Checking Object-Z Formal Specification with Z/EVES automatically

Qian Cheng¹ and Zhicheng Wen¹*
¹College of computer science, Jiangxi University of engineering, Xinyu, Jiangxi Province, 338000, China
*Corresponding author e-mail: zcwen@mail.shu.edu.cn

Abstract: Formal specifications can only be used when matching or non-colliding. In the conventional programming language, program integrity is checked at runtime. However, formal specification descriptions cannot be implemented in general. The grammar analysis and semantic checking of a tool sometimes invalidate conformance checks. Therefore, it is difficult to verify the consistency of formalization norms. In this paper, we propose a method to generate theorem proof for Object-Z norms. The purpose is to obtain clear confidence by verifying the consistency of norms. Finally, we use the theorem prover Z/EVES to analyze and verify theorems.

1. Introduction
Object-Z[1] is an extension for Z[2] and can accurately describe a large norm. So it can easily get the behavior and nature of the Object-Z norms. The verification for formalization can provide confidence in the formalized criteria, including theorem proof and model checking. There are several work in formal verification[3]. Attributes with this specification can be reasoned about. In paper [3], there are a reasoning rule and inference method for the inference of the Object-Z. A type of attribute represents a formal order: A:: d ∣ Ψ ├ Φ 
wardrobe,
A is a class name, d is a list of statements, and Ψ and Φ is a list of predicates. The d is a given statement and at least one predicate in the predicate are true in a class, when the order is valid, and that is, the attribute of the statement is true. Predicates in the formula Ψ and Φ can be accessed only when it is defined by the variable and the number of statements in d.

There are two parts to validate formal specifications. First, we generate an association theorem representing the specific properties of a specification; Second, these theorems is verified by an existing theorem prover, such as Z/EVES[4,5] (i.e., proving the theorems). This paper presents an method to use several theorems(Theorem) to verify the consistency of a formal specification.

2. Non-confliction of Object-Z specification
If invariance is uniform, there is at least one state in the state space. Operations can transform one state into another[6-8]. Therefore, state space or initial state may be non-conflictive.

2.1 Existence of state space
According to Object-Z syntax, global or local definitions can be regarded as the context information through context recognition. However, all variables defined in the state mode are not regarded as contexts. Generally speaking, we assume that the context is consistent. The variables defined in the state mode are represented by the x of X only. Using Invariance(x) to represent state invariance, OP is used to represent operations.

If invariance does not conflict, at least there must be state in the state space. From the context, we
should:

Context \(\vdash \exists x: X \cdot \text{Invariance}(x)\).
If there is no such reasoning relation, invariance is conflict. Therefore, we have a duty to prove:

**Theorem 1:** Context \(\vdash \exists x: X \cdot \text{Invariance}(x)\)

### 2.2 Existence of initial state

If one normative coincides, this initial state exists and has the denaturizing of the state. We have:

Context \(\vdash \text{Invariance(INIT)}\).
There should be a state satisfying the initial state in the state space. We have:

**Theorem 2:** Context \(\vdash \exists x: X \cdot \text{INIT}(x) \land \text{Invariance}(x)\)

From the above class Queue\([T]\), Invariance\((x)\) and Context are the same description as above and the INIT\(\equiv \text{items}=< >\). We have:

\[
\begin{array}{l}
\text{Theorem: } \vdash \\
\exists \text{item: seq } T; \text{size:N}; \text{size}\leq \text{maxlength} \\
\wedge \text{item}=<> \wedge \text{size}=\#\text{item}
\end{array}
\]

Otherwise, this initial state is unclear if the number of elements in the initial state schema is greater than 10.

### 2.3 Feasibility of an operation

If an operation is possible, two states (pre states, post states) of invariance can be satisfied accordingly. Invariance is included in the hidden expression operation. Therefore, we can generate theorem proofs for each operation:

**Theorem 3:** Context \(\vdash \exists x, x': X; y: Y \cdot \text{OP}(x, x', y) \land \text{Invariance}(x') \land \text{Invariance}(x)\)

Here, an unpredictable part can be represented by \(\text{OP}(x, x', y)\). \(x\) represents the pre-variable and \(x'\) represents the post-variable. For example, let \(\text{OP}\equiv \text{Join}, y\equiv \text{item}\) and \(Y\equiv T\):

\[
\begin{array}{l}
\text{maxlength: } \text{Lengtht}; \text{Lengtht}=1..10 \\
\text{Theorem: } \vdash \\
\exists \text{item, item': seq } T; \text{size, size':N}; \text{item'}: T; \\
\text{size}=\#\text{item}; \text{size}'\leq \text{maxlength}; \text{size}<= \text{maxlength} \\
\wedge \text{size}\leq \text{maxlength}; \text{size}'=\#\text{item'} \wedge \text{item'}=\text{item}'<>\text{item}?
\end{array}
\]

3. **Verifying specification (semi-)automatically**

From the previous chapter, you can generate a theorem about the subject Z standard. In this portion, we are validated by the theorem prover Z/EVES based on the first theory to generate the generated method based on the above theory. Z/EVES has a common theorem proving function, which can validate the non Z theorem, but the Z/EVES syntax requires code.

Examples of theorems described in the previous section are verified by Z/EVES. The introduction of this day introduces a theorem to (semi-)automatically generate a theorem from the Object-Z standard, and these theorems can be proved directly in the Z/EVES and can be made as shown in Fig. 1. First, through the Object-Z norm editor developed in this group, you can check the Object-Z standard grammar and bring it to the save file. Next, the preservation file (*.lax) can introduce the certification obligation of our group development. Finally, the theorem(*.zev) is introduced into Z/EVES and proved directly. All these saved files have a format of rubber.

