A Rewriting Approach to the Design and Evolution of Object-Oriented Languages

Mark Hills Grigore Roşu

University of Illinois Urbana-Champaign

{mhills,grosu}@cs.uiuc.edu

Abstract

Rewriting logic semantics provides an environment for defining new and existing languages. These language definitions are formal and executable, providing language interpreters almost for free while also providing a framework for building analysis tools, such as type checkers, model checkers, and abstract interpreters. Large subsets of several existing object-oriented languages have been defined, while a new research language, KOOL, has been created as a platform for experimenting with language features and type systems. At the same time, new tools and formalisms aimed specifically at programming languages are being developed.

Categories and Subject Descriptors D.3.1 [*Programming Languages*]: Formal definitions, design, theory.

General Terms Languages, Design

Keywords Object-Oriented Languages, Language Semantics, Language Design, Rewriting Logic, Formal Analysis

1. Introduction

Object-oriented languages and design techniques have been highly successful, with OO languages now used for many important applications in academia and industry. Along with commonly-used static languages, such as Java and C++, there has been a resurgence in the use of both dynamic languages, like Python, and domain-specific languages, which are often built on top of existing OO languages. This has led to a flurry of research activity in the design and formal definition of object-oriented languages and in methods for analyzing programs.

Unfortunately, even as object-oriented languages are used in more and more critical applications, formal techniques for understanding these languages are still often post-hoc attempts to provide some formal meaning to already existing language implementations. This decoupling of language design from language semantics risks allowing features which seem straight-forward on paper, but are actually ambiguous or complex in practice, into the language. Decoupling also makes analysis more difficult, since the meaning of the language often becomes defined by either a large, potentially ambiguous written definition or an implementation, which may be a black box.

With this in mind, it seems highly desirable to provide support for formal definitions of languages during the process of language design and evolution. However, existing methodologies have limitations that can make it difficult to define the entire feature set of a complex language [14, 15], leading the language designer to delay the use of formal techniques, perhaps indefinitely. A concise, modular, and broadly usable framework, providing support for defining even complex language features, while providing tools for language interpretation and analysis, is critical to increasing the use of formal techniques during language design.

Initial work on this framework has been done within the Rewriting Logic Semantics project [13]. This includes work defining multiple languages and a new framework, the K framework, for defining languages using rewriting logic while leveraging notation and techniques designed specifically for programming languages [15]. The KOOL language, designed using this technique, is discussed in Section 2. Large portions of other object-oriented languages have also been defined and are the subject of Section 3. Discussions of future work, focused on improving the usability and reach of this technique, are presented in Section 4.

2. KOOL

KOOL is a concurrent, dynamic, object-oriented language, loosely inspired by, but not identical to, Smalltalk [5]. KOOL was designed using the K methodology [15] to meet three main goals: 1) the KOOL language design should specifically support experimentation with other language features, including optional and pluggable types [1]; 2) KOOL should provide support for analysis and verification, with the ability

OOPSLA'07, October 21–25, 2007, Montréal, Québec, Canada.
ACM 978-1-59593-865-7/07/0010.

to investigate how different language features and definition techniques impact analysis; 3) KOOL should be conceptually simpler than languages such as Java, making it suitable for instructional use in courses discussing language design and semantics. At the time of writing the core of the KOOL language has been developed, including support for analysis and experimentation with extensions [10, 9].

3. Other Languages

While KOOL is an experimental language, it is important to verify that the framework being developed can also support existing languages, including all their complexities. To this end, definitions for Java [6] and Beta [11] are being developed. The definition of Java was developed as part of the JavaFAN tool [4] for analyzing Java programs, but also supports running programs and experimenting with extensions to the language [2]. This definition, based on Java 1.4, supports most of the language, but at the time of writing still lacks support for features such as reflection and dynamic class loading. The Beta definition [7] supports a significant subset of the language, including concepts such as patterns, inner calls, and alternation, but is being redesigned using techniques developed by the authors over the last several years to support the entire language.

