Growing Schemes

Twenty Years of Scheme Requests for Implementation

ARTHUR A. GLECKLER, Editor, Scheme Requests for Implementation, USA

SRFI[21], or Scheme Requests for Implementation, is an informal process for extending the Scheme programming language. It has run in parallel with, and has sometimes contributed to, more formal Scheme standardization efforts like R^{6}RS and R^{7}RS. Since the inception of SRFI in 1998, SRFI contributors have prepared 162 detailed documents proposing extensions to Scheme, most with sample implementations, and many with test suites as well. SRFIs cover ideas like mechanisms for concurrency, error handling, internationalization, and pattern matching; control constructs and data structures; module systems; operating system interfaces; and more.

In this paper, we explain the purpose of the SRFI process, how the process works, and how it has been part of the evolution of Scheme. We hope that implementers and users of Scheme and other programming languages can benefit from learning about this approach to moving the language forward.

CCS Concepts: • **Software and its engineering → Language features**;

Additional Key Words and Phrases: Scheme programming language, Scheme Requests for Implementation, standards

ACM Reference Format:
Arthur A. Gleckler. 2018. Growing Schemes: Twenty Years of Scheme Requests for Implementation. 1, 1 (September 2018), 18 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Twenty years ago, after an R^{5}RS meeting, thirty-two people attended the Scheme Strawman Workshop[22, 36]. The workshop announcement said, "Our intent is to get users and implementors together to discuss their experiences with existing extensions and to discuss possible extensions and modifications to Scheme." The discussion that started with that meeting has continued online ever since as SRFI, the Scheme Requests for Implementation.

1.1 Motivation

If you are a Scheme programmer, you may use more than one Scheme implementation, either because different ones are suited for different operating systems, or because you prefer one for its performance but another for its debugging tools, or because one is better for teaching and another is better for production, or for another reason. But you would surely still like to be able to run code you wrote for one implementation on the others. Formal Scheme standards like R^{5}RS, R^{6}RS[5], and R^{7}RS[8] lay the groundwork, providing a concrete, portable layer, but they only go so far. The Scheme Requests for Implementation process picks up where those standards leave off. SRFI is a less formal way to propose new extensions to Scheme, to discuss them with other users, and to encourage Scheme implementers to support them.

Author's address: Arthur A. Gleckler, Editor, Scheme Requests for Implementation, Sunnyvale, CA, 94087, USA, srfi@speechcode.com.

XXXX-XXXX/2018/9-ART
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1.2 Making use of SRFIs

There are many ways to make use of the SRFI process:

- If you need a specific data structure, or a portable operating system interface, or a testing framework, or support for bitwise operations or concurrency or random numbers or regular expressions, or some other library or extension to the language, you can turn to existing SRFIs. Your Scheme implementation may already support the SRFI you want. If not, try using the provided sample implementation.
- If the feature you want exists in one implementation but not in others, you can document it clearly and use the SRFI process to propose that other implementations adopt it.
- If you are the author or maintainer of an existing Scheme implementation, you can choose from a wide variety of SRFIs and either use the provided sample implementations or write your own.
- If you are helping to create one of the official Scheme standards, e.g. an R^{n}RS, you can use the SRFI process to publish and discuss proposals for inclusion in your standard. Both R^{6}RS[5] and R^{7}RS-large[8] have done this. Or you can simply draw ideas from SRFIs and incorporate them, as R^{7}RS-small[8] did.
- You can participate in the discussion of proposed SRFIs, influencing their APIs and how they work, contributing to their design and implementation, and encouraging the authors and maintainers of your favorite Scheme implementations to adopt them.

2 HOW DOES THE SRFI PROCESS WORK?

For many use cases, the SRFI process is simple: review the SRFIs provided by the Scheme implementation being used, find the one with the desired features, load it using that implementation's standard mechanism, and use it. If the implementation's documentation doesn't describe the SRFI, read the documentation on the SRFI web site[21].

Each SRFI is identified by a unique number in addition to its title, so it's easy to look it up. As of this writing, there are 124 "final" SRFIs (see figure 1 and table 3) covering a wide variety of needs.

2.1 Requests

SRFIs are requests, not formal standards. The idea is that someone who wants a new feature to be added to Scheme implementations can propose the feature, specify it in detail, provide a sample implementation and automated tests, and discuss and revise the proposal publicly so that it is as understandable and useful as possible. All this makes it as easy as possible for authors and maintainers of Scheme implementations to support the requested feature.

Once a SRFI has been finished, or "finalized," it's up to the authors and maintainers of Scheme implementations to decide whether to support it. But even if they don't, support for a particular SRFI can often be added to an implementation by its users.

The SRFI editors ensure that all the requirements are met before a SRFI can be finalized[1]. For example:

- Each SRFI document must be in HTML format.
- Each SRFI document must follow a standard template[20] that includes a rationale, a detailed specification, a description of the sample implementation, an acknowledgements section, and an open-source copyright statement.
- Sample implementations are required in all but the most unusual circumstances.[2]
- Automated tests are strongly encouraged.

[1] This work is done manually for the most part, but automated tests are run and spell checkers are used.

[2] Only five of the 124 final SRFIs — 18, 21, 22, 36, and 97 — do not include some form of sample implementation, either in the document itself, elsewhere in the Git repo, or in some external location, e.g. as part of an existing Scheme implementation.

Per the SRFI process, the editors are not allowed to reject a proposal because they "...disagree with the importance of the proposal, or because they think it is a wrong-headed approach to the problem. The editors may, however, reject a proposal because it does not meet the requirements listed here[24]." An editor's job is to facilitate the discussion and dissemination of SRFIs, but not to approve or reject them based on the ideas they contain.