[Fig 1 Verifying Object-Z specification](image)

3.1 **Generating a theorem proof**

This vessel can (semi-)automatically generate relevant theorems: (1) it can automatically generate
congruence of theorems; and (2) it can generate a type of child type of semi-automatic theorem. In this case we can manually generate theorems. Latex format allows you to automatically code. Z/EVES can identify these formats. These can be imported into the generator, which generates the certification obligation and displays it in the latex format shown in Figure 2.

4. Verifying Theorems in Z/EVES automatically
Within one proofs proof, the user starts with a certain goal and does not know one proof. If you are able to explore the proof space with various proofing instructions, and if you're lucky, the goal is proven. The proof of the plan creates an unofficial certificate that specifies the target over time before the user proves using the Z/EVES. In general, we prove to verify a certification obligation using a plan certificate. That is, it requires artificial intervention. We need to do some work before proving.

Examples of theorems described in the previous section are verified by Z/EVES. We explain how to validate it in the theorem 3 example.

4.1 Editing a theorem proof
First, the proof duty is edited in the theorem prover Z/EVES. When we run program Z/EVES, we display the main window (standard window), as shown in Figure 3. Of course, the standard window is blank at start time. The "new paragraph" is selected from the edit menu, and an editing window shown in Fig. 3 is provided. Click the edit area mouse to enter the editing content. After editing the content,
select "finish" from the file menu to bring the standard window. In this way we can edit the certification obligations, which can be verified.

4.2 Analyzing a theorem proof

In Z/EVES, using the plan certificate to verify the proof’s duty, maxlength is set to 5, a and b are T. Theorem 3 feasibilityofJoin was edited (theorem is keyword). All content is shown in Figure 4.

After editing the content, we prove the duty of certification. The sign "Y" in the grammar means that there is no grammar or type error. The "N" sign in the "proof" field must prove this theorem.

Fig 4 Specification window

In Fig. 4, since the "Y" is held in the state field of the grammar, and there is no grammar or type error, the "proof" column holds "N". We verify this theorem feasibilityofJoin in section 4.3.

4.3 Verifying a theorem proof

Three functions of the certificate: as shown in Fig.5, such as inspection and modification certificates, mutual structural proof and display of certificates. On this day, the possibility of the theorem feasibilityofJoin is verified in Z/EVES. If we select the theorems feasibilityofJoin that we have connected and choose "display certification" from the pop-up menu of figure 4, we will prove to be shown in Figure 5.

Many simple theorems proved by proof. If the certificate command is an off track, you can usually use a combination of commands reduce, rewrite, simplify, rearrange, prenex, and equality substitute. The command. Instance command is applied to validate the existence of quantifiers that require some examples. You can edit commands by selecting a new command from the command menu.

Verifying these theorems is demonstrated for feasibilityofJoin. The steps are shown in Fig. 5. When these instructions are gradually executed, the "true" symbol is displayed at the official window at the end, it is verified, and the execution state field stores "Y" for each command. This is an example of item? == a and items == <b>.

5. Conclusion

Formal norms can only be used when matching. However, there is a possibility that norms do not match. The formalization Standard cannot be executed generally. Grammar analysis and typing are sometimes inconsistent. In this paper, we examine the method of systematically verifying the normative consistency of Object-Z that has characteristics for the opponent. So it requires that you do not clash. Next, it is necessary to verify that some operators and functions are applied in domain. In addition, we are particularly considering the issue of reusing duties. Finally, we analyze and verify theorems using the theorem prover Z/EVES.
Acknowledgements
Qian Cheng(1982.4-), female, Jingshan county, HuBei Province, Han, Master’s degree, associate professor, majoring in: electronic engineering. Corresponding author: Zhicheng Wen (1972.11-), male, Dong’an county, Hunan province, Han, Ph.D., professor, majoring in: network security, trusted software.

References
[1] Smith, G, The Object-Z Specification Language[M], Advances in Formal Methods, Kluwer Academic Publishers, 2000.
[2] J. C. P. Woodcock and J. Davies. Using Z: Specification, Proof and Refinement[M]. Prentice Hall International Series in Computer Science, 1996.
[3] HOU Gang, ZHOU Kuan-Jiu, CHANG Jun-Wang, WANG Jie, LI Ming-Chu. Software Formal Modeling and Verification Method Based on Time STM[J]. Journal of Software, 2015, 26(2):223–238.
[4] I. Meisels and Mark Saaltink, The Z/EVES 2.0 reference manual[M]. Technical Report TR-99-5493-03e, ORA Canada, Canada, 1999
[5] Leo Freitas, Jim Woodcock and Yichi Zhang, Verifying the CICS File Control API with Z/Eves: An experiment in the verified software repository[J], Science of Computer Programming, 2009, 74(4):197-218
[6] Augusto Ribeiro and Peter Gorm Larsen. Theorem proof Generation and Discharging for Recursive Definitions in VDM[J], Formal Methods and Software Engineering, Lecture Notes in Computer Science, 2010, volume 6447: 40-55
[7] Zhang Yang, Duan Fu. Formal transformation and verification of UML model based on specification Z[J]. computer engineering and design, 2013 34(6): 2031-2035.
[8] Dong Yu, Gao Xue-juan. Method for generating Formal Interlocking Software Model Based on Scenario[J]. Computer science, 2015, 42(1): 193-195,226