4. Future Work

Our work on programming languages has focused mainly on the use of rewriting logic [12], a logic of computation that supports concurrency. While we believe rewriting logic, in combination with engines such as Maude [3], is a compelling solution for defining and reasoning about languages [13], the generality of rewriting logic can sometimes lead to verbose, non-modular language definitions [15]. To exploit the strengths of rewriting logic, while ameliorating some of the weaknesses, we have developed K, a domain-specific variant of rewriting logic focused on defining programming languages [15]. K is specifically being designed to provide for concise, modular definitions of languages, written in an intuitive style. Initial versions of K have been used in the classroom and to define Java, Beta, and KOOL, as discussed in Sections 2 and 3. We plan to extend this work, defining languages and building libraries of language features, while using the feedback from this process to improve the K framework. Beyond this, to make the K framework and associated language definition techniques widely useful, we plan to develop tools to support language design, analysis, and execution. Related to this, some initial work has demonstrated the promise of translating K language definitions into executable form [8]; additional work will involve developing a user interface for working with K definitions and animation tools to visualize the workings of the language semantics (for instance, by showing execution traces or possible concurrent interleavings), with further work on translating K to Maude and various target languages.

References

[1] G. Bracha. Pluggable type systems. Revival of Dynamic Languages workshop at OOPSLA 2004, October 2004.

[2] F. Chen, M. Hills, and G. Roşu. A Rewrite Logic Approach to Semantic Definition, Design and Analysis of Object-Oriented Languages. Technical Report UIUCDCS-R-2006-2702, University of Illinois at Urbana-Champaign, 2006.

[3] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. Quesada. Maude: specification and programming in rewriting logic. *Theoretical Computer Science*, 285:187–243, 2002.

[4] A. Farzan, F. Chen, J. Meseguer, and G. Roşu. Formal Analysis of Java Programs in JavaFAN. In *Proceedings of CAV'04*, volume 3114 of *LNCS*, pages 501–505. Springer, 2004.

[5] A. Goldberg and D. Robson. *Smalltalk-80: the language and its implementation*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1983.

[6] J. Gosling, B. Joy, and G. Steele. *The Java Language Definition*. Addison-Wesley, 1996.

[7] M. Hills, T. B. Aktemur, and G. Roşu. An Executable Semantic Definition of the Beta Language using Rewriting Logic. Technical Report UIUCDCS-R-2005-2650, University of Illinois at Urbana-Champaign, 2005.

[8] M. Hills, T. F. Şerbănuţă, and G. Roşu. A rewrite framework for language definitions and for generation of efficient interpreters. In *Proceedings of WRLA'06*, ENTCS. Elsevier, 2007. To appear.

[9] M. Hills and G. Roşu. KOOL: An Application of Rewriting Logic to Language Prototyping and Analysis. In *Proceedings of RTA'07*, volume 4533 of *LNCS*, pages 246–256. Springer, 2007.

[10] M. Hills and G. Roşu. On Formal Analysis of OO Languages using Rewriting Logic: Designing for Performance. In *Proceedings of FMOODS'07*, volume 4468 of *LNCS*, pages 107–121. Springer, 2007.

[11] O. L. Madsen, B. Møller-Pedersen, and K. Nygaard. *Object-Oriented Programming in the BETA Programming Language*. Addison-Wesley, 1993.

[12] J. Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96(1):73–155, 1992.

[13] J. Meseguer and G. Roşu. The rewriting logic semantics project. *Theoretical Computer Science*, 373(3):213–237, 2007.

[14] P. D. Mosses. The varieties of programming language semantics. In D. Bjørner, M. Broy, and A. V. Zamulin, editors, *Ershov Memorial Conference*, volume 2244 of *LNCS*, pages 165–190. Springer, 2001.

[15] G. Roşu. K: a Rewrite Logic Framework for Language Design, Semantics, Analysis and Implementation. Technical Report UIUCDCS-R-2006-2802, University of Illinois at Urbana-Champaign, 2006.