# Experiences Using Z2SAL

Maria Ulfah Siregar, John Derrick, Siobhan North, Anthony J.H. Simons
*Verification and Testing Laboratory, Dept. of Computer Science, The University of Sheffield, United Kingdom*
Email: acp12mus@sheffield.ac.uk

***Abstract*—The Z notation is a language that can be used for writing formal specifications of a system since it is based on mathematical notation and logic. However, there is less tool support for this language that one might wish for. In this paper, Z2SAL, a translator for Z which translates the Z notation into a SAL input language, is explored. The generated SAL file can be used further by an existing model checker, specifically ones provided in the SAL tool suite. This paper describes experiences during conducting several experiments on the Z2SAL translator.**

## I. Introduction

To date, computer applications have been used almost in every aspect of human life. Nevertheless, one needs those applications can do their jobs accurately, particularly safety-critical system.

To achieve that aim, several decades ago, natural language and graphics were used to draw system flowcharts and to write specifications. It turned out that natural language is inadequate as a vehicle for specification due to its imprecision. The alternative, which is the use of a programming language to write a specification is equally flawed in that it forces one to work at the wrong level of abstract [1].

Therefore, there is a role for a method of writing a specification that is not only precise enough but also implementation free. Moreover, such a method, if it is equipped with a proof theory, can help us to describe properties of specifications easily by conducting 'rigorous arguments' [1]. It needs a certain level of formality and for specifications to be written at a suitably high level of abstraction. Thus, mathematical notation is used which is based on set theory, logic, functions and relations to write those specifications. Notations used to do this are called specification languages or *formal methods*. Indeed, although their use is not widespread in every sphere, 'formal methods are recommended by many standards bodies concerned with Safety-Critical systems and for some they are mandatory' [2].

As a formal language, the use of Z can make a specification free from ambiguity. In addition, it can make such a specification be analysed mechanically [3].

Whilst there has been increasing interest in the use of Z, the tool support for Z is limited. There are many aspects to this situation, such as the abstraction and the logic of the language are undecided [3]. One of such deficiencies in tools is validating the intended meaning of a Z specification or model checking it [4], [5].

In this paper, we discuss the provision of a translation of the Z notation, in a tool called Z2SAL, into a format that an existing tool can be applied on. This exploration involves several experiments on the translator.

On providing a translator for Z into an input language of an existing tool, *Symbolic Analysis Laboratory* (SAL) was chosen since it has *similar representation* of many aspects of Z [7], such as *the module mechanism of SAL represents appropriately a Z state transition system* [6]. *SAL also supports expressive mathematics* which is a necessity in model checking an expressiveness of Z specification [6]. Moreover, *there exists many different tools that use the SAL input language* [5] which has been offered freely by SRI under academic licence such that attracts users to engage in international groups. SAL provides several tools reflecting its functions such as simulator of a system, model checker either symbolic or bounded, deadlock checker, etc. Some of them are detailed on I-A2.

The structure of this report is as follows. Section I-A describes the related works, this is followed by Section I-B, which discusses our experiments with Z2SAL. The next is Section II which concludes this paper and is followed by Acknowledgement and References.

### A. Related Works

In this section, we discuss the existing work on Z2SAL and the translation of Z notation into a SAL input language.

The idea of translating Z into a SAL input language is due to Smith and Wildman [6] at the University of Queensland, Australia. However, since the basic idea given in [6], the ideas have been implemented in a tool set, and the current Z2SAL has been extended in a different direction. In doing this, it has also had to tackle optimization issues [5], and thus is quite different from the ideas as originally envisaged.

