

R⁷RS Large Status and Progress

Daphne Preston-Kendal & John Cowan

December 2023

Work on the Revised⁷ Report on the Algorithmic Language Scheme (R⁷RS) began in 2009 and was intended produce two compatible languages: a small language, intended for ‘education, programming-language research, embedded systems, and embedded scripting languages’, and a large language, intended to meet the ‘practical needs of mainstream software development’. Dividing the Scheme language specification into two parts in this way was intended to ensure that the resulting languages together satisfied the needs and desires of as many Schemers as possible. The small language was completed in 2013; the large language is expected to be completed in the next two or three years. This paper details progress on the large language as well as the historical background of the Scheme language’s development.

1 Historical background

R⁵RS was completed in 1998 and added the well-known `syntax-rules` hygienic macro system as a mandatory part of the specification, after R⁴RS (1991) proposed it in an optional appendix. The R⁴RS appendix also proposed a lower-level procedural macro system for implementing the high-level `syntax-rules` system, but this was not included in R⁵RS. Several different implementation strategies for `syntax-rules` existed, and while there was consensus on `syntax-rules` as a high-level system, there was no agreement about which of these low-level systems should complement it for situations in which `syntax-rules` proved too limited.¹ The main competing systems were R. Kent Dybvig and Bob Hieb’s syntax object system from the R⁴RS appendix, which later evolved into the `syntax-case` system (discussed below); syntactic closures by Alan Bawden, Jonathan Rees, and Chris Hanson; and explicit renaming by Will Clinger.

In the same year that the R⁵RS was completed, the Scheme Requests for Implementation (SRFI) process began. This comparatively informal process allows any Schemer to propose a new language feature for Scheme. Initial SRFIs included, for example, SRFI 1 (a comprehensive list library) and SRFI 9 (a means of defining record types, the first ever standard on creating custom types in Scheme). These SRFIs among others were widely adopted.

Attendees of the 2002 Scheme Workshop resolved to appoint a group to create a successor to R⁵RS. The elected editors and steering committee completed their work in 2007, delivering the R⁶RS. This report specified a more comprehensive language than R⁵RS, but largely

¹`Syntax-rules` cannot destructure a Scheme atom (defined as a datum other than a pair or a vector), nor can it introduce deliberately unhygienic bindings. Although the `syntax-rules` system is Turing complete, it is purely a template-based rewriting system — macro definitions cannot run actual Scheme code at expand time — so it is also in practice something of a Turing tarpit for many non-trivial expansions.

ignored the fruits of the SRFI process, defining (for example) a different list library and record system from those specified in SRFIs 1 and 9. It also adopted the **syntax-case** system for procedural macros, although the controversy about low-level macro systems was (and is) still ongoing. For this and other reasons, including the increase in the size of the report and language, the R⁶RS was controversial in the Scheme community. While previous reports had been rejected by some implementations and users,² unusually many implementations chose not to adopt the R⁶RS.

A new steering committee was elected in 2009, which resolved that the next version of the report would be split into two parts, as described above, to resolve a perceived tension between those in the community who want a light-weight Scheme, useful as a tool for research and education in programming language design and implementation and for embedding, and those who want a language practical for programming in the large. Each part would be developed by its own working group. Working Group 1 delivered the R⁷RS small language specification in 2013; Working Group 2, developing the large language, paused its activity until this was finished, reactivating itself in 2014. At this time it also reconstituted itself as a community open to all interested Schemers, rather than a small committee with closed membership.

2 Large language progress

John Cowan as initial chair of Working Group 2 held three votes of the reconstituted Working Group 2 in 2016 (‘Red Ballot’), 2018–19 (‘Tangerine Ballot’), and 2021–2 (‘Yellow Ballot’). These represented three out of a planned 12 volumes to build R⁷RS up out of individual SRFIs, library by library.

In 2022, spurred by discussions after the results of the Yellow Ballot were announced, work on the large language was split into three parts: Foundations (core semantics); Batteries (portable standard libraries); and Environments (operating system service interfaces).

In 2023, John Cowan resigned as Working Group 2 chair due to what appeared to be irresolvable disagreements about the Foundations, and was succeeded by Daphne Preston-Kendal. She postponed further discussion of the contentious issues and produced a plan to complete the Foundations in time for Scheme’s fiftieth anniversary in December 2025, focussing initially on areas with strong existing consensus and progressing through a series of topic-themed ‘fascicles’ of the final report.

2.1 Macros

By the time of TFP24, the first such fascicle, on the macro system, should have been released or be nearly ready for release.

The R⁷RS Large macro system, as chosen on the Yellow Ballot, is **syntax-case** as in R⁶RS. To address criticisms levelled at the R⁶RS version of **syntax-case**, lower-level features from the original R⁴RS appendix have been restored. The most significant of these is **unwrap-syntax**, which allows a syntax object to be deconstructed procedurally: to answer objections that the **syntax-case** system offers no way to destructure macro uses other than

²R⁵RS’s macro facility, and the addition of the **dynamic-wind** facility for acting upon the activation and de-activation of a continuation, were its most controversial new features; to this day R⁴RS advocates contend that they are unnecessary warts, making very similar arguments to those used by opponents of R⁶RS.

with its own high-level pattern matcher, we explicitly encourage Schemers to use these low-level primitives in developing their own high-level macro systems. As an example, the fascicle includes a non-normative appendix implementing a version of explicit renaming in terms of syntax objects.

