R^{7}RS Large: Bringing Schemers (Back) Together for Scheme's Fiftieth Birthday

Daphne Preston-Kendal

Proposal for Trends in Functional Programming 2025
Not intended for publication in proceedings

The Revised^7 Report on the Algorithmic Language Scheme is the latest revision of the Scheme reports, the de facto standard specification for the full-funarg, lexically scoped dialect of Lisp – 'an interpreter for extended applicative order lambda calculus' – first invented by Gerald J. Sussman and Guy L. Steele in 1975. The community of Scheme programmers has been internally divided for much of the last twenty years, and as the chair of the working group responsible for its completion, my hope is that the new report will become part of a trend towards healing this division. This paper will explain the social and political factors behind the division, the technical and other challenges which face reunification of the community, and finally look at the new features which will bring standard Scheme up to the expectations of programmers in the 2020s.

1 Historical background

The history of the split in the Scheme community over the future of the language after the R^{5}RS (1998) is well known, but I will rehearse it here briefly for the benefit of those who are not familiar with the recent history of Scheme standardization.

Scheme reports up to and including the R^{5}RS were conservative in nature. This reflected the workings of the committees which designed them, which required unanimity of all sixteen or more members before any change or addition could be made. At the Scheme Workshops held in 2002 and 2003, participants voted to establish a Steering Committee to oversee all future standardization efforts and a panel of editors to develop what would become R^{6}RS. In contrast to previous committees, the R^{6}RS editors were only five in number and made decisions by majority vote, rather than by unanimous consensus.

The R^{6}RS was completed and ratified in 2007. However, warning signs about its future appeared almost immediately: two months after ratification, Marc Feeley surveyed maintainers of many implementations of Scheme about their intentions regarding R^{6}RS; most respondents said they had no intention of taking any action to support it. While some of those implementations did eventually add support, hostility towards R^{6}RS in some quarters did not subside quickly. This failure of R^{6}RS to achieve universal acceptance disheartened even its editors and advocates: Scheme 48, the implementation maintained by R^{6}RS editor Michael Sperber, never added support; PLT Scheme (now Racket), maintained by editor

Matthew Flatt, significantly reduced the usefulness of its R^{6}RS mode with version 4 in 2008, which made the pair/list type in its default mode immutable, impeding transparent interoperability between R^{6}RS libraries and other Racket code.

In 2009 new Steering Committee elections were held, and the result was a landslide of candidates who had voted against or abstained on the ratification of the R^{6}RS. The new Steering Committee resolved to split the Scheme language in two, appointing two new working groups to produce, respectively, a 'small' language intended to be useful for education, research, and embedded environments, and a 'large' language compatibly extended from it which would be more suitable for the needs of mainstream software development.

The R^{7}RS small language was completed in 2013 and has proven a success: all R^{5}RS implementations which had refused to support R^{6}RS have adopted R^{7}RS small. The R^{7}RS large language has had a considerably more difficult gestation, but completion may finally be appearing on the horizon.

2 Ongoing divisions

Both the Scheme language and the Scheme community remain divided over R^{6}RS and R^{7}RS small, however.

Neither R^{7}RS working group was directed to maintain any form of R^{6}RS compatibility. Despite this, the language developed by the R^{7}RS small working group was remarkably close to being a compatibly-extended subset of R^{6}RS.

Unfortunately, some differences remain. While both reports tried to update Scheme for a Unicode world where there is no longer 1:1 correspondence between bytes and characters, the R^{6}RS editors devised a new lexical syntax for bytevectors and a completely new set of I/O procedures; R^{7}RS small adopted an older lexical syntax already supported by many R^{5}RS implementations, and compatibly extended the R^{5}RS I/O library.[1]

Another difference is the library systems. R^{5}RS provided no practical means of namespacing: before R^{6}RS, every implementation devised its own approach. R^{6}RS provided a simple `library` form with import and exports. The R^{7}RS small committee decided that the library system should have some of the properties of a build configuration system with conditional compilation, and named their version `define-library`. Some Scheme implementations support one form, some the other, and few support both: it is still tricky to write a library that works seamlessly on both an R^{6}RS-only implementation and an R^{7}RS small-only implementation.

