Tests to the Left of Me, Types to the Right

How Not to Get Stuck in the Middle of a Ruby Execution (A Demo of Diamondback Ruby)

Michael Furr, Jong-hoon (David) An, Jeffrey S. Foster, and Michael Hicks

University of Maryland, College Park
{furr,davidan,jfoster,mwh}@cs.umd.edu

Abstract. Ruby is a popular dynamic scripting language that permits terse, expressive code, but provides no static checks to detect errors before running the program. To address this, we have developed Diamondback Ruby (DRuby), a tool that blends the benefits of static and dynamic typing. This paper briefly describes the main features of DRuby, which we will present in a tool demonstration. The presentation will concentrate on the development of a small, statically typed Ruby program, illustrating how DRuby might be used in practice. The audience will learn about some of the practical design decisions that went into DRuby, and how to use it to develop a type-safe Ruby program.

Ruby is an object-oriented scripting language that facilitates rapid development with dynamic typing and expressive language constructs. For example, Ruby includes an `eval` method that interprets strings as program text. However, this flexibility comes at a price. Mistakes that would be caught by static typing, e.g., calling a method with the wrong argument types, remain latent until run time. Such errors can be painful to track down, especially in larger programs.

Recently, we have been developing Diamondback Ruby (DRuby), an extension to Ruby that blends the benefits of static and dynamic typing. We chose to focus on Ruby as it provides an interesting cross section of features found in other dynamic languages such as Lisp, Smalltalk, Python and Perl. An important design goal for DRuby was to add static checking while keeping the *feel* of Ruby—correct Ruby programs should not require significant changes to be accepted as well-typed by DRuby. Similarly, any program that has been annotated with DRuby type annotations should also be a valid Ruby program. This second point is particularly important for rapid prototyping, since the programmer may wish to execute her program even in the presence type errors detected by DRuby. For example, she may wish to observe the layout of a GUI component, ignoring the fact that clicking on a button would cause the program to crash.

As many of the technical details of DRuby have been presented elsewhere [1, 2], our demonstration will focus on how DRuby might be used in practice. We will describe how to run the DRuby binary and give a tour of DRuby's main features: the type inference algorithm, the type annotation language, profile-guided analysis of dynamic constructs, and runtime contracts that ensure type safety

of dynamic code. We will illustrate these features on a small yet sophisticated statically typed Ruby application.

To run DRuby, the programmer simply invokes "`druby` *filename*" instead of "`ruby` *filename*." DRuby applies type inference to the entire program, emitting messages describing potential type errors. DRuby models common idioms as precisely as possible without programmer intervention. For example, DRuby types local variables flow-sensitively so their types may change during execution.

DRuby includes a type annotation syntax that closely resembles RDoc, a popular comment-based annotation language for Ruby. DRuby type annotations are useful both to type core library code (some of which is written in C rather than Ruby) and to allow the programmer to document code that has stabilized. Because DRuby's annotations appear in comments, they are ignored by the Ruby interpreter, keeping backwards compatibility. However, in contrast to RDoc, DRuby annotations are checked (either statically or dynamically) and will therefore never become out of date with the current implementation.

To analyze Ruby's dynamic constructs such as `eval`, the programmer can direct DRuby to profile these constructs while running unit tests, and then use these profiles to augment DRuby's static analysis [2]. Profile-guided analysis is particularly well-suited to DRuby because most Ruby programs already employ substantial test suites (testing is widely adopted in the Ruby community). It also gives the programmer a clear trade-off: The more dynamic features covered in the profile, the more accurate static checking will be. During the demonstration, we will show how to run the DRuby's profile-driven analysis using a popular Ruby testing framework.

Finally, even with profiling, some constructs cannot be statically analyzed by DRuby. For these cases, DRuby adds additional dynamic checks that properly blame the errant code for any runtime failures. We have taken special care to implement these checks in a practical way, e.g., keeping line numbers unchanged so that stack traces remain faithful to the original source file. We will demonstrate this technique, including showing how a programmer might ultimately distribute her instrumented, type-safe program.

To conclude, we believe that DRuby is a practical solution to integrating static and dynamic typing. While we continue to develop DRuby, it already has the potential to be useful to Ruby programmers. We hope that the audience will gain insight into the design of DRuby, and will begin writing type-safe Ruby programs with its help. DRuby is an open-source program built on top of RIL, an intermediate language for building static analyses for Ruby. DRuby and RIL are available as a single download from the DRuby website [3].

References

1. Furr, M., An, J., Foster, J.S., Hicks, M.: Static Type Inference for Ruby. In OOPS Track, SAC. (March 2009)
2. Furr, M., An, J., Foster, J.S.: Profile-guided static typing for dynamic scripting languages. In OOPSLA (October 2009). To appear.
3. http://www.cs.umd.edu/projects/PL/druby