IJID International Journal on Informatics for Development, *e-ISSN :2549-7448*
Vol. XX, No. XX, 2017

Translations of Embedded Theorems in Z Specifications

Abstract—**This paper discusses our proposal on how to embed theorems in Z specifications. One reason behind this proposal is to ease Z users in writing theorems directly in their Z specifications. Another reason is not to overwhelm Z users in learning other language, which in this case is SAL language. In doing so, we need to inform Z2SAL programmers how to translate these embedded theorems into equivalence theorems in SAL specifications. Based on our experiments, Z2SAL is able to translate these kind of theorems and SAL model checker is also able to model check SAL specifications with theorems that are written directly in the Z specifications.**

Keywords--Z; theorems; Z2SAL; SAL model checker.

I. INTRODUCTION

Previously, a user added several theorems to the generated SAL file in order to allow the system, which is modelled by the Z specification, verifies these theorems. By doing so, this user should know how the SAL language presents this theorem, which might be a problem to learn other language, the SAL language, especially a user who just knows Z language.

Then, we had an idea, how if the user specifies the theorem inside the Z specification. We proposed this idea to Z2SAL programmers and the method how to achieve this goal. Z2SAL programmers accept our proposal and follow our method. As a result, the current Z2SAL can translate either a Z specification or a Z specification added with theorems.

Following is brief descriptions about Z notation, Z2SAL, SAL model checker, and Model Checking. For further explanation, interested readers can refer to paper, especially the ones in [1,2,3,4].

A. Z Notation

Z is a notational convention for logic and simple mathematics. Z is a model based notation, which has states and operations. As mentioned earlier, it is a notation, not a method. Furthermore, Z is a not tool either, but several tools implementing Z are available, tough is not many. Z is also not an executable since it is not a programming language. Z is able to express concurrency and objects after it has been extended. Z is usually used to design a specification of a system. Thus, this specification tells us what the system can do, not how to do something.

B. Z2SAL

Z2SAL is a translation tool built by researchers from the University of Sheffield, United Kingdom. They John Derrick, Siobhan North, and Anthony Simons. As its name, it can do translations on its input file which is a Z specification to its output which is a SAL specification representing that Z specification. Sometimes, Z2SAL also generates several context files, which are mathematical toolkits, needed to model check that SAL file later with SAL model checker. Mathematical toolkits built for Z2SAL are rich enough to represent Z notation.

In addition to a translation tool, Z2SAL can also do refinement. Achieving this function, Z2SAL needs two Z specification files.

C. SAL Model Checker

SAL model checker is a tool that can model check systems. It can accept inputs of SAL specifications. SAL, which stands for symbolic analysis laboratory, previously is a collaboration research of two famous universities which then this tool is developed further at SRI. SAL model checker has ability to do symbolic model checking with command smc. A symbolic model checker supports LTL (Linear-time Temporal Logic) and CTL (Computation Tree Logic) formulas. SAL model checker can also do bounded model checking with command bmc. This bounded model checker supports only LTL formulas.

D. Model Checker

Model checker is one method of automatic formal verification. This method consists of three steps: modelling, formalization of properties, and verification [5]. Modelling is performed using one of formal specification languages. Formal logics, which are usually in temporal logics form, are used to do the second step. The third step is to check whether the model satisfies properties/ theorems given in temporal logics form.

As mentioned above, formal logics in model checker is formed from temporal logics. These temporal logic, which are used in theorems, are to specify concurrent systems. This logic can describe events in ordered time. With this logic, a formula can be true in some states and false in other states dynamically.

Based on time, the temporal logic is classified into two: the linear time logic (LTL), and the branching time logic (CTL). In LTL, a time is a set of paths, where a path is a sequence of time instances. Meanwhile, in CTL, a time is represented as a tree, rooted at the present moment and branching out into the future.

SAL model checker can support both of these logics, please check the above description.

In this paper, we discuss briefly this proposal. A further discussion can be read in [6]. We begin the discussion with the current method, which is adding theorems in the generated SAL specification. The flow of the following section begins with the examples of theorems. Then, we do manual verification on these theorems. Finally, we use SAL model checker to do the verification.

II. ADDING THEOREMS IN THE GENERATED SAL

A. Examples

At the end of our SAL file of `club.tex` (can be read in [6] subsection 2.2.4, several LTL theorems and CTL theorems were added as presented in [6] subsection 3.1. We write again here those theorems as follows:

- `th1: THEOREM State |- G (NOT (members = set {PERSON;} ! full));`

  It is not the case such that a club ever gets full. `G` means always.

- `th2: THEOREM State |- G (NOT(set {PERSON;}! empty?(members)));`

  It is not the case such that the club ever be empty.

- `th3a: THEOREM State |- G (EXISTS(m, n: PERSON): m /= n);`

  There exists at least one instance of `members`, who is different from other `members`. `EXISTS` represents an existential statement, whereas `/=` is an operator to ask whether to variables are not the same.

