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

Mark Hills and Grigore Roşu
{mhills, grosu}@cs.uiuc.edu

Department of Computer Science
University of Illinois at Urbana-Champaign

30 July 2007

Outline

Defining Languages

- Most languages defined informally first, formally later (if ever)
- Order seems backwards; would be nice to start with a formal definition
- Problem: often not very flexible, need something that can be useful during design

Why Formal Definitions?

- Define a real language "meaning", independent of a specific implementation
- Develop a solid basis for experimenting with extensions
- Help guard against ambiguities, unexpected feature interactions, overly complex features
- Provide a formal basis for analysis and proofs

Example: Java Ambiguities

Chan, Yang, and Huang found several tricky ambiguities in the Java language (Traps in Java, *Journal of Systems and Software*, Volume 72, Issue 1, pp 33–47):

- interplay between inheritance and packages
  - field access
  - abstract methods
  - overriding with static and instance methods
- overloading and overriding with widening

Example: Problems with Extensions

Bracha, Odersky, Stoutamire, and Wadler mention two problems introduced with different approaches to adding generics to Java (Making the future safe for the past: Adding Genericity to the Java Programming Language, *Proceedings of OOPSLA'98*):

- In their solution, it is possible to lose type soundness when generic code is used by non-generic code (i.e. when generic Java is compiled into standard class files and used in code compiled with a standard Java compiler)
- In another solution proposed by Agesen, Freund, and Mitchell, it is possible to specify a desired generic instance which cannot be properly placed in any package, leading to potential run-time errors

Formalizing Language Definitions

Many methods to define semantics have been proposed:

- Structural Operational Semantics (SOS)
- Natural (big-step) Semantics
- Modular SOS
- Reduction Semantics
- Denotational (various)
- Rewriting-based

We believe all have limitations which limit use. Regardless, none have come into widespread use as tools for language design.

Outline

Formal Support for Language Design

Our overall goal is to

- provide a **formal** environment for language design and evolution

Formal Support for Language Design

Our overall goal is to

- provide a **formal** environment for language design and evolution
- powerful enough to define real-world languages

Formal Support for Language Design

Our overall goal is to

- provide a **formal** environment for language design and evolution
- powerful enough to define real-world languages
- providing support for language feature prototyping

Formal Support for Language Design

Our overall goal is to

- provide a **formal** environment for language design and evolution
- powerful enough to define real-world languages
- providing support for language feature prototyping
- and supporting program execution and analysis.

Solution Components

- Graphical environment for language construction and animation
- Framework (notation, standards, environment tools) for language definition
- Language modules and complete definitions of new and existing languages
- Translations into existing languages and/or notations for execution and analysis

Why not fix other methods?

- In some cases, no good fixes – supporting some language features not possible
- In other cases, fixes too cumbersome
- Some methods are hard for practitioners to use regardless (many denotational methods require advanced knowledge of mathematics)
- Techniques based on rewriting logic and term rewriting are well understood, fairly simple, and have nice mathematical meaning (which can be leveraged or ignored)

Why another framework?

- Framework at level of semantic definition, not execution
- Not competing with Java, .NET, etc in these terms
- Trying to provide a formal, not just operational, definition of languages

Outline

Method Overview

Our approach encompasses several areas:

- A Framework for Language Definition
- Defining and Evolving Object-Oriented Languages
- Tool Support

A Framework for Language Definition

- Current work revolves around rewriting logic and Maude
- Move to using K, a PL-specific framework and notation based on rewriting logic
- K provides some conventions to make definitions more concise and modular
- Currently have used K in the classroom and on definitions of Java and KOOL
- Feedback from use of K is going back in to further enhancement

Defining and Evolving Object-Oriented Languages

- Defining existing languages ensures that technique is powerful enough to work in practice
- So far have defined Java, JVM bytecode, most of Beta
- Important to experiment with novel concepts as well: focus of KOOL and its extensions
- Trying to lift out common features into reusable feature libraries

Tool Support

- Current tool support is sparse: Maude provides good analysis and execution tools, but can be cumbersome
- Looking at graphical tools for working with language definitions
- Want to add ability to animate uses of semantics, follow process of program running in semantics
- Also looking at methods of automatically translating K definitions to various languages and term rewriting systems
- Other research on impact of language design decisions on analysis performance

Other Ideas

- Would it be possible to provide a common representation for various languages and translate between them?
- Can we check to ensure all needed rules and equations are defined?
- Can we automatically generate test cases based on defined rules to test the semantics?
- Can logical nature of semantics be leveraged to prove correctness of program transformations or optimizations?

Summary: Does this Meet Requirements?

- Define a real language "meaning", independent of a specific implementation: **Yes**
- Develop a solid basis for experimenting with extensions: **Yes** (we believe)
- Help guard against ambiguities, unexpected feature interactions, overly complex features: **Partially**
- Provide a formal basis for analysis and proofs: **Yes**

Outline

Questions and Discussion