Ultimately, each SRFI is the responsibility of its author. While the author may revise the SRFI based on public discussion on the SRFI mailing list — and, indeed, this is common — the final decision whether to finalize it is up to the author, assuming that the editor agrees that it meets the SRFI requirements.[3] Once it is finalized, though, it is up to the maintainers and users of Scheme implementations to decide whether to incorporate it into each implementation. This approach gives many ideas the breathing room and support that they might not get with a more consensus-driven process.

2.2 Email

The process of proposing and reviewing a new SRFI is organized around email. It is described in detail at [24], but is summarized here.

An author proposing a new SRFI submits a proposal to the SRFI editor(s) at srfi-editors@srfi.schemers.org. The proposal document must be in HTML format and must follow the standard template[20]. Once the editor is satisfied that the proposal conforms to the SRFI requirements, the editor assigns it a unique SRFI number, publishes it as a first draft on the SRFI web site, creates a public email mailing list specifically for discussion of the new proposal, creates a Git[12] distributed version control repository on Github[17], and announces the new SRFI and mailing list publicly.

Discussion of the SRFI is held on its public mailing list. Anyone may subscribe to any SRFI's mailing list. Any subscriber may participate in the discussion. All messages are not only delivered to every subscriber, but are archived publicly on the web so that there's a public, long-term, searchable record of all the decisions and thinking that went into making the final SRFI, and of any alternatives that were discussed, objections that were raised, etc.

Even after an SRFI is finalized, its email mailing list remains open. That way, clarifications and corrections can be made and other discussion can occur even years later.

2.3 States

Each SRFI can be in one of several states: *draft*, *final*, and *withdrawn*. (See figure 1.) Each SRFI starts in the draft state, and remains in that state through however many drafts are published as a result of discussion.

At any time while a SRFI is in the draft state, its author may decide that it is not ready for publication. The editor will then move it into the withdrawn state and publicly announce that change. The withdrawn state is permanent. Discussion may continue, but the SRFI has effectively been retired at that point. Later, the same author or another author may decide to revive the proposal. In that case, a new SRFI is created and the process starts over, leaving the original one untouched. SRFI numbers are not reused, and each withdrawn SRFI retains its number.

If, after a thorough discussion and perhaps several drafts, the author decides that the SRFI is finished and the editor agrees that all the requirements have been met (e.g. the document is clear and contains all the right information and the sample implementation and tests work correctly), the editor places the SRFI into the final state and publicly announces that change. After that happens, the specification is never changed except perhaps to correct indisputable, unambiguous errors, e.g. grammar errors, typos, links to no-longer-existing web sites,

[3]The idea that the author controls the SRFI is sometimes difficult for reviewers to accept. Prior editors have had to work hard, when moderating discussion of SRFIs, to get this point across.[45]

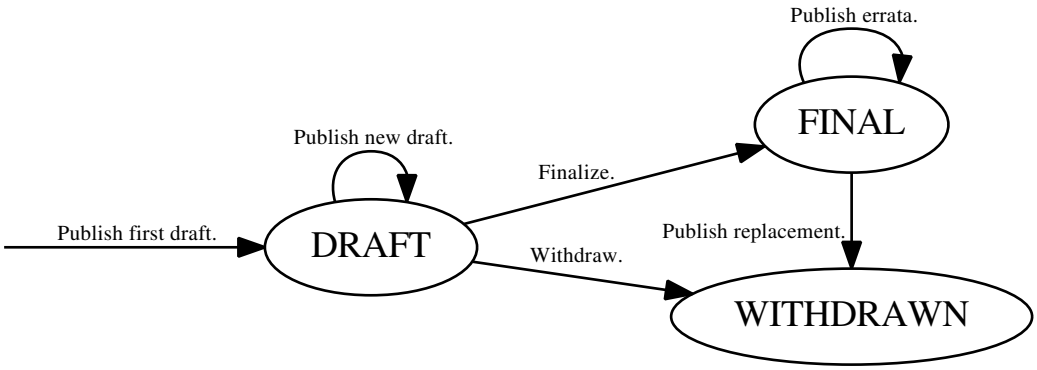

Fig. 1. SRFI state transitions

or self-contradictory statements. All substantial errata are announced on the SRFI's mailing list and explained in the document's Status section with specially marked notes. Corrections or improvements to the sample implementation are also allowed after finalization. Design flaws and substantial mistakes must be addressed in a new SRFI. Note that withdrawn SRFIs are still present on the web site, in full — just marked withdrawn.

Occasionally, a new SRFI is created with the intention of replacing an older SRFI. The older, final SRFI may be withdrawn, but only if a newer SRFI replaces it and the author of the original SRFI agrees. In that case, there will be a "See also" link from the withdrawn SRFI to the new one on the home page. That way, implementers will know that there is a new version. This is the only official way to deprecate an SRFI. Several SRFIs have gone through this transition, e.g. SRFIs 40 (replaced by 41), 114 (obsoleted by 128), and 142 (obsoleted by 151).

Note that the sample implementation for a SRFI is not necessarily portable code. Some SRFIs propose extensions to the language that cannot be made without deep changes to the implementation itself. In that case, the sample implementation is to be used as an example rather than ported directly.

2.4 Version control

All changes to the specification document, the sample implementation, and any other code or documents that comprise an SRFI are recorded in its public version control repository. We[4] use the Git[12] version control system because it is popular and practical and because it is distributed, which means that anyone can easily make a copy of the repository, make changes to that copy independently, and offer those changes for inclusion in the draft SRFI (or to fix errata).