Besides technical issues such as these, there are also still political tensions between advocates of the R^{5}RS/R^{7}RS small languages and advocates of R^{6}RS. Time has healed these wounds somewhat: as mentioned, some implementations which initially expressed reservations about supporting R^{6}RS have added support, and some R^{6}RS implementations support R^{7}RS small. The Steering Committee has responded positively to recent decisions by the large language working group which have set the goal of transparent R^{6}RS/R^{7}RS interoperation by providing enough low-level primitives in R^{7}RS to implement all the features of R^{6}RS. Nonetheless, feelings still run high in some quarters, and engagement with some

[1]The R^{6}RS kept the R^{5}RS provisions around for backward compatibility, but did not fully upgrade them for both Unicode and binary I/O as R^{7}RS small later did.

implementations' communities and developers has been difficult due to residual sentiment against one or another version of the report.

3 R^{7}RS large im Wandel der Zeiten

Upon completion of the R^{7}RS small specification in 2013, work on the large specification began in early 2014. John Cowan was the first chair. At this point, membership of the working group was also opened so that anyone interested in Scheme could join.

Work initially focussed on providing a portable standard library of data structures and similar basic extensions to the small language through the SRFI (Scheme Requests for Implementation) process,[2] and for a while it looked like this might in fact be what R^{7}RS large would be: a collection of SRFIs. In all, three ballots of working group members (any interested Schemers) were held in 2016, 2019, and 2022. These mostly adopted proposals for portable libraries, although some decisions affecting the core semantics were also met, such as a requirement that R^{7}RS large implementations would have to support the full tower of numeric data types. Most notably, the last ballot in 2022 led to the adoption of the `syntax-case` procedural macro system from R^{6}RS, which opened the door to the possibility that R^{7}RS large might be, in some form, compatible with R^{6}RS.

This discussion merged with several other long-standing concerns of some working group members that the semantic issues of the small language, which made it unsuitable for the purposes the large language was intended for, were greater than John Cowan seemed to appreciate. The small language report specifies almost all error cases, including out-of-bounds array accesses and many others with safety implications, as undefined behaviour (in the C sense). The working group agreed to resolve these issues by defining a new superset of the small language with better safety guarantees, to be called the R^{7}RS large Foundations and published as the first volume of the eventual report. Other work on standard libraries would go into a Batteries volume[3] for portable libraries, and an Environments volume specifying additional non-portable features, mainly those requiring operating system support.

Arguments in the working group, mainly centred around the extent of R^{6}RS compatibility in the Foundations and in the R^{7}RS large language as a whole, became increasingly heated. It was against the background of this heated discussion that John Cowan, the working group's first chair, resigned in August 2023. The steering committee installed me as his successor the next month.[4]

My first decision as chair was to postpone discussion of the most controversial issues and begin immediate work on the specification for the Foundations using

[2]The SRFI system is similar to Python Enhancement Proposals (PEPs) in the Python community, Java Specification Requests (JSRs) for Java, Swift Evolution documents (SEs) for Swift, etc.; it is considerably more open than these processes, but also less binding. Any Schemer may submit a proposal for a library or semantic change in Scheme; but finalization of a proposal simply means the community agrees that it has reached a state of maturity, without in itself imposing any requirement on implementations to support the proposal.

[3]The reference is to Python's 'batteries included' slogan; the final volume is likely to be called something more prosaic.

[4]I was the only one crazy enough to volunteer for the job, although John Cowan did some not-so-subtle persuading, both to get me to volunteer and to get the steering committee to accept me.

the consensus we already had. I also developed a roadmap that would ensure discussion at any one time remained focussed on one particular area.

