CafeInMaude: A CafeOBJ Interpreter in Maude

Adrián Riesco[1(✉)], Kazuhiro Ogata[2,3], and Kokichi Futatsugi[3]

[1] Facultad de Informática, Universidad Complutense de Madrid, Madrid, Spain
ariesco@fdi.ucm.es

[2] School of Information Science, JAIST, Nomi, Japan

[3] Research Center for Software Verification, JAIST, Nomi, Japan
{ogata,futatsugi}@jaist.ac.jp

Abstract. We present in this paper CafeInMaude, an interpreter for non-behavioral CafeOBJ specifications. The interpreter has been implemented in Maude. This alternative implementation combines CafeOBJ specification and theorem proving capabilities with efficient and extensible Maude commands and tools. Hence, it makes it possible to use both CafeOBJ proof scores and reduction commands and Maude model checking, narrowing, or theorem proving capabilities with the same tool.

Keywords: CafeOBJ · Full maude · Model checking · Theorem proving

1 Introduction

CafeOBJ [4] is a language for writing formal specifications for a wide variety of software and/or hardware systems, and verifying properties of them. CafeOBJ implements equational logic by rewriting and can be used as a powerful interactive theorem proving system. Specifiers can write proof scores [5] also in CafeOBJ and perform proofs by executing these proof scores. CafeOBJ, implemented in Lisp, provides several features to ease the specification of systems. These features include a flexible mix-fix syntax, powerful and clear typing system with ordered sorts, parameterized modules and views for instantiating the parameters, module expressions, operators for defining terms, equations for defining the (possibly conditional) equalities between terms, and (possibly conditional) transitions for specifying how a system evolves, among others.

CafeOBJ and Maude [1] are sister languages of the OBJ family. Maude modules are executable rewriting logic specifications and its C++ implementation shares many features with CafeOBJ. However, while the CafeOBJ community has focused on proofs via proof scores, the Maude community has focused on (i) verification of properties via model checking and exhaustive search, efficiently implemented in Maude, and (ii) tools implemented in Maude itself, thanks to the reflective capabilities of Maude [1], which allows users to extend Maude with new

Research partially supported by Japanese project Kakenhi 23220002, MICINN Spanish project *StrongSoft* (TIN2012-39391-C04-04), Comunidad de Madrid project N-Greens Software-CM (S2013/ICE-2731), and UCM-Santander grant GR3/14.

syntax and commands.[1] Among these tools we have the Maude Formal Environment (MFE) [3], which includes tools for proving termination, confluence, and coherence, the Constructor-based Inductive Theorem Prover (CITP) [6], a tool for proving inductive properties of systems specified with constructor-based logics, and the declarative debugger and test-case generator [8].

Taking into account the similarities between both languages, using Maude reflective capabilities, and using the translation in [7] we have implemented CafeInMaude, a CafeOBJ interpreter implemented in Maude. It defines both the parsing mechanisms required to introduce CafeOBJ specifications into the Maude database (hence allowing other Maude tools to be used with these specifications) and the execution strategies for faithfully executing proof scores. In this way it is possible to combine both Maude and CafeOBJ features. For example, it is now possible to first prove the termination and confluence of a theory using the MFE and then prove properties on it by using proof scores.

Moreover, CafeInMaude provides a simple framework where new features and commands can be added and tested just by using Maude code (much more familiar to CafeOBJ programmers than the Lisp implementation of CafeOBJ or the C++ implementation of Maude). Once these new features become mature they can be added to the standard Lisp implementation. In fact, we have easily added new features like the `metadata` and the `nonexec` attributes, which are used to indicate extra information and to prevent the engine from using these statements when executing, respectively.

We outline in the next section how to use the tool, explaining how to parse and execute standard CafeOBJ specifications in Sect. 2.1, how to use Maude tools with these specifications in Sect. 2.2, and how to extend CafeOBJ syntax in Sect. 2.3, while the limitations of the tool are presented in Sect. 2.4. Section 3 concludes and outlines some lines of future work. The tool and several case studies are available at https://github.com/ariesco/CafeInMaude.

2 Using CafeInMaude

We present in this section how to execute CafeOBJ specifications, how to combine them with Maude tools, and how to extend CafeOBJ syntax and commands. Finally, we summarize the limitations of the tool with respect to the standard Lisp implementation.

2.1 Executing CafeOBJ Specifications

Although Maude requires some constraints to allow input/output with the user, being the most important of them that modules and commands must be enclosed in parentheses, standard CafeOBJ specifications can be easily loaded by CafeInMaude by using the script provided in the webpage above. It executes a Java pre-parser to the files and introduces these modified files into Maude.

[1] Actually, we extend Full Maude [1, Part II], an extension of Maude written in Maude itself that is used as base for any further extension.

CafeInMaude supports any non-behavioral CafeOBJ specification and open-close environment, including those using the search predicates available in the latest releases of CafeOBJ [9]. As an interesting case study, we have used our tool for the falsification of the NSPK protocol, which finds a state in which NSPK does not enjoy the authentication property by combining bounded model checking by means of `search` and induction by using proof scores. More details are available at http://www.jaist.ac.jp/~kokichi/class/i613-1312 and https://github.com/ariesco/CafeInMaude.

