Verifications with Proof Scores in CafeOBJ

Kokichi Futatsugi[1], Joseph A. Goguen[2], and Kazuhiro Ogata[1,3]

[1] Japan Advanced Institute of Science and Technology
[2] University of California at San Diego
[3] NEC Software Hokuriku, Ltd.

Keywords: system validation and verification, specification languages and methods, systems and techniques for reasoning about specifications, proof by reduction/term-rewriting, proof score, CafeOBJ

Motivation and History of Proof Scores

Although a success of model checking has improved technologies for system verification, they are still limited only to system models which can be translated to finite transition systems. On the other hand it seems that fully automatic theorem provers will be unable to do really difficult proofs in the near future (if ever). Given this situation, we believe that successful use of current technologies requires a careful blend of user and machine capabilities, and of rigor and intuition, to make optimal use of the respective abilities of humans and computers, with computers doing the tedious formal calculations, and humans doing the high level planning. This principle is also supported by the observation that fully automated proofs are not necessary good for empowering human beings to perceive the logical structure in the reality which has vital importance in designing truly reliable systems.

The dominant modern logical system is first order predicate logic, but advances in computer science have spawned a huge array of new logics, e.g., for database systems, knowledge representation, and the semantic web; these include variants of propositional logic, modal logic, intuitionistic logic, higher order logic, and equational logic, among many others. For maximum flexibility, users should be able to choose whatever logical system they need for their proof, and a mechanical theorem prover should provide a basic engine for applying rules of inference by term rewriting. In this way, a wide variety of logical systems can be implemented by supplying appropriate definitions to the rewrite engine. For example, [1] use UNITY[2] as a logical basis for verifying real time concurrent systems using CafeOBJ system[3].

This approach requires the user to construct proof scores, which are instructions such that when executed (or "played"), if everything evaluates as expected, then the desired theorem is proved. A proof score is executed by applying proof measures, which progressively transform formulae in a language of goals into expressions which can be directly executed.

It is particularly convenient to implement this approach using language that support equational specification and term rewriting. CafeOBJ[3, 4] and Maude[5] are the major languages of such kind with usable implementations. In addition, the advanced module facilities provided in these languages which are inherited from OBJ language can be used to express the structure of proofs.

Writing proof scores in algebraic specification languages has been promoted by researchers of the OBJ family of languages[6, 7]. The proof score approach was influenced by the theory of institutions[8], and although its motivation resembles that the Edinburgh Logical Framework[9] and Paulson's Isabelle system[10] in avoiding commitment to any particular logical system, it differs significantly in its use of equational logic and term rewriting as basis. Maude can be used as a meta-tool for theorem proving in a variety of logical systems, but differs in relying on rewriting logic.

Proof Scores in CafeOBJ

CafeOBJ is a general formal specification language with reduction and resolution engines, and can be used at least in the following three ways:

1 Write a formal specification and simulate or test the behaviors of the system specified by "executing" the specification using the reduction engine.
2 Write a formal specification and accompanying proof score for verifying the properties of the system by using only the reduction engine.
3 Write a formal specification and accompanying proof score for verifying the properties of the system by using both of reduction and resolution engines.

We concentrate on the approach 2 here. There are several reasons for this:

a This approach can provide an appropriate "blend of user and machine capabilities, and of rigor and intuition" which is mentioned above.
b Proof score execution only using reduction/term-rewriting can encourage transparent proof structures, and this support modular and readable proof scores. This feature is important because we want to use proof scores as effective documents which should be attached to important requirement/design specifications.
c Generally, executions of proof score with reduction engine are sufficiently efficient and support prompt interactions between users and systems.

The approach 3 sounds attractive but we still do not have enough experiences on this; one important achievement is doing (infinite state) software model checking by calculating fixpoint using the resolution engine[11]. In the context of verification with proof score, the resulution engine can be expected to suport limited systematic applications of backward rules (applications of equations from right to left).

In the course of research and development of CafeOBJ language and sytem, we have developed through several case studies a technique to write proof scores showing that a transition system have an invariant property (the property which holds for any reachable state).

Several important techniques are developed for writing proof scores in CafeOBJ. One important technique is for modular verification of the invariant property. This technique makes it possible to divide a formula stating an invariant property under discussion into reasonably small ones, and each of which is proved by writing and executing proof scores individually. This relieves the load to reduce logical formulas and can decrease the number of subcases into which the case is split in case analysis.

These techniques are successfully applied to modeling, specification and verification of distributed systems such as distributed mutual exclusion algorithms[12] and security protocols[13]. We named this method the OTS/CafeOBJ method. In OTS/CafeOBJ method, systems are modeled as Observational Transition Systems, or OTSs, which are described in CafeOBJ.

An OTS (observational transition system) is a transition system formulated as a set of operations of following two kinds:

– `observations` which model the observable values of a system; a state of the system can be identified as a set of values returned by the observations.
– `actions` which model the system's state changes; a state change modeled by an action `act` is formulated as the changes of values returned by the observations and is described by equations which defined the values after the action `act` based on the values before the action.

CafeOBJ description of OTSs can be regarded as restricted coherent hidden algebra[14]. We can verify that OTSs, which are models of systems, have properties by writing proof scores in CafeOBJ.

References

1. Ogata, K., Futatsugi, K.: Modeling and verification of distributed real-time systems based on cafeojb. In: Proceedings of the 16th Int'l Conference on Automated Software Engineering (ASE 2001), IEEE CS Press (Nov.2001) 185–192
2. Chandy, K.M., Misra, J.: Parallel Program Design: A Foundation. Addison-Wesley, Reading, MA (1988)
3. CafeOBJ: CafeOBJ web page. `http://www.ldl.jaist.ac.jp/cafeobj/` (2003)
4. Diaconescu, R., Futatsugi, K.: CafeOBJ report. AMAST Series in Computing, 6. World Scientific, Singapore (1998)
5. Maude: Maude web page. `http://maude.cs.uiuc.edu/` (2003)
6. Goguen, J., Malcolm, G., eds.: Software Engineering with OBJ: Algebraic Specification in Action. Kluwer Academic Publishers (2000)
7. Futatsugi, K., Nakagawa, A., Tamai, T., eds.: CAFE: An Industrial-Strength Algebraic Formal Method. Elsevier (2000)
8. Goguen, J., Burstall, R.: Institutions: Abstract model theory for specification and programming. Journal of the Association for Computing Machinery 39 (January 1992) 95–146
9. Harper, R., Honsell, F., Plotkin, G.: A framwork for defining logics. In: 2nd Symposium on Logic in Computer Scienc, IEEE CS Press (1987) 194–204
10. Paulson, L.C.: The foundation of a generic theorem prover. Technical Report 130, University of Cambridge, Computer Laboratory (1988)
11. Mori, A., Futatsugi, K.: Verifying behavioural specifications in cafeobj environment. In: World Congress on Formal Methods. Volume 1709 of LNCS., Springer (1999) 1625–1643
12. Ogata, K., Futatsugi, K.: Formally modeling and verifying Ricart&Agrawala distributed mutual exclusion algorithm. In: APAQS '01, IEEE CS Press (2001) 357–366
13. Ogata, K., Futatsugi, K.: Formal analysis of the iKP electronic payment protocols. In: ISSS 2002. Volume 2609 of LNCS., Springer (2003) 441–460
14. Diaconescu, R., Futatsugi, K.: Behavioural coherence in object-oriented algebraic specification. Journal of Universal Computer Science 6 (2000) 74–96