Overall, my (ambitious) aim as chair is to reunify the Scheme community, not only to reunify the R^{6}RS and R^{7}RS small languages with one another. It is still not yet decided exactly to what extent R^{6}RS will become a 'part' of R^{7}RS, but as mentioned above, interoperability and the ability to implement R^{6}RS in terms of R^{7}RS are the current consensus for minimum baseline goals. As we work through the roadmap, our decisions on these and many other issues will come more into focus.

The roadmap also significantly cut down the scope of potential new features. The proposals list maintained by John Cowan had 139 proposed but not yet accepted features, almost all of which would have had to be codified into one (or more) SRFIs. The SRFI process officially takes between 60 and 90 days per proposal, and often much longer; furthermore, there were in practice very few people working on turning these proposals into SRFIs.

The new roadmap needs only 14 SRFIs to be written and finalized for the Foundations volume, and in practice that volume could likely become a satisfactory specification of a core language without most of those. The Batteries have also been reduced so that only 20 SRFIs are currently requested; we could likewise do without many of those and still have a perfectly serviceable set of portable standard libraries. The Environments will require considerably more creativity to be useful, which is partly why no possible completion date for that volume can yet be announced.

The original target completion date for the Foundations was December 2025, in time for the fiftieth anniversary of the first Scheme report. This was an ambitious target when I set it in 2023, and Hofstadter's law was bound to intervene, as it did in 2024 with administrative and other delays, scuppering hope of realistically meeting this target. It might be feasible to have the whole R^{7}RS large report – Foundations, Batteries, and Environments – completed by 2028, the anniversary of Steele and Sussman's Revised Report on Scheme.

4 Scheme revival and community reunification

The split over R^{6}RS and R^{7}RS was so severe that it could perhaps have spelled a slow death for Scheme: Racket's decision to break R^{n}RS compatibility in its core language could have set a precedent which other larger implementations might have followed.

Happily, R^{7}RS large is arriving just as a new generation of hackers discovers Scheme and breathes new life into the community. To name one significant project, the Spritely Institute is developing a new toolkit for creating distributed applications in Scheme. Since 2021, they have mainly been working in Guile, which is notable as one of the implementations which supports both R^{6}RS and R^{7}RS small with a high level of interoperation. Application developers who use Spritely's tools can therefore freely use a mix of libraries written for either report's version of the language, as well as Guile's own implementation-specific libraries. The success of Spritely and Guile is an encouraging sign that R^{7}RS large's approach of reunifying the two languages with the goal of creating a reunited Scheme community might work. Other exciting developments include Spritely's Hoot project, which adds a WebAssembly backend to Guile and may prove to be the first WebAssembly compiler for any garbage-collected language whose closure sizes are small enough to be practical for use in in-browser web applications. Notably, Spritely aimed first for R^7RS small support in Hoot before trying to support more of Guile's standard library.

There are other developments in reuniting the two versions of the language. Scheme does not yet have a universally accepted package registry like Hackage in Haskell, but the Akku package registry and package manager by Gwen Weinholt is well-positioned to become one. It supports a dozen Scheme implementations, with some support for installing packages written for R^7RS on R^6RS implementations by rewriting `define-library` to `library` declarations at installation time. It also indexes packages from the older Snow repository which Alex Shinn, chair of the R^7RS small working group, created for R^7RS small packages.

As mentioned, there are still some in the R^6RS and R^5RS/R^7RS small camps who seem somewhat implacably opposed to one or the other version of the language. Time will tell if they can be persuaded by the success of projects which point towards a reunification of the community, such as Spritely, Akku, and hopefully the R^7RS large specification.

5 R^7RS large changes and additions

In contrast to previous status updates on R^7RS large, this report has focussed less on technical features and more on the social and political aspects of a programming language community. To conclude, I will provide a brief overview of some things that may appear in R^7RS large which were not in any previous Scheme report.