*1) Z2SAL:* Z2SAL translates a Z specification into a SAL module. In this module, it groups a number of definitions including types, constants and modules to describe the states transition system [7]. A SAL module has general format as follows:

```
State : MODULE =
   BEGIN
    INPUT ...
    LOCAL ...
    OUTPUT ...
    INITIALIZATION [ ... ]
    TRANSITION [ ... ]
END
```

 978-1-4799-8075-8/14/$31.00 

There are several challenges to the translation of Z into the SAL input language [5]. *First* is *bounding the infinite*. Z supports *fully abstract* (non-grounded, non-constructive) specification styles, while SAL input language is a *concrete and grounded language*. For example, Z supports the built-in numerical types $\mathbb{Z}$, $\mathbb{N}$ and $\mathbb{N}_1$, whose ranges are infinite. On the other hand, the SAL has the similar unbounded types `INTEGER`, `NATURAL` and `NZNATURAL`, which can only be used as the base types of finite sub ranges in the actual specification. Z also supports basic types which have the semantics of un-interpreted sets, such as `[TAPE, NAME]`. Therefore, the translations provided by Z2SAL should define a finite number for those sets.

The *mismatched formal paradigms* is the second challenge. Z and SAL have very different styles of specification and description. A Z specification is built-up increasingly, which consists of state and operational schemas. It views locally and functionally such that every operational schema operates on its input and output variables, or on variables of the state schema. In contrast to this, a SAL specification is created as a 'monolithic finite state automaton' such that all inputs, outputs and local variables are compiled into the aggregate states and all operations act upon guard transitions from one state configuration to other state configurations [5]. Thus, this mismatch could be approached by re-ordering all the information in a Z specification. Another mismatch is Z specifications often use partial functions. This is to express incomplete operations of operational schemas and to express the associative data types, *maps* of the state schema, whose sizes are dynamics. By contrast, as SAL is based on *Binary Decision Diagrams* (BDDs), SAL always needs a representation of a function given as a total function. This means one needs a work-around in order to represent partial functions in Z specifications, which frequently exist, as total functions in SAL. Furthermore, a set cannot be treated as a monolithic of SAL, but as a 'polylithic collection of judgements' over its elements instead. Thus, several operations in a set need to be expressed differently, such as the cardinality of a set which is not supported by SAL.

The last challenge is the issue of *non-computable specifications*. A Z specification naturally supports non-constructive styles of specification. These styles need to be expressed in computable specification in SAL, which essentially are different. Normally, a SAL specification consists of a series of update assignments to primed variables, which indicates posterior variable states. In contrast, in a Z specification this direction of constructive approach is not necessary. Z2SAL adopts an assertion of posterior existence of variables and restricts their values in the precondition. This needs a searching for suitable precondition values.

Currently, the tool has two operating modes, which it will either translate a single Z specification into the input format of SAL for model checking purposes, or translate a pair of Z specifications for refinement checking purposes [8]. The translated output is placed in the same directory as the source. More information relating to Z2SAL can be found on related references. The Z language syntax can also be read further on [14].

*2) SAL:* SAL is a framework for combining different tools for abstraction, program analysis, theorem proving and model checking towards the calculation of properties (symbolic analysis) of transition systems [9]. Thus, SAL is used to change the perception and implementation of model checkers and theorem provers which previously based on verification to based on calculation of properties such as abstraction, slicing and composition [10].

As an intermediate language which serves as a medium for representing the state transition semantics of systems with their own source languages, SAL has been integrated with several loosely coupled back-end components. These components relate to each other by using well-defined interfaces [10].

The SAL environment contains a simulator for finite states specifications based on BDDs which allows users to explore different execution paths of a SAL specification [11]. By doing such an exploration, users will be more confident of their model before verification is done on such a model.

Regarding model checking, SALenv contains a symbolic model checker called SAL-smc (*simple model checker*). Users can specify properties in LTL and CTL temporal logics. In addition to SAL-smc, SALenv also contains SAL-bmc (*bounded model checker*) which only supports LTL formulas. By using bounded model checker, SAL can search on a state space on a given depth. When a property is invalid, a counter-example will be produced, otherwise, it will be proven. The SAL language syntax can be read further on [9].

### B. Experiments with Z2SAL

We have conducted several experiments with Z2SAL by providing Z specifications, and translating them with Z2SAL. The generated SAL could be processed further either by simulating or verifying them with SAL simulator or SAL model checker. Due to the page limitation, only few of them will be presented here, particularly specifications which have modification in their original specifications.