While we use Github to host the repositories for all of the SRFIs, we studiously avoid using any features of Github other than those which can easily be recorded in the email history. That way, if we switch from Github to another host someday, we will not have to go to any effort to preserve our public history. While we do accept Github pull requests[26], the editor carefully copies any information from accepted pull requests to the SRFI mailing list so that everyone can see it.

There are some SRFI contributors and users who do not want to use Github at all, either because they use a purely email-based workflow or for other reasons. They are first-class contributors, too, and we support them by accepting new code, documents, and patches by email and by making the complete contents of all SRFI repositories available available as a downloadable archive on the SRFI home page. (We would welcome the creation of cloned repositories, either self-hosted or on other providers like Gitlab.)

[4]We'll refer to the SRFI editor(s) as "we" henceforth.

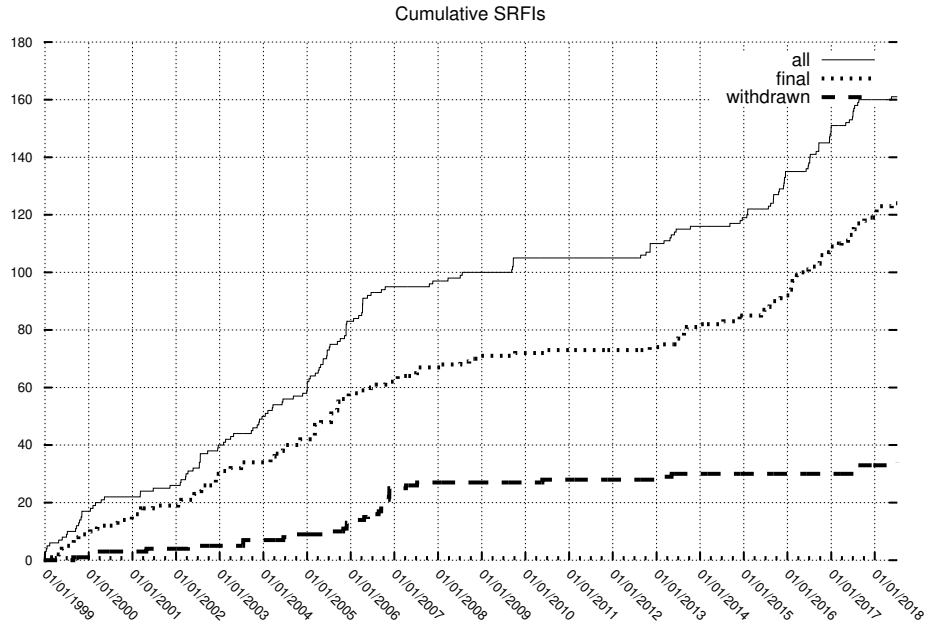

Fig. 2. SRFIs over time

Note that, until April 2015, SRFI version control was done using the CVS version control system[38]. All SRFI work was done in a single repository, and that repository was not public. When the transition to Git was made, the CVS history was preserved in Git, but separated into one repository per SRFI and one more for common work, including administrative code and the contents of the home page and other parts of the web site. All of those repositories are now public.

3 HISTORY OF SRFI

SRFI has been a long-lived institution[22], operating for twenty years so far under eight different editors[19], all volunteers. Fifty-seven people have been authors of now 162 SRFIs (124 finalized (see table 3), 4 draft, and 34 withdrawn[18]). Many more people have contributed to SRFI discussions and implementations.

Figure 2 shows how the number of SRFIs has grown over time. For reference, note that R^{5}RS was published in 1998, R^{6}RS was published in 2007, R^{7}RS-small was published in 2013, and that work on R^{7}RS-large began in 2013. SRFIs continued to be written in between the publication of R^{6}RS and the start of R^{7}RS-small, but more slowly, perhaps because of the lull in formal standardization efforts.

3.1 Relation to R^{n}RS standards

While SRFI has operated in parallel with the R^{n}RS standards, acting as a supplement to them, not an alternative, its history has been intertwined with theirs, too.

The editors of R^{6}RS used the SRFI process as a way to organize public discussion of language changes and extensions they were considering[13]. They adopted the process of publishing draft SRFIs, holding public discussion on the SRFI mailing lists, and then withdrawing them once a decision had been made about what to

include in the standard, regardless of the decision. This period accounts for many of the SRFIs that are now in the withdrawn state. Despite the withdrawal of these SRFIs, many of the ideas that are in R^{6}RS reached it through the SRFI process. For example, SRFIs 11, 33, 34, 60, 74, 75, 76, 77, 83, and 93 were largely incorporated into R^{6}RS, and a subset of SRFI 1 was, too[6].

The editors of the R^{7}RS-small working group didn't explicitly make SRFIs a part of the standard-making process. Nevertheless, several specific SRFIs, e.g. for records (SRFIs 9 and 99), were discussed, debated, and used directly or as inspiration for features of R^{7}RS-small. R^{7}RS-small incorporates nine SRFIs. The working group used SRFIs as far as was appropriate for an effort organized around standardizing the language as opposed to adding libraries. The R^{7}RS-small standard document specifically acknowledges the influence of SRFIs 0, 1, 4, 6, 9, 11, 13, 16, 30, 34, 39, 43, 45, 46, 62 and 87[7].

The R^{7}RS-large discussion and voting process, which is still under way, is explicitly organized around SRFIs. Many new SRFIs have been proposed as part of R^{7}RS-large, and many more are planned as part of later dockets.