- `th3b: THEOREM State |- G (NOT(EXISTS(m, n: PERSON): m /= n));`

  It is not the case such that there is a member of `members`, who is different with others.

- `th4a: THEOREM State |- G (FORALL(m, n: PERSON): m /= n);`

  All `members` are different. `FORALL` represents a universal statement.

- `th4b: THEOREM State |- G (NOT(FORALL(m, n: PERSON): m /= n));`

  It is not the case such that all `members` are different.

The next subsection will discuss about verifications of those theorems.

B. Manual Verification

Before these theorems were verified by using SAL model checker, they were investigated manually. Following is the discussion of this manual verification.

For the first theorem, `th1`, it should be invalid since there is an operation `JoinOk` that can add a member to this club. Furthermore, this operation only stops if the maximum number of members is reached.

For `th2`, it is also invalid since in the initialization of this system, this club has no member. In other words, this system has ever been empty, especially in the initialization stage.

For `th3a`, it will be proved. The operation performed by `JoinOk` schema will only add a new member who has not been available in this club.

For `th3b`, it is the opposite of `th3a` theorem. Thus it is invalid.

`th4a` is also invalid due to the assignment of no members for this club in the initial state. Thus, in the initialization state, all members are the same, which are empty.

The last theorem, `th4b`, is the opposite of theorem `th4a`. Thus, it will be proved or it is valid.

C. Verification by SAL Model Checker

Based on these prior knowledge, SAL model checker was run on this generated SAL file to verify those theorems. The summary of that verification is given as follows and they are the same as our expected results:

```
  The assertion 'th1' located at [Context:
club, line(55), column(0)] is invalid.

  The assertion 'th2' located at [Context:
club, line(58), column(0)] is invalid.

  The assertion 'th3a' located at [Context:
club, line(61), column(0)] is valid.

  The assertion 'th3b' located at [Context:
club, line(64), column(0)] is invalid.

  The assertion 'th4a' located at [Context:
club, line(67), column(0)] is invalid.

  The assertion 'th4b' located at [Context:
club, line(70), column(0)] is valid.
```

Many other examples given on other sections in [6] show this practice. Let us now move to the next discussion on embedded theorems.

III. EMBEDDED THEOREMS ON Z SPECIFICATIONS

Duke and Smith in [7] mention that properties of a system such as liveness can be evaluated by presenting a specification of a system using Z notation. There are two alternatives to express such properties.

The first alternative is to express them using Z notation. The second one is to express them using temporal logic notation. One benefit of using the second alternative is predicates are more readable and shorter than the former.

Supporting that second alternative, King [8] added tags for presenting several temporal logics to his Object Z package. Currently there are three tags available, as follows:

- □: this symbol means always
- ○: this symbol means next
- ◇: this symbol means eventually

Based on their meanings, we assume those three tags represent LTL formulas.

Regardless of research found in [2,3], only fewer tools that support embedded temporal theorems in Z specifications. Therefore, our research aims to propose extensions to Z standard notation adapted by Z2SAL to also include King's temporal logic.

In our proposal [6], syntax to define the embedded theorems is adapted from Object-Z Concrete Syntax [9]. This syntax is an extension to syntax of Spivey [10]. Based on discussion on [6], the theorems are defined in the predicate part of schemas.

We propose several steps of how to translate embedded properties on a Z specification [6]. The translation of these properties will follow Z2SAL's form of theorems in SAL specifications which have a form as follows:

```
th i: THEOREM name_of_module |-
temporal_logics;
```

Our steps are:

- `th` is an identifier, so it is possible to modify this identifier's name to other identifier's name.
- Every `th` is followed by *i*. An *i* is a non-zero natural number starting from 1, which plays as an index. This *i* number will be given for every line of predicates containing temporal logics which begins with 1 and this number will be incremented by 1 for each successor of such a line. The *i* is also part of the user identifier, so it can be modified to other identifier.
- `THEOREM` is a SAL keyword, but it is not case-sensitive.
- *name_of_module* is taken from the name of the SAL's module and is case-sensitive.

As said in [6], King's style for representing temporal logics are different with temporal logics of SAL specifications. However, there are equivalences between both syntaxes. The table has been given in [6] which shows these equivalences (please refer to Table 3.1). From the table, SAL has `G` for representing always, `X` for representing next, and `F` is for eventually.

For representing `U(p,q)`, which is not supported in King's syntax, we have given the equivalent notation for `U(p,q)` in [6]. `U(p,q)` means that *p* holds until *q* holds on a particular path [4].

The next discussion gives one example of schema that represents a theorem from our experiments in [6] relating to this proposed translation method. The example is taken from [10] namely Birthday Book Specification.