*1) Experiment with Hotel Specification:* This specification is taken from [12, p. 55-57]. The specification has one basic/ given type, `GUEST`, and has a data type definition `HOTELROOM` whose values are from `Room1` until `Room15`. It also has another data type definition `RESPONSE` which values are `no_room_vacant`, `not_a_guest`, `success`, `wrong_number`, and `add_to_tab_ok`. The state schema of this specification is:

 978-1-4799-8075-8/14/$31.00 

*Hotel*

$current\_guest : \mathbb{P}\, GUEST$
$unoccupied\_room, occupied\_room : \mathbb{P}\, HOTELROOM$
$occupies : GUEST \leftrightarrow HOTELROOM$
$tab : HOTELROOM \leftrightarrow \mathbb{N}$

---

$current\_guest = \mathrm{dom}\, occupies$
$occupied\_room = \mathrm{ran}\, occupies$
$unoccupied\_room = HOTELROOM \setminus occupied\_room$

There are new types which are formed by relating a basic type to a defined type, such as `occupies` whose domain is `GUEST` and whose range is `HOTELROOM`. This relation gives information about guests and their occupied rooms. There is also another relation, `tab` which relates `HOTELROOM` and a natural number. By the relation, every guest knows the price they should pay for their room.

The specification includes a relational composition which relates two relations to create a new relation. This new relation treats the domain values of the first relation as its domain and the range values of the second relation as its range. For example, `occupies;tab`, this operation will give us a new relation relating each guest to their bill. The schema that has this operator is `DepartGuest`:

*DepartGuest*

$\Delta Hotel$
$guest? : GUEST$
$bill! : \mathbb{N}$
$reply! : RESPONSE$

---

$\exists\, b : \mathbb{N} \bullet (guest? \in current\_guest$
$guest?(occupies \fatsemi tab)b$
$b = bill! \land occupies' = \{guest?\} \lhd occupies$
$tab' = tab \land reply! = success)$

The schema also contains a non-constructive, originated from Z styles, predicate in the second lines of the existential quantifier block.

Based on our experiment, Z2SAL cannot translate it. Thus, this predicate should be written in another way around as follows:

```
(guest?,b) ∈ (occupies;tab)
```

There is another schema that also contains the non-constructive predicate as above schema, as written below:

```
room? tab n
```

Thus, the related schema after its first line predicate modification is as follows:

TABLE I
EXPERIMENTS ON SOLVING THE OUT OF MEMORY ERROR

| Max size of GUEST | Max size of HOTELROOM | Result |
|---|---|---|
| 3 | 15 | Fail |
| 2 | 15 | Fail |
| 2 | 8 | Success |
| 1 | 15 | Success |
| 3 | 8 | Success |

*AddToTab*

$\Delta Hotel$
$room? : HOTELROOM$
$charge? : \mathbb{N}$
$reply! : RESPONSE$

---

$\exists\, n : \mathbb{N} \bullet (room?, n) \in tab$
$room? \in occupied\_room$
$tab' = (\{room?\} \lhd tab) \cup$
$\{room? \mapsto (charge? + n)\}$
$occupies' = occupies \land reply! = add\_to\_tab\_ok$

Indeed, these constructive writing are easy to read and understand. Both of those which are rewritten in other way around predicates express the constructive predicates which are supported by SAL model checker.

Although this specification can be verified by SAL model checker, it cannot be simulated by SAL simulator, due to ran out of memory. Originally, there are 15 rooms on `HOTELROOM` defined in Z specification and there are three guests on `GUEST` defined by Z2SAL.

There are three alternatives to combat the problem. The first is to reduce the size of `GUEST`. The second is the same as the first, but is done on `HOTELROOM`. The third is to reduce the size of both those given type.

All of our attempts are given on Table I. These experiments were conducted on a machine with Intel(R) Core (TM) i5-2320 CPU 3.00 GHz.

*2) Experiment with Telephone Network Specification:* This specification is taken from [13, p. 31-34]. The specification has one given type `[PHONE]`. It has one defined data type `Status` whose values are `Yes` and `No`. In order to translate this specification, several modifications must be taken place first.