3.2 Web site

The SRFI primary web site, srfi.schemers.org, is ordinary in most respects, but it has a few noteworthy characteristics:

- It's entirely static in the sense that only static files are served to the users. There are no dynamic web request handlers.
- Despite this, it supports searching and sorting as well as filtering by keywords and SRFI state (i.e. draft, final, or withdrawn). All of this is done in Javascript so that it can be instantaneous. Searching for a SRFI is much like incremental search in Emacs[15]. However, as searching, sorting, and filtering are done, the URL is updated, so it's possible to bookmark the results.
- Care has been taken to make all the common pages, including the SRFI landing pages, display well on mobile web browsers. While the SRFI documents themselves haven't been adjusted, CSS has been added that makes them display well on mobile devices, too.
- SSL/TLS has been added to the entire site to prevent unauthorized tampering with the contents as it is transmitted.
- The aforementioned mobile and TLS improvements have the additional advantage that they improve Google search ranking.

3.3 Survey of implementations

We have surveyed the documentation of twenty-two Scheme implementations to determine which SRFIs are implemented by each. See table 1 for the survey data. Note that some SRFIs are supported as part of the R^{6}RS or R^{7}RS conformance, and are not listed in this table. (See section 3.1.)

Figure 3 shows the number of Scheme implementations in which each SRFI is implemented. This includes not only final SRFIs, but also ones that have been withdrawn, e.g. as part of R^{7}RS standardization.

Figure 4 is a visualization of which final SRFIs are supported in which implementations. The X axis represents increasing final SRFI numbers. The Y axis represents all the Scheme implementations in the survey data.

Table 2 lists SRFIs in order of decreasing number of implementations. As one would expect, earlier SRFIs typically have the most implementations. Some are even supported by almost all implementations. However, more recent SRFIs are gradually getting more implementations.

Note that some of these Scheme implementations are no longer actively maintained, which is why they don't support more recent SRFIs.

3.4 Other history

There have been eight editors over the history of SRFI, often several at a time, but there is currently only one. There is no formal process for the transition from one editor to the next. Each group of editors has chosen the next based at least partly on who has volunteered.

The topics covered by SRFIs continue to be an eclectic mix of everything from data structures to operating system interfaces. There is no clear pattern of change in the topics. However, many of the most recent SRFIs are revisions of earlier ones, sometimes to make them more consistent with each other or with R^{n}RS standards.

Over the past three years, we have made several changes to the SRFI process:

- We have experimented with allowing the SRFI draft period to extend past ninety days. This experiment has had mixed success. Since then, sometimes due to factors beyond the authors' control, some SRFIs have taken more than a year to finalize. On the other hand, some of these long-running SRFIs have produced excellent results. We intend to return to enforcing the original deadlines for new SRFIs.
- We now allow a finalized SRFI to be withdrawn with the approval of its author, but only if there is a new SRFI to replace it.
- There is now a formal process for accepting and publishing errors in the document.
- Every SRFI has its own public Git version control repository.

Implementation	Count	SRFIs Supported	Sources
Bigloo	11	0, 1, 2, 4, 6, 8, 9, 14, 18, 22, 28	[40]
Chez	59	0, 1, 2, 4, 5, 6, 8, 9, 11, 13, 14, 16, 17, 19, 23, 25, 26, 27, 28, 29, 31, 34, 35, 37, 38, 39, 41, 42, 43, 45, 48, 51, 54, 60, 61, 64, 67, 69, 78, 98, 99, 115, 117, 125, 126, 127, 128, 129, 130, 131, 132, 133, 141, 143, 145, 151, 152, 156, 158	[25]
Chibi	50	0, 1, 2, 6, 8, 9, 11, 14, 16, 18, 23, 26, 27, 33, 38, 39, 41, 46, 55, 69, 95, 98, 99, 101, 111, 113, 115, 116, 117, 121, 124, 125, 127, 128, 129, 130, 132, 133, 134, 135, 139, 141, 142, 143, 144, 145, 147, 151, 154, 159	[41], [42]
Chicken	68	0, 1, 2, 4, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 23, 25, 26, 27, 28, 29, 30, 31, 34, 37, 38, 39, 40, 41, 42, 43, 45, 46, 47, 48, 49, 55, 57, 58, 60, 61, 62, 63, 64, 66, 69, 71, 72, 78, 87, 88, 89, 90, 95, 98, 99, 101, 102, 113, 116, 117, 121, 127, 128, 133	[10], [9]
Foment	8	1, 60, 106, 111, 112, 124, 125, 128	[37], [37]
Gambit	22	0, 2, 1, 4, 6, 8, 9, 13, 14, 16, 18, 19, 21, 22, 23, 27, 28, 30, 39, 40, 88, 95	[11]
Gauche	75	0, 1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14, 16, 17, 18, 19, 22, 23, 25, 26, 27, 28, 29, 30, 31, 34, 35, 36, 37, 38, 39, 40, 42, 43, 45, 46, 55, 60, 61, 62, 64, 66, 69, 74, 78, 87, 95, 96, 98, 99, 106, 111, 112, 113, 114, 117, 118, 121, 125, 127, 128, 129, 131, 132, 133, 134, 141, 143, 145, 146, 149, 151, 152, 158	[35]
Guile	44	0, 1, 2, 4, 6, 8, 9, 10, 11, 13, 14, 16, 17, 18, 19, 23, 26, 27, 28, 30, 31, 34, 35, 37, 38, 39, 41, 42, 43, 45, 46, 55, 60, 61, 62, 64, 67, 69, 71, 87, 88, 98, 105, 111	[14]
Ikarus	22	0, 1, 2, 6, 8, 9, 11, 13, 14, 16, 19, 23, 26, 27, 31, 37, 39, 41, 42, 43, 67, 78	[16]
Kawa	38	0, 1, 2, 4, 6, 8, 9, 10, 11, 13, 14, 16, 17, 23, 25, 26, 28, 30, 35, 37, 38, 39, 41, 45, 60, 62, 64, 69, 87, 88, 95, 97, 98, 101, 107, 108, 109, 118	[3]

