RDT: A Dynamic Tracing Framework for R

AVIRAL GOEL, Northeastern University, USA
FILIP KŘIKAVA, Czech Technical University, Czechia
JAN VITEK, Northeastern University, USA and Czech Technical University, Czechia

Static analysis of R code is hard. A combination of dynamic typing, laziness with side-effects, introspection, metaprogramming, first-class environments and `eval` enable programming idioms that make it nearly impossible to statically derive reliable insights. Fortunately, most of the R packages available in CRAN and Bioconductor contain runnable programs in the form of examples, tests and vignettes[1] which makes the language well-suited for dynamic analysis. However, currently there is no dynamic tracing functionality in R other than the coarse-grained `trace` function which only allows tracing the entry and exit points of R closures.

In this talk we present RDT, a scalable and modular dynamic tracing framework that exposes many aspects of R, facilitating a fine-grained inspection of programs. It has two components:

– *R-dyntrace* is a small extension[2] to GNU R Virtual Machine with probes that are triggered on specific program execution events. There are probes for function entry and exit, variable definition and mutation, garbage collection entry and exit, non-local jumps, promise creation and execution, `eval` entry and exit, and S3/S4 dispatch. R-dyntrace provides an API to register user-defined callbacks to these probes along with a facility to store arbitrary state per tracing session.
– *dynalyzer* is a companion R package that provides utilities for building data-intensive dynamic analyses. The package facilitates quick construction of multistage pipeline for analysis of tracing data using the map-reduce programming model. It contains functions to automate logging and memoization of partially analyzed data from each stage. It also defines a data format for fast storage and retrieval of collected data with support for compression and streaming.

A dynamic analysis is written as a standalone R package. Its separation from the framework simplifies development and fosters reuse. It contains code that defines tracer state and registers callbacks for relevant events. When a callback is triggered, it receives the tracer state and information about the event, *i.e.*, affected R objects and the relevant R interpreter state. R-dyntrace API exports utilities to access fields of these objects as well as the global R interpreter state such as the evaluation order of builtin function arguments and variable lookup without triggering callbacks. Furthermore, R-dyntrace can detect nesting in callbacks, a consequence of the tracing algorithm inadvertently executing R code that potentially modifies the program state. This is a common stumbling block in naïve tracing attempts and our design captures these cases, reporting them as meaningful error message. There is also an escape hatch in the form of disabling specific probes when the R code in question is known to be benign.

We have used RDT for a large-scale study of the design and use of laziness in R. Its design has enabled us to scale the dynamic analysis to 14,875 R packages, tracking over 177.8 billion promises and 3.5 trillion events at the rate of 1.5 million events per second. The result is a comprehensive, data-oriented understanding of the role of laziness in real-world R code.

RDT enables a data-driven approach to understanding R program behavior and language usage patterns. We believe this approach is instrumental in providing insights for improving the design of existing features and driving the design of new features and better libraries for the dynamic languages of today.

[1]The packages contain approximately 4.4 million lines of runnable R code
[2]The complete extension to the virtual machine is about 1.8K lines of C code.