Other features adopted in this fascicle by the Yellow Ballot include syntax parameters, which allow quasi-unhygienic identifier capture in **syntax-rules** macros by adjusting the binding of a syntax keyword, and identifier properties, an analogue of classical Lisp symbol properties which fully respect hygiene and the library system. The latter in particular makes it possible to define the **syntax-case** pattern matcher in terms of lower-level primitives.

The nature of a democratic vote makes it hard to explain definitively why **syntax-case** was chosen despite the controversy over it in R⁶RS, since voters' reasons for their choices cannot be known. In our view, the following factors are relevant. Syntax object based systems provide the strongest hygiene (including in the case that hygiene should deliberately be broken). The **syntax-case** pattern matcher is also among the most ergonomic solutions for developing macros for which **syntax-rules** is too limited, for the same reasons pattern-matching code is in general superior to equivalent procedural destructuring. Patterns and templates in **syntax-case** are identical in form to those in **syntax-rules**, meaning the system is an easy step up, both for learners and when extending existing code.

2.2 Procedural programming

Apart from resolving public comments submitted on the Macrological Fascicle, the immediate next task is the preparation of the next fascicle after it, the Procedural Fascicle. This part of the report will specify the standard syntax for procedural programming in Scheme, including variable definition and binding, procedure creation with **lambda**, etc. Its contents are essentially the core of Scheme as an 'interpreter for extended lambda calculus'.

This fascicle may relax the classical Scheme restriction that, in a procedure or similar body, all definitions have to precede all expressions. Apart from this, the remaining features adopted on the Yellow Ballot will be incorporated, including tagged procedures (procedures with boxes for arbitrary Scheme values) and a form from 1985's RRRS for creating locally-recursive anonymous procedures.

2.3 Completing the Foundations

Five additional fascicles are planned: the Valued Fascicle (all of Scheme's standard data types and storage management facilities, except for procedures and record types); the Erroneous Fascicle (the handling of error conditions in R⁷RS Large); the Bibliothecarial Fascicle (syntax and semantics of libraries and top-level programs as well as the REPL); the Flow Controlling fascicle (control-flow features, likely including delimited continuations); and the Record-Winning Fascicle (the record type system).

The most significant consequence of the 2022 decision to adopt **syntax-case** as the R⁷RS Large macro system was that total compatibility with R⁶RS became a plausible goal. Throughout 2022 and 2023, an initial consensus emerged in favour of at least making it possible for implementations to support both standards in a fully interoperable way; more recently, this goal has developed into making it possible to implement all the syntax and procedures of R⁶RS on top of R⁷RS Large.

More broadly, it is Daphne Preston-Kendal’s goal as chair not to reunite the languages per se, but to make R⁷RS Large a tool for reuniting the community fragmented by the split between R⁶RS and R⁷RS Small. In order to ensure support from implementers and users in different parts of the Scheme community, Foundations features must be supported by at least three major implementations before they can be definitively accepted.

2.4 Batteries

Most work done under John Cowan’s chairship had the nature of the Batteries; in large part, the overall outline of this report is clear. Daphne Preston-Kendal has taken steps to cull proposals by consensus, to ensure the scope of the Batteries volume is manageable. Some libraries may be removed by vote for this reason.

The Batteries will provide data structures, algorithmic primitives, and syntactic extensions which programmers of Lisp and other functional languages have come to expect, while taking advantage of developments in computer science since the development of Common Lisp. Batteries libraries can be implemented portably on top of the Foundations; sample implementations will be provided for all of them, making Batteries support ‘free’ for implementers who support the R⁷RS Foundations.

There is no target completion date for this work at present, but work is continuing in parallel to work on the Foundations. As such, the Batteries report should be complete at latest not long after the Foundations report.

2.5 Environments

The final volume of the R⁷RS Large specification is the Environments volume, providing mainly operating system service interfaces such as file system access and networking.

There is as yet very little progress in this area. Very many questions remain open, including even whether we should favour low-level or higher-level solutions in this area in general.

3 Conclusion

R⁷RS Large is the most complex specification the Scheme community has ever attempted. As such, it has made slow progress; however, it has now moved from exploration to consolidation. We believe that the end, or at least the beginning of the end, is very much in sight. Progress is being made through the SRFI process and coordinated on an issue tracker;³ much informal discussion also happens on the IRC channel `#scheme` on the Libera Chat network. In a few years, we expect we will have a Scheme specification in which the majority of Schemers perceive the essential qualities of Scheme, as they understand them, to be reflected. R⁷RS Small has already been adopted by both existing and new implementations, some of which also support R⁶RS; we hope this development towards reconciliation continues with adoption of R⁷RS Large.

³<https://codeberg.org/scheme/r7rs/issues>