Larceny	85	0, 1, 2, 5, 6, 8, 9, 11, 13, 14, 16, 17, 19, 23, 25, 26, 27, 28, 29, 30, 31, 34, 37, 38, 39, 41, 42, 43, 45, 48, 51, 54, 55, 59, 60, 61, 62, 63, 64, 66, 67, 69, 71, 74, 78, 86, 87, 95, 98, 99, 101, 111, 112, 113, 114, 115, 116, 117, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 141, 142, 143, 144, 145, 146, 147, 151, 152	[4]
MIT	15	0, 1, 2, 6, 8, 9, 10, 23, 27, 30, 39, 60, 62, 69, 131	[23]
Mosh	27	0, 1, 2, 6, 8, 9, 11, 13, 14, 16, 19, 23, 26, 27, 31, 37, 38, 39, 41, 42, 43, 48, 61, 67, 78, 97, 98	[31]
Racket	47	1, 2, 4, 6, 7, 8, 9, 11, 13, 14, 16, 17, 19, 23, 25, 26, 27, 28, 30, 31, 34, 35, 38, 39, 40, 41, 42, 43, 45, 48, 54, 57, 59, 60, 61, 62, 63, 64, 66, 67, 69, 71, 74, 78, 86, 87, 98	[50]
Sagittarius	71	0, 1, 2, 4, 6, 8, 13, 14, 17, 18, 19, 22, 23, 25, 26, 27, 29, 31, 37, 38, 39, 41, 42, 43, 45, 49, 57, 60, 61, 64, 69, 78, 86, 87, 98, 99, 100, 101, 105, 106, 110, 111, 112, 113, 114, 115, 116, 117, 120, 121, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 139, 141, 142, 143, 144, 145, 151, 158	[27]
Scheme48	38	1, 2, 4, 5, 6, 7, 8, 9, 11, 13, 14, 16, 17, 19, 22, 23, 25, 26, 27, 28, 31, 34, 37, 39, 40, 42, 43, 45, 60, 61, 62, 63, 66, 67, 71, 74, 78, 95	[39], [28]
SCM	11	0, 1, 2, 8, 9, 47, 58, 59, 60, 63, 70	[29]
SCSH	22	1, 2, 5, 6, 7, 8, 9, 11, 13, 14, 16, 17, 19, 23, 25, 26, 27, 28, 30, 31, 37, 42	[44]
Sig	17	0, 1, 2, 6, 8, 9, 22, 23, 28, 34, 38, 43, 48, 55, 60, 69, 95	[30]
SLIB	11	0, 1, 2, 8, 9, 47, 59, 60, 61, 63, 96	[34]
STklos	42	0, 1, 2, 4, 6, 7, 8, 9, 10, 11, 13, 14, 16, 17, 18, 22, 23, 26, 27, 28, 30, 31, 34, 35, 36, 38, 39, 45, 48, 55, 59, 60, 62, 66, 69, 70, 74, 88, 89, 96, 98, 100	[32]
Ypsilon	16	1, 6, 8, 9, 13, 14, 19, 26, 27, 28, 38, 39, 41, 42, 48, 98	[33]

Table 1. SRFI support by Scheme implementations

SRFI	Title	Count
1	List Library	22
8	receive: Binding to multiple values	21
2	AND-LET*: an AND with local bindings, a guarded LET* special form	20
9	Defining Record Types	20
6	Basic String Ports	19
0	Feature-based conditional expansion construct	17
14	Character-set Library	17
23	Error reporting mechanism	17
27	Sources of Random Bits	16
39	Parameter objects	16
13	String Libraries	15
26	Notation for Specializing Parameters without Currying	15
60	Integers as Bits	15
16	Syntax for procedures of variable arity	14
28	Basic Format Strings	14
11	Syntax for receiving multiple values	13
19	Time Data Types and Procedures	13
38	External Representation for Data With Shared Structure	13

31	A special form ‘rec’ for recursive evaluation	12
42	Eager Comprehensions	12
69	Basic hash tables	12
98	An interface to access environment variables	12
4	Homogeneous numeric vector datatypes	11
17	Generalized set!	11
37	args-fold: a program argument processor	11
41	Streams	11
43	Vector library	11
30	Nested Multi-line Comments	10
45	Primitives for Expressing Iterative Lazy Algorithms	10
61	A more general cond clause	10
25	Multi-dimensional Array Primitives	9
34	Exception Handling for Programs	9
62	S-expression comments	9
78	Lightweight testing	9
18	Multithreading support	8
48	Intermediate Format Strings	8
64	A Scheme API for test suites	8
95	Sorting and Merging	8
22	Running Scheme Scripts on Unix	7
55	require-extension	7
67	Compare Procedures	7
87	=> in case clauses	7
128	Comparators (reduced)	7
7	Feature-based program configuration language	6
10	#, external form	6
35	Conditions	6
63	Homogeneous and Heterogeneous Arrays	6
66	Octet Vectors	6
99	ERR5RS Records	6
111	Boxes	6
117	Mutable Queues	6
125	Intermediate hash tables	6
127	Lazy Sequences	6
133	Vector Library (R7RS-compatible)	6
5	A compatible let form with signatures and rest arguments	5
29	Localization	5
40	A Library of Streams	5
59	Vicinity	5
71	Extended LET-syntax for multiple values	5
74	Octet-Addressed Binary Blocks	5
88	Keyword objects	5
101	Purely Functional Random-Access Pairs and Lists	5
113	Sets and bags	5
121	Generators	5