5.1 Macro features

The first fascicle of the Foundations specification was released as a first public draft in October 2024. Besides cleaning up and unifying the R^6RS and R^7RS small provisions for macros, it includes two significant new features: syntax parameters and identifier properties.

Syntax parameters allow dynamically adjusting the macro transformer associated with a binding within an entire lexical block of code. This means some seemingly unhygienic macros, which re-introduce the same unhygienic identifier binding each time, can be implemented by merely 'bending' hygiene. They are already supported by Racket, Chibi Scheme, Guile, Gerbil, and Chez Scheme.

Identifier properties allow programmers to write macros which add context-specific new semantics to existing bound identifiers. For example, my pattern matching proposal (section 5.6) uses identifier properties to determine the meaning of a 'constructor' identifier which appears on the left-hand side of a pattern matching clause. It would also be possible, for example, to extend Scheme's `set!` to work like a hygienic version of Common Lisp's `setf` using identifier properties, or to provide an expand-time mechanism for finding the relationships between the procedures which belong to a record type. They were first introduced in Chez Scheme and, as of writing, are still only supported by Chez and by the R^7RS expander Unsyntax.

5.2 Storage control

Weak references and finalization are important primitives which, among other things, provide a flexible means of extending the functionality of built-in Scheme types without memory leaks. Under John Cowan's chairship, ephemerons were adopted as the primitive weak referencing system, which is semantically beneficial since all other known weak reference types can be implemented in terms of ephemerons, but not vice versa. Ephemerons are fully supported by Chez Scheme (and by extension in Racket, which uses the same storage manager) and MIT Scheme; slightly buggy implementations are provided by Chibi Scheme and Gambit (and by extension Gerbil, whose relationship to Gambit mirrors that of Racket to Chez).

We will probably adopt guardians as a finalization mechanism. Unlike classical finalizers, guardians queue objects for finalization but require explicit action to de-queue them and run any finalization code. The mechanism is generation-friendly, and it is always clear when and in which dynamic environment and thread finalization code will run, which is not always the case with classical finalizers. Guardians are supported by Chez Scheme and by Guile.

5.3 Tagged procedures or applicable records

R^{4}RS defined the behaviour of Scheme's built-in polymorphic equivalence predicates when applied to first-class procedure values. This is sometimes useful, but requiring `lambda` to return a newly-created procedure on each evaluation inhibits many compiler optimizations, such as lifting. R^{6}RS therefore removed this feature of the R^{4}RS and left unspecified whether these equivalence predicates would return true or false when given procedures as arguments. Upon request of Gerald J. Sussman and several over voters on the ratification, R^{7}RS small restored a variant of the R^{4}RS semantics at the eleventh hour.

R^{7}RS large is a compatible superset of R^{7}RS small, so its definition of procedure equivalence remains. It is being clarified, though, in a way which does not hinder any known optimizations of `lambda` expressions. In parallel, we are probably going to provide some variant of `lambda` which does guarantee that every evaluation will create a first-class procedure object which is distinguishable from every other object.

There are several potential approaches here. One is simply as described above: a constructor for procedures which may not be lifted, eta-converted, or have any other optimizations applied to it which would prevent its identity from being useful. Another is a form of tagged procedure, which contains an additional box whose value can be accessed without calling the procedure. Many Scheme implementations already support this kind of tagged procedure, but they are not strictly type-safe as one can easily pull the value out of one procedure's box and put it in another's, even if the other procedure does not have the behaviour that would normally be implied by that value being in its box. So a type-safe variant may be developed; or, in the most ambitious proposal, record types may be extended to add the ability for a type to define what happens when one of its instances is applied like a procedure.

5.4 Delimited control operators

Scheme's `call/cc` is possibly its most distinctive feature. It is very powerful, but unsafe. Calls for its removal have grown ever louder in the last quarter-century, since newer primitive control operators have been developed which are just as powerful but which enable derived control features to compose better with one another. Many Scheme implementations already include some delimited control system: it is thus a matter of standardizing a common interface.