*Firstly*, it contains a *generic constant*, such as:

$[X]$

---

$disjoint : \mathbb{P}\,\mathbb{P}\,\mathbb{P}\, X$

---

$\forall\, cons : \mathbb{P}\,\mathbb{P}\, X \bullet cons \in disjoint \Leftrightarrow$
$(\forall\, c1, c2 : cons \bullet c1 \neq c2 \Rightarrow c1 \cap c2 = \varnothing)$

A generic constant which is a generic construct supported by Z is used to define a parameter without explicit type. Some mathematical tool kits are defined by this generic constructor.

Unfortunately, to date, Z2SAL has not supported yet the generic constructs. To solve this problem, the generic constant was deleted and all occurrences of

 978-1-4799-8075-8/14/$31.00 

following predicate:

```
cons ∈ disjoint
```

in other schemas were deleted and were replaced by:

```
∀ c1,c2: cons • c1 ≠ c2 ⇒ c1 ∩ c2 = ∅
```

and referred to appropriate `cons`. For example, a state schema below:

*TN*

$reqs, cons : \mathbb{P}\, CON$

---

$cons \subseteq reqs \land cons \in disjoint$

it contains the predicate taken from the generic constant. The schema will be changed into:

*TN*

$reqs, cons : \mathbb{P}\, CON$

---

$cons \subseteq reqs$
$\forall c1, c2 : cons \bullet c1 \neq c2 \Rightarrow c1 \cap c2 = \varnothing$

`CON` is a connection in a set of `PHONE`.

*Secondly*, there are schemas which consist of a predicate referring to other schema and having parameters, namely schema references. For example, a schema as below:

*efficientTN*

*TN*

---

$\neg\, (\exists\, cons0 : \mathbb{P}\, CON \bullet cons \subset cons0 \land$
$TN\,[cons0/cons])$

and this schema:

$\Delta TN$

*TN*
$TN'$

---

$\neg\, (\exists\, cons1 : \mathbb{P}\, CON \bullet$
$(cons \setminus cons1) \subset (cons \setminus cons')$
$efficientTN'\,[cons1/cons'])$

For those schemas, changes were made by defining those schemas without including those references. For the first schema, `TN[cons0 / cons]` was replaced by all the contents of `TN` schema. Next, changing `cons` into `cons0`. Here is the new `efficientTN` schema:

*efficientTN*

*TN*

---

$\neg\, (\exists\, cons0 : \mathbb{P}\, CON \bullet cons \subset cons0 \land$
$cons0 \subseteq reqs$
$\forall c1, c2 : cons0 \bullet c1 \neq c2 \Rightarrow c1 \cap c2 = \varnothing)$

and below is the $\Delta$`TN` schema:

$\Delta TN$

*TN*
$TN'$

---

$\neg\, (\exists\, cons1 : \mathbb{P}\, CON \bullet$
$(cons \setminus cons1) \subset (cons \setminus cons')$
$\neg\, (\exists\, cons0 : \mathbb{P}\, CON \bullet cons1 \subset cons0 \land$
$cons0 \subseteq reqs$
$\forall c1, c2 : cons0 \bullet c1 \neq c2 \Rightarrow c1 \cap c2 = \varnothing))$

*Thirdly*, this specification also includes theta symbol which is used to bind information. The predicate is:

```
Θ TN′ = Θ TN
```

The schema consisting of the predicate is:

*Engaged*

$\Delta TN$
*engaged*! : *Status*
*other*! : *PHONE*

---

$\theta TN' = \theta TN$
$(engaged! = Yes) \Rightarrow (\{ph?, other!\} \in cons)$
$(engaged! = No) \Rightarrow ph? \notin (\bigcup cons)$

Z2SAL does not support this tag, so it was rewritten into its definition of laws based on [14, p. 62] and replaced by two lines of predicates as follows:

```
reqs′ = reqs ∧ cons′ = cons
```