2.2 Using Maude Commands

CafeInMaude considers both CafeOBJ and Maude specifications first-order citizens, so it is possible to import Maude modules into CafeOBJ modules and vice versa. Hence, it is possible to import, for example, the `MODEL-CHECKER` module. Once this module is imported, it is possible to define, *using CafeOBJ syntax*, the type for the states and the atomic formulas to be used in our LTL formulas. Then, it is possible to verify whether some initial CafeOBJ configurations fulfill these formulas by using the predefined `modelCheck` predicate.

In the same way, any command can be applied to CafeOBJ specifications. Therefore, it is possible to use Maude commands such as narrowing [2], which allows the user to perform symbolic search starting with non-ground terms.

Finally, additional tools extending Maude can also be used. In order to use these tools the user must indicate that the grammar used by the tool is an extension of the CafeOBJ grammar: `CafeGrammar`. Using this idea, we have already integrated the Maude Formal Environment (MFE) [3], the Constructor-based Inductive Theorem Prover (CITP) [6], and the declarative debugger and test-case generator [8].

2.3 Extending CafeOBJ

Since the implementation of CafeInMaude depends on a grammar defined in Maude, it is easy for Maude and CafeOBJ programmers to extend it. Extensions just require two steps: (i) defining the type (if it does not exist yet) and the syntax of the new feature and (ii) define how to parse it, which includes its translation into Maude. The former is straightforward and just requires to define some operators in the grammar module, while the latter, though complex, is greatly eased by the parsing functions that we provide for parsing and translating any CafeOBJ term. Using these ideas we have added the `metadata`, `nonexec`, and `owise` attributes to add information, prevent from executing, and apply only when it is the only applicable equation, respectively.

2.4 Limitations

Since CafeInMaude is based in a translation from CafeOBJ to Maude, it is constrained by the constructions available in CafeOBJ that are not available in

Maude. These limitations mainly affect the modules with loose semantics: in Maude these modules cannot be parameterized and can only be imported by other modules with loose semantics, and only in `including` mode (indicating that junk and confusion are allowed). We deal with these restriction in a conservative way: if the user allows a non-strict translation (which is enough for the tools currently integrated in CafeInMaude), they are translated as modules with tight semantics, while a warning message indicates the changes performed in the modules; otherwise, the translation fails.

Moreover, Maude does not allow modules with free parameters to be used to instantiate parameterized modules. In this case the tool cannot translate the module and it displays an error message.

3 Concluding Remarks and Ongoing Work

We have presented in this paper CafeInMaude, a tool to introduce CafeOBJ specifications into the Maude database. This tool provides an alternative implementation of CafeOBJ that allows us to use Maude modules and commands with CafeOBJ specifications, improves the performance of some of its commands, and eases the task of connecting CafeOBJ specifications with tools implemented on top of Full Maude. As future work we plan to use the narrowing techniques implemented in Maude [2] to analyze protocols previously defined in CafeOBJ.

References

1. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.: Reflection, metalevel computation, and strategies. In: Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C. (eds.) All About Maude - A High-Performance Logical Framework. LNCS, vol. 4350, pp. 419–458. Springer, Heidelberg (2007)
2. Clavel, M., Durán, F., Escobar, S., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer,J., Talcott, C.: Maude Manual (Version 2.7), March 2015. http://maude.cs.uiuc.edu/maude2-manual
3. Durán, F., Rocha, C., Álvarez, J.M.: Towards a maude formal environment. In: Agha, G., Danvy, O., Meseguer, J. (eds.) Formal Modeling: Actors, Open Systems, Biological Systems. LNCS, vol. 7000, pp. 329–351. Springer, Heidelberg (2011)
4. Futatsugi, K., Diaconescu, R.: CafeOBJ report. World Scientific, AMAST Series (1998)
5. Futatsugi, K., Gâinâ, D., Ogata, K.: Principles of proof scores in CafeOBJ. Theor. Comput. Sci. **464**, 90–112 (2012)
6. Găină, D., Zhang, M., Chiba, Y., Arimoto, Y.: Constructor-based inductive theorem prover. In: Heckel, R., Milius, S. (eds.) CALCO 2013. LNCS, vol. 8089, pp. 328–333. Springer, Heidelberg (2013)
7. Riesco, A.: An integration of CafeOBJ into full maude. In: Escobar, S. (ed.) WRLA 2014. LNCS, vol. 8663, pp. 230–246. Springer, Heidelberg (2014)
8. Riesco, A., Verdejo, A., Martí-Oliet, N., Caballero, R.: Declarative debugging of rewriting logic specifications. JLAP **81**(7–8), 851–897 (2012)
9. Sawada, T., Futatsugi, K., Preining, N.: CafeOBJ Reference Manual (version 1.5.3), February 2015