Racket has integrated `call/cc` seamlessly into its system of delimited control operators by adding an additional, optional prompt argument, alongside a 'composable continuation' variant. This seems a good model for standard Scheme to follow, and SRFI 226 proposes adopting Racket's delimited control operators. SRFI 226 is a maximalist and somewhat un-approachable proposal, though, and includes provisions for some things that may not be practical to standardize at all, such as threading. We will likely develop a reduced version.

5.5 Data structures

Moving from the Foundations to the Batteries, we aim to include a selection of data structures which satisfies the expectations of functional programmers today. R^{6}RS standardized a library for classical, mutation-based hash tables, but functional programmers now expect efficient persistent mapping types of some kind. Both sorting- and hash-based persistent mappings will be included in the Batteries. Classical hash tables will also be provided, but likely with a twist: they will probably be required to remember the order in which items were added, since implementation techniques have been demonstrated by which a hash table with this property can be implemented with less memory usage than one which does not maintain insertion order.

Finger trees and Clojure-style persistent growable vectors are two other significant developments in data structures since the R^{6}RS. Since finger trees depend on laziness for good asymptotic performance, and Scheme runtimes are optimized for eager evaluation, it is not clear whether they would be practical if included in R^{7}RS large's Batteries. We will likely opt for one or the other.

Mutation-based dynamically-sized vectors are also useful in some situations and have been proposed. We are not yet sure whether their utility over persistent growable vectors and/or finger trees is great enough to warrant inclusion.

While proposals have been made for more specialized data structures, such as radix trees for persistent integer-keyed maps, the working group has not yet decided whether these should be included in the Batteries or left as install-able packages.

5.6 Pattern matching

Programmers of other functional languages have complained about Scheme's lack of pattern matching facilities for almost as long as Scheme has existed. In the 1990s a de facto standard third-party library emerged in the form of an unhygienic macro written by Bruce Duba and Andrew K. Wright; this was subsequently ported to hygienic `syntax-rules` by Alex Shinn. Many Scheme implementations included the 'Wright' pattern matcher or Alex Shinn's re-implementation of it.

Alas, an attempt to standardize Wright-style pattern matching in SRFI 204 foundered, in part because of the SRFI author's self-admitted inexperience, but also because of subtle semantic disagreements between Wright's original implementation, Shinn's re-implementation, and other re-implementations, which could not be usefully resolved. Duba and Wright also made the unfortunate decision that the syntax for deconstructing values should be based on the lexical syntax of datum types, rather being named by the procedural forms used to construct values. This makes it tricky to adapt to match other types such as hash tables or the other data structures mentioned above, which do not have a lexical syntax.

I am proposing an alternative pattern matching construct which does not have these issues. This pattern matcher can be taught about new types of patterns using Scheme's familiar macro system, so writing patterns for new types of data structure is easy, and new patterns have comparable efficiency at run time to the ones for Scheme's built-in data structures. The proposal is based on a pattern matching library developed by Sam Tobin-Hochstadt for Racket, which has also been used as a model for pattern matching macros in Emacs Lisp and Common Lisp.

Pattern matching ought to be more widely used in Scheme. Unfortunately it is unlikely that it will become a standard idiom for some time yet, but incorporating a pattern matching construct into the R^{7}RS Batteries would set a positive example. Wright's pattern matcher is well-known but has the syntactic and semantic weaknesses mentioned; my pattern matcher, while based on solid precedents, is new to Scheme, depends on the likewise-new identifier properties, and does not have much acceptance yet.

6 Conclusion

In the year of Scheme's fiftieth birthday, it may be possible to hope that the divisions which have plagued the Scheme community for the last twenty years might soon be resolved.

One unique challenge faced by Scheme standardization is the sheer number of different implementations, the variety of platforms and environments they run in, and the variety of approaches to implementing the language. The fact that all of these implementations are maintained by volunteers, so any new requirement in the specification adds more to the to-do lists of dozens of people working in their free time, only exacerbates this issue. As working group members we are trying to mitigate this problem by actively engaging with implementers, and by submitting patches to implementations ourselves adding support for R^{7}RS features. Through this work, we can see that most implementers, including those who rejected one or the other previous version of the language, are open to supporting efforts to bring the two versions together.