These refer to laws of Θ [14]:

$$\Theta S' = \Theta S \Leftrightarrow x'_1 = x_1 \land ... \land x'_n = x_n$$

*Lastly*, Z2SAL has a standard meaning for a delta schema which says that variables in the state schema can change their after operational values. Therefore, the related schema only knows all variables that are listed in the state schema, state schema variables, so does the predicates. This is a convention but not enforced by the semantics. And indeed, in this specification, there is another meaning of a delta schema which is to add predicates not defined in the state schema. To overcome this problem, add all the variables and predicates of the delta schema into other schemas that refer to this schema, and keep those that are listed in the state schema. The delta schema ($\Delta$`TN`) was renamed into another name, `DeltaTN`, and its contents are as follows:

*DeltaTN*

*TN*
$TN'$
*ph*? : *PHONE*

---

$\neg\, (\exists\, cons1 : \mathbb{P}\, CON \bullet$
$(cons \setminus cons1) \subset (cons \setminus cons')$
$\neg\, (\exists\, cons0 : \mathbb{P}\, CON \bullet cons1 \subset cons0 \land$
$cons0 \subseteq reqs$
$\forall c1, c2 : cons0 \bullet c1 \neq c2 \Rightarrow c1 \cap c2 = \varnothing))$

 978-1-4799-8075-8/14/$31.00 

The operational schemas which call such a different meaning of $\Delta$TN schema are also modified appropriately. For example, the Engaged schema above will be modified into:

*Engaged*

$\Delta TN$
$engaged! : Status$
$other! : PHONE$
$ph? : PHONE$

---

$reqs' = reqs \land cons' = cons$
$(engaged! = Yes) \Rightarrow (\{ph?, other!\} \in cons)$
$(engaged! = No) \Rightarrow ph? \notin (\bigcup cons)$
$\neg\ (\exists\, cons1 : \mathbb{P}\, CON \bullet$
$(cons \setminus cons1) \subset (cons \setminus cons')$
$\neg\ (\exists\, cons0 : \mathbb{P}\, CON \bullet$
$cons1 \subset cons0 \land cons0 \subseteq reqs$
$\forall\, c1, c2 : cons0 \bullet$
$(c1 \neq c2) \Rightarrow (c1 \cap c2 = \varnothing)))$

Z2SAL has also been updated by revising its translation for universal quantifier which appears on this specification. The predicate with this quantifier is as follows:

$$\forall\, c1, c2 : cons \bullet (c1 \neq c2) \Rightarrow (c1 \cap c2 = \varnothing)$$

Previously, it was translated by Z2SAL as follows:

$(FORALL(q\_1 : CON, q\_2 : CON) :$
$(q\_1/ = q\_2 =>$
$set\{PHONE;\ \}!intersection(q\_1, q\_2) =$
$set\{PHONE;\ \}!empty)AND$
$set\{CON;\ \}!contains?(cons, q\_1)AND$
$set\{CON;\ \}!contains?(cons, q\_2)))$

Based on the Z book [15, p . 31]

$$\forall\, x : a \mid p.q$$

this is equivalent to:

$$\forall\, x : a.p => q$$

Thus, the translation was revised and the new translation is as follows:

$(FORALL(q\_1 : CON, q\_2 : CON) :$
$((set\{CON;\ \}!contains?(cons, q\_1)AND$
$set\{CON;\ \}!contains?(cons, q\_2))$
$AND(q\_1/ = q\_2)) =>$
$(set\{PHONE;\ \}!intersection(q\_1, q\_2) =$
$set\{PHONE;\ \}!empty)))$

which is equivalent to:

$(FORALL(q\_1 : CON, q\_2 : CON) :$
$(set\{CON;\ \}!contains?(cons, q\_1)AND$
$set\{CON;\ \}!contains?(cons, q\_2)) =>$
$((q\_1/ = q\_2) =>$
$(set\{PHONE;\ \}!intersection(q\_1, q\_2) =$
$set\{PHONE;\ \}!empty)))$

However, this generated SAL cannot be simulated by SAL simulator due to ran out of memory. Several experiments have been tried, such as deleting one by one the invariant, deleting both the invariants, but all of these did not work. After the size of PHONE was changed into 1, default is three; this SAL can be simulated successfully.

