Provenance Tracking in R

Andrew Runnalls and Chris Silles

School of Computing, University of Kent, UK

Abstract. This poster describes current progress and issues in introducing provenance-tracking facilities into the CXXR implementation of the R statistical computing environment.

1 CXXR

The object of the CXXR project (`www.cs.kent.ac.uk/projects/cxxr`) is gradually to reengineer the fundamental parts of the R interpreter from C into C++ in such a way that the full functionality of the standard distribution of R (including the recommended packages) is preserved. In particular, the behaviour of R code is unaffected (unless it probes into interpreter internals), and there is no change to the existing interfaces for calling out from R to other languages such as C or Fortran, nor to the main APIs for calling into R. CXXR achieves a high degree of compatibility with R packages from the CRAN repository, as [1] illustrated.

Work on CXXR started in May 2007, at that time shadowing R-2.5.1. Since then CXXR has been regularly upgraded to keep pace with the major releases of R (usually synching on the .1 minor release), so for example over the last year CXXR has shadowed the increasing deployment of the bytecode compiler within standard R. The current release of CXXR shadows R-2.14.1.

A key difference between CXXR and standard R is in the implementation of R data objects. Standard R provides for only a fixed range of object types (implemented as a C union) to be assigned to R variables, and to participate in the interpreter's garbage collection scheme. In contrast, data objects in CXXR are implemented as a C++ class hierarchy, which can be extended at will. The provenance-tracking variant of CXXR leverages this feature extensively.

2 Provenance Tracking and CXXR

The AUDIT facilities [2] that once formed part of S and S-plus were an invaluable feature, as one of the present authors can testify, and one motivation behind CXXR was to introduce similar but better facilities into the R interpreter. Early work on a provenance-enabled variant of CXXR was presented in [3].

In the terminology of the OPM, the approach used is to regard **bindings** of R symbols (variables) to R data objects as being the OPM **artifacts**, and to regard R **top-level commands** (i.e. expressions entered directly at the interpreter prompt) as being OPM **processes**. At present no use is made of the OPM

agent concept. The interpreter instruments the reading and writing of bindings within the 'global environment' (R's main workspace), and maintains an audit trail defining the OPM graph leading up to all extant bindings. This provenance information can then be interrogated within the interpreter itself: this marks a difference from the S AUDIT facility, which required a separate tool to query provenance data.

For example, the CXXR command `pg <- pedigree("lm1")` will retrieve the 'pedigree' of the current binding of the symbol `lm1` (presumably to a linear model): `pg$commands` will then return an R list showing, in time order, the history of top-level commands that may have influenced that current value of `lm1`. Other components of `pg` record the dates and times when these commands were issued, and information relating to xenogenesis (Sec. 3).

A recent development in CXXR is to reengineer the way that data are serialized and deserialized between one session and the next, by drawing on the serialization facilities of the well-regarded open-source Boost C++ libraries (`www.boost.org`). This means that not only can developers extend the range of data types usable within the interpreter, they can also—within the new C++ class definitions themselves—specify how objects of that class are saved to and restored from the session archive (which now uses an XML format). This applies not least to the classes implementing the provenance audit trail, so that this is carried forward from one CXXR session to the next.

3 Xenogenesis

Many R functions are pure: their return value depends only on the values of the function arguments. Other functions may have a return value which depends on the values of other variables within the R session, and some functions—for example pseudorandom number generators—may have side effects, modifying the bindings of R variables otherwise than via their return values. Fortunately, none of the preceding presents any inherent problem for CXXR's provenance-tracking mechanism, provided the function's behaviour is entirely mediated by the bindings of R variables.

However, there are some R functions whose behaviour is *not* fully defined by the current state of the interpreter. This may be because the function (e.g. `scan`) reads from an external file or database, or because it accepts user input in some way (e.g. `identify`) or because it calls external non-R code via one of the foreign language interfaces. We call such functions **xenogenetic**, and the bindings they give rise to **xenogenous**: "due to an outside cause".

If a top-level command calls a xenogenetic function (either directly, or indirectly via some other function), this means that the text of the top-level command no longer completely defines the OPM process that maps its input artifacts (bindings) to its output artifacts. To work around this problem, the approach currently being explored is for the provenance record to identify whether a binding is xenogenous, and if so *to record the value* of that binding. So if for example an R function `my.function` was created using an external editor using R's `edit` command, the provenance record will permanently record the value thus given to `my.function`—permanently, that is, for as long as any artifact depending on that function is retained in the R session.

4 Environments

In R (and CXXR), an **environment** is a container holding a mapping from R symbols to R data objects. As previously mentioned, at present CXXR tracks the provenance of bindings within R's global environment `.GlobalEnv`. It is straightforward to extend this tracking to other standard environments set up at the start of an R session, though this results in a deluge of provenance data that would rarely be of value.

However, each invocation of a function written in R results in the creation of a local environment. In the overwhelming majority of cases this local environment becomes inaccessible after the function returns, and it is soon garbage-collected. However, there are some exceptions, and it is for example possible to define an R function which in effect has internal state, stored in a local environment and carried forward from one invocation to the next. At present this would result in a 'backchannel' of influence that evades the provenance record, but work is in the pipeline to rectify this.

One remaining concern is that there is currently no method of referring to local environments in a way that is meaningful between R sessions: this can hamper reproducibility, especially in the presence of xenogenesis.

5 Conclusion

This poster has described the current state of work to introduce provenance-tracking facilities into CXXR. The reader will have noted that the tracking is self-contained within the interpreter, and does not rely on any external provenance-tracking tool. A less satisfactory converse is that the implementation does not currently provide for any interoperation with such external tools. The authors would be interested to hear from researchers interested in collaborating to rectify this and other gaps.

References

1. Runnalls, A.: CXXR and add-on packages. In: useR! 2010 (2010), `http://user2010.org/slides/Runnalls.pdf`
2. Becker, R.A., Chambers, J.M.: Auditing of data analyses. SIAM J. Sci. Stat. Comput. 9, 747–760 (1988)
3. Silles, C.A., Runnalls, A.R.: Provenance-Awareness in R. In: McGuinness, D.L., Michaelis, J.R., Moreau, L. (eds.) IPAW 2010. LNCS, vol. 6378, pp. 64–72. Springer, Heidelberg (2010)