B-035

Provably Correct Translation from OTS/CafeOBJ Specifications to Java Programs

Jittisak Senachak† Takahiro Seino† Kazuhiro Ogata‡ Kokichi Futatsugi†

1. Introduction

As the safety of software systems become more important recently, formal methods can help it to specify and verify such properties before implementing them. In practical, formal specifications and implementations are developed separately, and the difference causes the system fail. In this research, we have built a tool to generate programs that meet the specifications from the code of specification. Furthermore, we have also verified that for any OTS/CafeOBJ specification, a Java program which can be generated by the translator is an implementation of specification by writing proof score[1] in CafeOBJ. That can guarantee the correctness of the translator and the generating code meet the specification. Our approach is as follows:

1. We have designed notations (in CafeOBJ representation) of both in OTS/CafeOBJ specifications and Java programs called "Meta-OTS/CafeOBJ" and "Meta-Java/CafeOBJ" respectively.
2. We have written a specification of the translation rules in CafeOBJ, which takes Meta-OTS/CafeOBJ specifications and generates Meta-Java/CafeOBJ programs.
3. We have specified and verified a Meta-Java/CafeOBJ program by writing proof scores in CafeOBJ.

2. Preliminaries

2.1 OTS/CafeOBJ Specification

OTS (Observational Transition System) [2] is one of transition models defining the behavior of a system on the changes of observation values (that is, system's attributes) after applying a sequence of transitions at any state of system. Let call an OTS specification written in CafeOBJ as *OTS/CafeOBJ* [2]. One of benefit of writting a specification in this style is that we can specify a system without knowing the detail of system implementation [3].

2.2 Java Class Declaration

As specified in [5], a Java class defines:

- Field declarations describe class variables which can be primitive or instance variables.
- Method declarations denote code that may be invoked by method invocation expressions. An instance method is invoked with respect to some particular object that is an instance of the class type.
- Static initializers are blocks of executable code that may be used to help to initialize a class.
- Constructors are similar to methods, but cannot be invoked directly by a method call; they are used to initialize new class instances.

An instance of class, we call it as *an object*. Like other object-oriented programming language, a Java object can hide the information (details of implementation) inside the class.

†Japan Advanced Institute of Science and Technology
‡NEC software Hokiriku, Ltd.

3. Translation Approach

3.1 How to specify the source and target languages

For an OTS/CafeOBJ specification, we have designed a meta-language called "Meta-OTS/CafeOBJ" based on OTS model. In an OTS model($anOTS$), it consists 3 sets of observations(O), transitions(T) and initial state(I). A Meta-OTS/CafeOBJ notation will represent set of observation declarations as O, set of action declarations with relevant equations as T and set of initial values as I. In each action, there is an effective condition and list of effects showing the changes of observation values when action takes effect. Expressions and data types are all defined on the CafeOBJ built-in INT and BOOL modules[1].

In the counterpart, a Java program, we also have "Meta-Java/CafeOBJ", which represents a program retaining the structure of an OTS model. It has 3 sets of variables(V) as O, methods(M) as T and constructor(C) as I. Inside the methods, there are 5 kinds of statements: value-assignment, object-instantiation, if-then-else, method invocation, and method return. Like "Meta-OTS/CafeOBJ", the same structure of expressions and data types will be applied.

3.2 How to specify the translation rules

In this part, we will show the translation rules from a Meta-OTS/CafeOBJ module to a "Meta-Java/CafeOBJ" class. It can be built based on the structure of the specification as follows:

Category 1 *A Meta-OTS/CafeOBJ module (existing one hidden sort)*

A hidden sort name translates into the class name.

> A variable on a hidden sort represents an internal state of a system. It is corresponding to 'this' operator[5] in Java.

Category 2 *Set of observation declarations*

A signature of an observation translates into a Java private variable declaration and variable access methods to retrieve(public) and assign(private) the observation value. All parameters of methods can be generated from the arity of each observation. For retrieval method, the return type can be a corresponding type to coarity of an observation.

Category 3 *Set of action declarations*

A signature of an action translates into a Java public void method called "action method". All parameters of methods can be generated from the arity of each action and all observations for referencing inside its method.