*3) Experiment with One Increment Specification:* This specification is obtained from [12, p. 94]. The specification includes a user-defined function to add one to other natural numbers. This function, *f*, needs one argument whose type is natural number and returns a result which is also a natural number. Here is the full specification:

$f : \mathbb{N} \to \mathbb{N}$

---

$\forall\, n : \mathbb{N} \bullet f(n) = n + 1$

*State*

$number : \mathbb{N}$

*Init*

$State'$

---

$number' = 0$

*Increment*

$number? : \mathbb{N}$
$result! : \mathbb{N}$

---

$result! = f(number?)$

Z2SAL can translate this specification into its SAL. However, the generated SAL cannot be run by SAL simulator due to the existing of empty initial set. Based on the evaluation, this error might be occurred since the invariant could yield false. After modified the specification as follows, it can be simulated by SAL simulator.

$max : \mathbb{N}$
$f : \mathbb{N} \to \mathbb{N}$

---

$max = 3$
$\forall\, n : \mathbb{N} \bullet (n > max \Rightarrow f(n) = n)$
$(n <= max \Rightarrow f(n) = n + 1)$

*State*

$number : \mathbb{N}$

*Init*

$State'$

---

$number' = 0$

 978-1-4799-8075-8/14/$31.00 

*Increment*

$number? : \mathbb{N}$
$result! : \mathbb{N}$
$\Xi State$

---

$(number? > max \Rightarrow result! = number?)$
$(number? <= max \Rightarrow result! = f(number?))$

*4) Experiment with Inverse Relation in Hotel Specification:* This specification is almost the same as specification in Experiment 1. The difference is in this specification one operational schema is added. The schema is as follows:

*WhoWhichRoom*

$\Xi Hotel$
$room? : HOTELROOM$
$guest! : GUEST$
$reply! : RESPONSE$

---

$\forall g : GUEST \bullet ((g, room?) \in occupies) \Rightarrow$
$guest! = g$

We then rewrote the predicate by using inverse relational operator as follows:

$$(room?, guest!) \in occupies^{\sim}$$

and it works. It means that Z2SAL has also supported inverse relational operation. However, from our experiment using this operator on function instead of relation as above example, there was a problem, Z2SAL cannot translate the specification.

*5) Results and Discussion:* For the first experiment, the specification contains non-constructive predicates. In order to enable the translation, those predicates are rewritten in another way around which is more constructive.

For the second experiment, the generic constant is deleted and any occurrence of its predicate in other schemas is replaced appropriately, so does with theta operator, and schema references. We do the same for another meaning of delta schema, change it into the ordinary delta schema and add manually other variables or predicates which are not included in the state schema.

For the third experiment, based on our investigation, it is identified that the invariant is sometimes false since the function is not really total. Z2SAL defines the maximum number for the natural number used here which is 4. This maximum number is one above the maximum number specified in the Z specification. Thus, for this maximum natural number, it will not be mapped to any number and it gives false. In order to avoid the problem, the specification should be modified to make it never reach the number more than its Z defined maximum one which is 3. If such a number is reached then it returns the maximum number defined by Z2SAL in generated SAL. Otherwise, the output is the same as this number.

For the last experiment, as mention above, it seems Z2SAL has supported inverse relational operation, but not for all types of variables. For example, variables formed by functions are not translated at this moment.

## II. Conclusion and Future Works

As stated previously, the aim of this paper is to report experiences during conducting several experiments with the Z2SAL tool. This study has shown that Z2SAL is rich enough with tags accepted by Z LaTeX styles and supports many parts of Z, such as set, sequence (although needs further testing), relations and functions, several mathematical operator, horizontal schema writing as well as vertical one, also accepts more than one Z package styles. In these experiments, **oz** and **zed** package styles were used. Therefore, a specification which contains Z language is written by using LaTeX styles either **oz** or **zed** package styles. For the translation strategies of those Z language into SAL language, could be read on [5]. We cannot describe it here due to the page limitation. Based on this finding, we could declare that the Z2SAL is not complete since it has not supported all parts of Z language. Due to this incompleteness, our research has aims to explore parts of Z that has not been translated by Z2SAL and to suggest those parts to be able to translation by Z2SAL.