129	Titlecase procedures	5
131	ERR5RS Record Syntax (reduced)	5
132	Sort Libraries	5
141	Integer division	5
143	Fixnums	5
145	Assumptions	5
151	Bitwise Operations	5
46	Basic Syntax-rules Extensions	4
112	Environment Inquiry	4
115	Scheme Regular Expressions	4
116	Immutable List Library	4
124	Ephemerons	4
130	Cursor-based string library	4
134	Immutable Deques	4
47	Array	3
54	Formatting	3
57	Records	3
86	MU and NU simulating VALUES & CALL-WITH-VALUES, and their related LET-syntax	3
96	SLIB Prerequisites	3
106	Basic socket interface	3
114	Comparators	3
126	R6RS-based hashtables	3
135	Immutable Texts	3
142	Bitwise Operations	3
144	Flonums	3
152	String Library (reduced)	3
158	Generators and Accumulators	3
36	I/O Conditions	2
49	Indentation-sensitive syntax	2
51	Handling rest list	2
58	Array Notation	2
70	Numbers	2
89	Optional positional and named parameters	2
97	SRFI Libraries	2
100	define-lambda-object	2
105	Curly-infix-expressions	2
118	Simple adjustable-size strings	2
123	Generic accessor and modifier operators	2
139	Syntax parameters	2
146	Mappings	2
147	Custom macro transformers	2
12	Exception Handling	1
15	Syntax for dynamic scoping	1
21	Real-time multithreading support	1
33	Integer Bitwise-operation Library	1

72	Hygienic macros	1
90	Extensible hash table constructor	1
102	Procedure Arity Inspection	1
107	XML reader syntax	1
108	Named quasi-literal constructors	1
109	Extended string quasi-literals	1
110	Sweet-expressions (t-expressions)	1
120	Timer APIs	1
122	Nonempty Intervals and Generalized Arrays	1
136	Extensible record types	1
137	Minimal Unique Types	1
138	Compiling Scheme programs to executables	1
149	Basic Syntax-rules Template Extensions	1
154	First-class dynamic extents	1
156	Syntactic combiners for binary predicates	1
159	Combinator Formatting	1

Table 2. SRFI rank by number of Scheme implementations
SRFIs in order of decreasing number of implementations

4 RELATED WORK

For the relationship between SRFIs and the R^{n}RS standards, see section 3.1.

4.1 CDR

The Common Lisp programming language has a web site that is somewhat similar to SRFI in that it is a place where proposed improvements to the language, etc. are collected: the CDR, or Common Lisp Document Repository[1]. CDR aims to be a way to collect "specifications of libraries, language extensions, example implementations, test suites, articles, etc." In that sense, it's similar to SRFI. However, it's different in that it doesn't attempt to organize the discussion or review of the documents it collects: "The Common Lisp Document Repository intentionally does not define a process for coming up with specifications or any other means to guarantee some level of quality of the submitted documents. Instead, we aim for a community-driven, decentralized approach to come up, discuss and finalize specifications. In this sense, we only provide the services of librarians." Currently, there are fifteen documents in the CDR, and it was last updated in 2013.

4.2 Package managers

Package managers are an increasingly common feature of programming language implementations. They typically allow automatic downloading and installation of new code, and sometimes dependencies as well.

- Quicklisp[2] is a popular package manager for Common Lisp. As of this writing, it supports eleven different Common Lisp implementations. It can download and install over 1,500 packages and their dependencies.
- Snow[43] is a package manager for any R^{7}RS-compliant Scheme. As of this writing, it has two client implementations that together support seven different Scheme implementations. It can download and install 134 different packages. Note that 22 of those are SRFI implementations.
- Eggs Unlimited[47] is a package manager only for Chicken Scheme. As of this writing, it can download and install 777 different packages. Note that 66 of those are SRFI implementations.

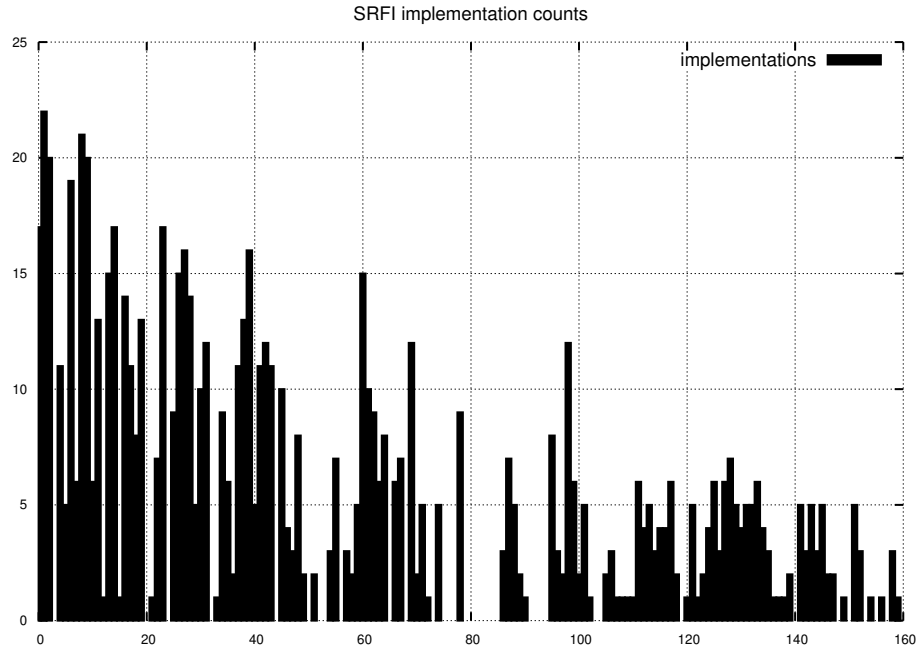

Fig. 3. SRFI implementation counts
The number of Scheme implementations in which each SRFI is implemented

Fig. 4. SRFI implementation matrix
A visualization of which final SRFIs are supported in which implementations. The X axis represents increasing final SRFI numbers. The Y axis represents all the Scheme implementations in the survey data.