Category 4 *Set of initial values*

An equation for setting an initial observation value translates to be a value-assignment statement in the class constructor

Category 5 *A (conditional/unconditional) equation in axiom declaration*

An equation for an action translates to a statement of the corresponding action method; for example,

```
cq balance(withdraw(A:Account, N:Int)) =
   balance(A) - N
   if(balance(A) > N) .
```

translates to

```
void withdraw(int N){
  if(balance() > N)
    setbalance(balance() - N);
  //...for other withdrawal equations ...
}
```

4. Correctness of Translator

The correctness of translator can be shown by specifying and verifying the translation rules. We can say that if there exists a simulation (refinement) relation between an arbitrary Java generated code and its specification, it can be guaranteed that the translation rules are sounded. Such relation can be considered as for any Java context(precisely, state and values of referencing symbols) the evaluation on that should give the same result as on corresponding OTS context. As shown in Fig. 1

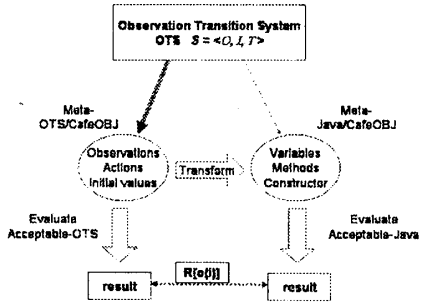

Figure 1: Guarantee of correctness

4.1 How to specify the simulation relation

As stated above, we can show the simulation relation on the transitions of action and the evaluations of observation values in both of specification and implementation. For a Meta-OTS/CafeOBJ module, there are 2 operators. One, "observe", is to observe a value of any observation (with parameters as index) at any state. Other one, "exec-ots", is to execute a sequence of execution at any state. It should specify in the both cases of the effective condition is satisfied and not satisfied (just ignored). In addition, we also need an evaluation operation of expressions, "evaluate", on a state and some referenced symbol. In the same phenomena, the simulation of a Meta-Java/CafeOBJ program also has "value", "exec-java" and "jevaluate" operations.

The proof of existing a simulation, it can be shown the predicate as below. We will prove by induction on any sequence of actions for a general OTS/CafeOBJ module.

$$\boxed{\begin{array}{c}\texttt{exec-OTS}(AnOTS, AL, OC) \\ \texttt{R}[o(i)] \\ \texttt{exec-Java}(\texttt{tr}(AnOTS), \texttt{tr}(AL), \texttt{tr}(OC))\end{array}}$$

where

- $AnOTS$: An OTS/CafeOBJ specification
- AL : A sequence of actions with action parameter
- OC : OTS context consisting state S and symbol-reference-list
- R : Equality function of the evaluation of $o(i)$ on the OTS context and Java Context
- $o(i)$: An observation o with indexing i, where o is in set of observation declarations in $AnOTS$
- tr : Translation functions

5. Related Work

Previously, there is a research [4] concerning only the signature part of CafeOBJ specification. Its tool can generate Java template code of classes leaving for programmer to full-filled the implementation inside a method. For another one [3], it is a tool for discovering an algebraic specification from a Java class using the concept of object-oriented. In this research, we are considering all code in OTS/CafeOBJ specification and also showing that this translator can be proved.

6. Conclusion

In this work, we have made, specified, and verified a collection of translation rules. The proof shows that the generated code is an implementation of its specification (in CafeOBJ representation). Also, we have already implemented a translator, and experience it with some simple concrete specifications, such as, banking account, cruise controller (as a finite state machine).

In this paper, the translator can accept only the simple data type; such as, integer and boolean. We plan to study further more complicated abstract data type which can be defined in CafeOBJ. Also, the translation among several modules which has the inheritance relation is a good research topic for distributed systems.

References

[1] CafeOBJ official homepage:
`http://www.ldl.jaist.ac.jp/cafeobj/`

[2] K. Ogata and K. Futatsugi: Proof Scores in the OTS/CafeOBJ Method. In *Proc. of the 6th IFIP WG6.1 International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS 2003)*, Springer, pp.170-184.

[3] J. Henkel and A. Diwan: A Tool for Writing and Debugging Algebraic Specifications. In *Proc. of the 26th International Conference on Software Engineering (ICSE'04)*, IEEE, pp.449-458.

[4] C.Doungsa-ard and T. Suwannasart: An Automatic Approach to Transform CafeOBJ Specifications to Java Template Code. In *Proc. of the International Conference on Software Engineering Research and Practice (SERP'03)*, pp.171-176.

[5] J. Gosling, B. Joy and G. Steele: The Java(TM) Language Specication. Sun Microsystems, 1996.