As mention in [6], the property that need to be proven is "*If it is known the birthday of a person then the person should be recognized*". This schema has a predicate part that represent that property. It begins with a universal statement, thus for all person in this birthday book (we take their names), there will exist one unique date of birth for each person (it is represented by an existential statement). The schema that is defined is as follows [6]:

WhichDate

$\Xi BirthdayBook$

$\forall n: NAME \bullet \exists d: DATE \bullet$
$\Box\ (d = birthday(n) \Rightarrow n \in known)$

After the complete specification is translated by Z2SAL, the generated SAL specification for the above schema is as follows:

```
th1 : theorem State |- (FORALL (q__2 : NAME) : (EXISTS (q__3 :DATE) :
G (q__3 = birthday(q__2) => set {NAME;} !contains?(known, q__2))));
```

Figure 1 A SAL translation for the above embedded theorem

This theorem in Fig. 1 is VALID, in other word; it is satisfied by the system. The command that is given to SAL model checker to verify the above embedded theorem is as below:

```
$ sal-smc birthdaybook_templog
```

We specify a new theorem for this paper that we also embedded in the same Z specification as above. Thus theorem in not available in [6]. This theorem represents "if there is a birthday date for someone, it means that this person is already in the system which means that the name is not a new name. In

other word, only one name is recorded for one date. The theorem is as follows:

JustOnePerson

$\Xi Birthdaybook$

$newName?: NAME$

$\forall n: NAME \bullet \exists d: DATE \bullet$

$\Box(d = birthday(n) \Rightarrow newName? = n)$

Z2SAL generates a SAL theorem as follows:

```
th2 : theorem State |- (FORALL (q__4 : NAME)
: (EXISTS (q__5 : DATE) : G (q__5 =
birthday(q__4) => newName? = q__4)));
```

We model check the SAL specification then. The command given to SAL model checker and results are as follows:

```
$ sal-smc birthdaybook_templog

Summary:
The assertion 'th1' located at [Context:
birthdaybook_templog, line(71), column(2)]
is valid.
The assertion 'th2' located at [Context:
birthdaybook_templog, line(74), column(2)]
is valid.
```

`birthdaybook_templog` is the name of the context file which represents the name of SAL file. Thus, we use symbolic model checker of SAL model checker and we use LTL formulas. The first theorem uses a LTL operator, namely `G`. This operator means its argument is always true. It is similar to the second theorem; this theorem uses also the same LTL operator, `G`.

Based on result given by model checking the second theorem, we could assume that this birthday book system only specifies one date of birthday for one person. Thus, it does not support if there are many persons have the same birthday date. This behavior is quite surprising, it is not usual.

IV. RESULT AND DISCUSSION

Our complete results from experiments in this proposal are given in [6]. Interested readers are encouraged to read that paper. We could obtain the translations of theorems, which are embedded in Z specifications. Z2SAL are able to perform translation over these embedded theorems. We also could model check the generated SAL specifications using SAL model checker. Thus, SAL model checker seems that it does not differentiate the theorems, which are defined directly in the Z specification from the ones, which are defined in the SAL specification. For the discussion about the last paragraph in the previous section, to allow shared birthday date among person, the Z specification seems to be revised.

V. CONCLUSION

There are six experiments which have been conducted in our research and written in [6]. We reported again one of those experiments in this paper. In addition to this experiment, we add our new example for this paper. Thus, there are two experiments in this paper. Based on these experiments, we conclude that Z2SAL supports embedded theorems in Z specifications. It is because Z2SAL can translate these theorems into the equivalence theorems in SAL specifications. These theorems can also be verified by SAL model checker. Furthermore, many persons who share the same birthday date could be assigned as future research.

ACKNOWLEDGMENT

We want to thank John Derrick, Siobhan North and Anthony Simons who allow us use their Z2SAL. Furthermore, our thanks also are for discussion on Z2SAL, Z and SAL with those three researchers.

REFERENCES

[1] J. Jacky, The Way of Z: Practical Programming for Formal Methods. Cambridge University Press, 1997.

[2] J. Derrick, S. North, and T. Simons, “Issues in Implementing a Model Checker for Z,” in Formal Methods and Software Engineering, 2006, pp. 678–696.

[3] L. de Moura, S. Owre, and N. Shankar, “The SAL Language Manual,” 2019.

[4] M. Huth and M. Ryan, Logic in Computer Science: Modelling and Reasoning about Systems. Cambridge University Press, 2004.

[5] R. Pelanek, “Reduction and Abstraction Techniques for Model Checking,” Masaryk University, 2006.

[6] M. U. Siregar, “Support for Model Checking Z Specifications,” The University of Sheffield, 2016.

[7] R. Duke and G. Smith, “Temporal Logic and Z Specification,” Aust. Comput. J., vol. 21, no. 2, pp. 62–66, 1989.

[8] P. King, “Printing Z and Object_Z Latex Documents,” Dep. Comput. Sci. Univ. Queensl., vol. 393, pp. 404–410, 1990.

[9] R. Duke, P. King, G. Rose, and G. Smith, “The Object-Z Specification Language: Version 1,” 1991.

[10] J. M. Spivey, The Z Notation. Prentice Hall New York, 1989.