- Racket Packages[49] is the package manager only for Racket[48], a language implementation that "started life as a Scheme implementation." As of this writing, it can download and install 1,005 different packages and their dependencies.

There is overlap between package managers and the SRFI process, but they are quite different. Package managers go beyond the SRFIs in that they include software for downloading and installing code rather than being the collections themselves. However, the SRFI process goes beyond package managers because it is a process for producing new extensions rather than just a way of downloading and installing them. Furthermore, SRFIs are often used to propose extensions to the language that cannot be made with portable code, but which require deep changes to the Scheme implementation itself. In such cases, package managers are not enough.

It is worth noting that SRFIs 0 and 7 were originally intended to specify a package manager that would be used for defining and installing Scheme software packages, but that this plan didn't come to fruition[46].

5 FUTURE OF SRFI

We hope that the SRFI process will continue for even more decades, producing more SRFIs that are taken up and supported by more Scheme implementations, both directly and through package managers like Eggs Unlimited, Racket Packages, and Snow. In support of those goals, we'd like to make more improvements, including:

- Improve the formatting of SRFI documents to make them more pleasant to read on desktop displays, mobile displays, and paper.
- Add semantic markup to SRFI documents so that indexes of definitions, for example, can be made automatically.
- Support more Git hosts, including perhaps GitLab and self-hosted Git repositories.
- In collaboration with contributors, revise the policy on the length of discussion periods. Technically, the process requires that it take sixty to ninety days, but we haven't enforced that for a while. Still, some SRFIs have taken over a year to finish, and that's too long.

Help from fellow Scheme users is and would be greatly appreciated.

6 CONCLUSIONS

By many metrics, SRFI has been a successful process, e.g. by longevity (twenty years), number of proposals (162[18]), number of Scheme implementations supporting SRFIs (at least 22, according to table 1), number of authors (57[18]), and number of participants (unknown, but in the hundreds, as there are currently 716 unique email addresses subscribed to SRFI mailing lists.)

Furthermore, SRFIs have been incorporated into all R^{n}RS standards since SRFI started: R^{6}RS, R^{7}RS-small, and R^{7}RS-large.

SRFI has been less successful by some metrics, e.g. number of SRFIs supported by each implementation. (See table 1.) Many implementations support only a few of the 124 finalized SRFIs. Note, however, that some SRFIs are revisions of or replacements for earlier SRFIs, or are directly incorporated into the standards that Scheme implementations support, so implementing the full 124 is not to be expected. Finally, some SRFIs may just be bad ideas. A committed author can get a pet idea through the SRFI process even if few users, much less Scheme implementers, would be interested in it. This is both a good thing and a bad thing.

One lesson to be learned from the SRFI process is that a process less formal than the R^{n}RS processes, one with smaller scope per work product (i.e. a SRFI rather than an R^{n}RS report), and one that has less demanding requirements for approval, but that still facilitates public discussion (as opposed to the Common Lisp Document Repository[1], which leaves that entirely to the author), can lead to increased frequency and quantity of contribution, and to work that Scheme implementers and standards committee members alike will adopt.

Another lesson is that less formal processes can help facilitate more formal standardization efforts, and can help bridge the often long periods between publication of formal standards. A language like Scheme, with so many implementations, benefits from a common way to propose improvements — one that does not require consensus. New ideas can be tried with little cost, then folded into formal standards once their benefits, costs, and ramifications are well understood through experience.

Requests really can grow a language.

A WHAT SRFIS EXIST?

Here are all the finalized SRFIs as of this writing:

No.	Title
0	Feature-based conditional expansion construct
1	List Library
2	AND-LET*: an AND with local bindings, a guarded LET* special form
4	Homogeneous numeric vector datatypes
5	A compatible let form with signatures and rest arguments
6	Basic String Ports
7	Feature-based program configuration language
8	receive: Binding to multiple values
9	Defining Record Types
10	#, external form
11	Syntax for receiving multiple values
13	String Libraries
14	Character-set Library
16	Syntax for procedures of variable arity
17	Generalized set!
18	Multithreading support
19	Time Data Types and Procedures
21	Real-time multithreading support
22	Running Scheme Scripts on Unix
23	Error reporting mechanism
25	Multi-dimensional Array Primitives
26	Notation for Specializing Parameters without Currying
27	Sources of Random Bits
28	Basic Format Strings
29	Localization
30	Nested Multi-line Comments
31	A special form ‘rec’ for recursive evaluation
34	Exception Handling for Programs
35	Conditions
36	I/O Conditions
37	args-fold: a program argument processor
38	External Representation for Data With Shared Structure
39	Parameter objects
41	Streams
42	Eager Comprehensions
43	Vector library
44	Collections
45	Primitives for Expressing Iterative Lazy Algorithms
46	Basic Syntax-rules Extensions
47	Array
48	Intermediate Format Strings
49	Indentation-sensitive syntax
51	Handling rest list
54	Formatting