The second major finding is that if Z2SAL does not support such tags or definition of Z language, we could rewrite them by using their similar meaning. This might be applied to schema calculus which is not supported yet by Z2SAL, but we could rewrite them by using a direct single schema definition as usual.

Third, it seems that some errors found are merely a consistency preservation of Z2SAL and SAL model checker, such that Z2SAL avoids to translate a non-constructive style of Z specification which is appropriate with SAL's common expressions writing, the constructive style. We have also found that sometimes the unable to run by SAL simulator is a technical deficiency, for example the size of memory on the used machine. This issue relates to the state space explosion problem in model checking. We have taken into account the issue of ran out of memory by investigating the use of abstraction as a means to enable model checking can verify arbitrary Z specifications.

Fourth, although we have not yet proved it formally due to there is no a common semantic module for Z and SAL, we think the Z2SAL is sound. We could claim that based on our experiments, for almost all translations of Z language into SAL language, both of them have equivalent meanings. However, some awareness of the differences between the Z language and SAL language should be taken into consideration for that soundness. For example, Z language supports infinite types in contrast to SAL language. Thus, such as $\mathbb{N}$, Z2SAL must translate the infinite $\mathbb{N}$ of Z specification into the finite of that type which can be recognized by SAL.

 978-1-4799-8075-8/14/$31.00 

## ACKNOWLEDGMENT

The first author would like to thank John Derrick, Siobhan North and Anthony Simons since this paper is initially based on their Z2SAL, and to Graeme Smith and Kirsten Winter for inspiring us with the use of abstraction in model checking Z specification. The first author would also like to thank ISIHEMORA the Republic of Indonesia for its financial support.

## REFERENCES

[1] Potter, B., Till, D., and Sinclair, J.: An introduction to formal specification and Z. Prentice Hall PTR (1996)
[2] West, M.M.: Issues in Validation and Executability of Formal Specifications in the Z Notation. Thesis of University of Leeds (2002)
[3] Jackson, D.: Abstract model checking of infinite specifications. FME'94: Industrial Benefit of Formal Methods. Springer, 519–531 (1994)
[4] Malik, P., Groves, L. and Lenihan, C.: Translating z to alloy. ASM, Alloy, b and Z. Springer, 377–390 (2010)
[5] Derrick, J., North, S., and Simons, A.J.H.: Z2SAL: a translation-based model checker for Z. Formal aspects of computing. Springer, 23 1, 43–71 (2011)
[6] Smith, G. and Wildman, L.: Model checking Z specifications using SAL. ZB 2005: Formal Specification and Development in Z and B. Springer, 85–103 (2005)
[7] Derrick, J., North, S., and Simons, A.J.H.: Issues in implementing a model checker for Z. Formal Methods and Software Engineering. Springer, 678–696 (2006)
[8] Simons, AJH: The Z2SAL User Guide. Accessed from http://staffwww.dcs.shef.ac.uk/people/A.Simons/z2sal/userguide.html (2012)
[9] De Moura, L., Owre, S. and Shankar, N.: The SAL language manual. Computer Science Laboratory, SRI International, Menlo Park, CA, Tech. Rep. CSL-01-01 (2003)
[10] Bensalem, S., Lakhnech, Y. and Owre, S.: Computing abstractions of infinite state systems compositionally and automatically. Computer Aided Verification. Springer, 319–331 (1998)
[11] de Moura, L.: SAL: tutorial. Computer science laboratory, SRI International (2004)
[12] Rann, D. and Turner, J. and Whitworth, J.: Z: a Beginner's Guide. CRC Press (1994)
[13] Hayes, I. and Flinn, B.: Specification case studies. Prentice-Hall International London (1987)
[14] Spivey, J.M.: The Z notation. Prentice Hall New York (1989)
[15] Woodcock, J. and Davies, J.: Using Z: specification, refinement, and proof. Prentice-Hall, Inc. (1996)

 978-1-4799-8075-8/14/$31.00 