. Derrick, J., North, S., and Simons, A.J.H. Formal Asp. Comput. Z2SAL: a Translation-based Model Checker for Z. 2011;23(1):43-71.
2. Derrick, J., North, S., and Simons, A.J.H. Issues in Implementing a Mode Checker for Z. In: Liu, Z. and He, J. ICFEM. Berlin Heidelberg: Springer-Verlag; 2006. p. 678-696.
3. Smith, G., and Winter, K. Proving Temporal Properties of Z Specifications Using Abstraction. In: Bert, D., et al. ZB. Berlin Heidelberg: Springer-Verlag; 2003. p. 260-279.
4. Clarke, E.M., Grumberg, O., and Peled, D. Model Checking. USA: MIT Press; 2001.
5. Clarke, E.M., Gupta, A., and Strichman, O. SAT-based counterexample-guided Abstraction Refinement. IEEE Trans. on CAD of Integrated Circuits and Systems. 2004;23:1113-1123.
6. Clarke, E.M., et al. SAT Based Abstraction-Refinement Using ILP and Machine Learning Techniques. In: Brinksma, D., and Larsen, K.G. CAV. Berlin Heidelberg: Springer-Verlag; 2002. p. 265-279.
7. Clarke, E.M., et al. Counterexample-guided Abstraction Refinement for Symbolic Model Checking. J. ACM. 2003;50:752-794.
8. Jackson, D. Abstract Model Checking of Infinite Specifications. In: Naftalin, M., Denvir, T., and Bertran, M. FME. Berlin Heidelberg: Springer-Verlag; 1994. p. 519-531.
9. Graf, S., and Saidi, H. Construction of Abstract State Graphs with PVS. In: Grumberg, O. CAV. Berlin Heidelberg: Springer-Verlag; 1997. p. 72-83.
10. Clarke, E.M., et al. Counterexample-Guided Abstraction Refinement. In: Emerson, A.A., and Sistla, A.P. CAV. Berlin Heidelberg: Springer-Verlag; 2000. p. 154-169.