Furthermore, as implementations, third-party packages, and the R^{7}RS specification develop, it is becoming more and more practical to write Scheme libraries portable between implementations without significant porting effort. This would be a new milestone in the development of Scheme and, empowered by the growing new community of Schemers, set us up to be a lively and relevant language for the next fifty years.

Partial Bibliography

This partial bibliography should provide good starting points for the interested reader to find more information about any particular point raised above.

1. Alex Shinn, John Cowan, & Arthur A. Gleckler (eds.) *Revised^7 Report on the Algorithmic Language Scheme* (small language), 2013. `https://standards.scheme.org/official/r7rs.pdf` Includes brief history of the language with references to all prior Scheme reports and standards.

2. ‘R6RS Ratification Vote’, 2007. `https://www.r6rs.org/ratification/results.html` Lists the rationales of all voters against the ratification of R6RS, and many of those who voted in favour.

3. Marc Feeley, ‘Implementors’ intentions concerning R6RS’, posting on *r6rs-discuss@r6rs.org*, 27 October 2007. `https://www.r6rs.org/r6rs-discuss/3298.html`

4. Arthur A. Gleckler, ‘Scheme Requests for Implementation Status Report’, presentation at the 2022 ICFP Scheme Workshop. `https://www.youtube.com/watch?v=xzZfdPtHv0k`

5. John Cowan, reports on the Red, Tangerine, and Yellow editions of R^{7}RS large, 2016–22. Archived at `https://codeberg.org/scheme/r7rs/src/branch/main/ballot-results/jcowan/edition`

6. John Cowan, ‘ColorDockets.md’ (2014–22) `https://github.com/johnwcowan/r7rs-work/blob/master/ColorDockets.md`

7. Christine Lemmer-Webber, Randy Farmer, & Juliana Sims ‘The Heart of Spritely: Distributed Objects and Capability Security’ (draft white paper, last updated 2024) `https://files.spritely.institute/papers/spritely-core.html`

8. Gwen Weinholt, ‘Akku: Package management made easy’ `https://akkuscm.org`

9. Daphne Preston-Kendal (ed.) ‘The Revised^7 Report on the Algorithmic Language Scheme: Foundations Fascicle I: The Macrological Fascicle’ (public draft 1), 2024. `https://r7rs.org/large/fascicles/macro/1/`

10. Central repository for development of the R7RS Large specification of the Scheme programming language, `https://codeberg.org/scheme/r7rs` The files `FOUNDATIONS.txt` and `FOUNDATIONSTOC.txt`, `BATTERIES.txt`, and `ENVIRONMENTS.txt` contain current complete agendas for all features that might possibly be in the language. The wiki page ‘Needed SRFIs’ contains a list of proposals for which no existing SRFI-level specification yet exists and will need to be developed.

11. John Cowan, ‘Ephemerons’, Scheme Request for Implementation 124. `https://srfi.schemers.org/srfi-124/srfi-124.html` Includes references to papers on the design and implementation of ephemerons.

12. R. Kent Dybvig, David Eby, & Carl Bruggeman, ‘Guardians in a generation-based collector’ in *ACM SIGPLAN 1993 Conference on Programming Language Design and Implementation* 207–16, 1993.

13. Oleg Kiselyov, ‘An argument against call/cc’ (last updated 2012) `https://okmij.org/ftp/continuations/against-callcc.html` Includes extensive references to articles on alternative control operators.

14. Daphne Preston-Kendal, ‘Pre-SRFI for an extensible pattern matcher’. `https://codeberg.org/dpk/extensible-match` Includes extensive history of pattern matching with a focus on its use in Scheme, with exhaustive bibliography on the topic.