55 require-extension
57 Records
58 Array Notation
59 Vicinity
60 Integers as Bits
61 A more general cond clause
62 S-expression comments
63 Homogeneous and Heterogeneous Arrays
64 A Scheme API for test suites
66 Octet Vectors
67 Compare Procedures
69 Basic hash tables
70 Numbers
71 Extended LET-syntax for multiple values
72 Hygienic macros
74 Octet-Addressed Binary Blocks
78 Lightweight testing
86 MU and NU simulating VALUES & CALL-WITH-VALUES, and their related LET-syntax
87 => in case clauses
88 Keyword objects
89 Optional positional and named parameters
90 Extensible hash table constructor
94 Type-Restricted Numerical Functions
95 Sorting and Merging
96 SLIB Prerequisites
97 SRFI Libraries
98 An interface to access environment variables
99 ERR5RS Records
100 define-lambda-object
101 Purely Functional Random-Access Pairs and Lists
105 Curly-infix-expressions
106 Basic socket interface
107 XML reader syntax
108 Named quasi-literal constructors
109 Extended string quasi-literals
110 Sweet-expressions (t-expressions)
111 Boxes
112 Environment Inquiry
113 Sets and bags
115 Scheme Regular Expressions
116 Immutable List Library
117 Mutable Queues
118 Simple adjustable-size strings
119 wisp: simpler indentation-sensitive scheme
120 Timer APIs
121 Generators

122	Nonempty Intervals and Generalized Arrays
123	Generic accessor and modifier operators
124	Ephemerons
125	Intermediate hash tables
126	R6RS-based hashtables
127	Lazy Sequences
128	Comparators (reduced)
129	Titlecase procedures
130	Cursor-based string library
131	ERR5RS Record Syntax (reduced)
132	Sort Libraries
133	Vector Library (R7RS-compatible)
134	Immutable Deques
135	Immutable Texts
136	Extensible record types
137	Minimal Unique Types
138	Compiling Scheme programs to executables
139	Syntax parameters
140	Immutable Strings
141	Integer division
143	Fixnums
144	Flonums
145	Assumptions
146	Mappings
147	Custom macro transformers
148	Eager syntax-rules
149	Basic Syntax-rules Template Extensions
150	Hygienic ERR5RS Record Syntax (reduced)
151	Bitwise Operations
152	String Library (reduced)
156	Syntactic combiners for binary predicates
157	Continuation marks
158	Generators and Accumulators
159	Combinator Formatting

Table 3. Finalized SRFIs as of this writing

ACKNOWLEDGMENTS

The author hereby thanks the organizers of all Scheme Workshops since the first, as well as everyone who has attended; all the previous SRFI editors (Donovan Kolbly, Shriram Krishnamurthi, Dave Mason, David Rush, Francisco Solsona, Mike Sperber, and David Van Horn); all SRFI authors; everyone who has contributed to SRFI discussions and implementations; everyone who has participated in the IEEE Scheme and R^{n}RS standards processes, especially the editors; Guy L. Steele and Prof. Gerald J. Sussman, the inventors of the language; and especially the author's beloved wife and daughter.

REFERENCES

[1] Marc Battyani, Pascal Costanza, Arthur Lemmens, and Edi Weitz. 2013. CDR - Common Lisp Document Repository. Retrieved 2018-7-5 from https://common-lisp.net/project/cdr//

[2] Zach Beane. 2018. Quicklisp beta. Retrieved 2018-7-5 from https://www.quicklisp.org/beta/

[3] Per Bothner. 2017. Kawa page. Retrieved 2018-7-3 from https://www.gnu.org/software/kawa/Implemented-SRFIs.html

[4] Will Clinger. 2017. Larceny SRFI page. Retrieved 2018-8-23 from https://github.com/larcenists/larceny/tree/master/lib/SRFI/srfi

[5] R6RS Editors. 2007. R6RS. Retrieved 2018-8-26 from http://www.r6rs.org/

[6] R6RS Editors. 2007. R6RS. Retrieved 2018-8-26 from http://www.r6rs.org/final/r6rs-rationale.pdf

[7] R7RS Editors. 2007. R7RS. Retrieved 2018-8-26 from https://bitbucket.org/cowan/r7rs/raw/4c27517de187142ad2cf4bcd8cb9199ae1e48c09/rnrs/r7rs.pdf

[8] R7RS Editors. 2018. R7RS home page. Retrieved 2018-7-5 from https://bitbucket.org/cowan/r7rs-wg1-infra/src/default/R7RSHomePage.md?fileviewer=file-view-default

[9] Felix Winkelmann et al. 2018. Chicken Scheme page. Retrieved 2018-7-3 from http://wiki.call-cc.org/SRFI-conformance

[10] Felix Winkelmann et al. 2018. Eggs Unlimited (release branch 4). Retrieved 2018-8-23 from http://wiki.call-cc.org/chicken-projects/egg-index-4.html

[11] Marc Feeley et al. 2010. Gambit page. Retrieved 2018-7-3 from http://gambitscheme.org/wiki/index.php/SRFI:s

[12] Scott Chacon et al. 2018. Git home page. Retrieved 2018-7-5 from https://git-scm.com/

[13] Marc Feeley. 2005. [R6RS] SRFIs for R6RS. Retrieved 2018-7-5 from http://www.r6rs.org/r6rs-editors/2005-May/000557.html

[14] Free Software Foundation. 2017. Guile page. Retrieved 2018-7-6 from https://www.gnu.org/software/guile/manual/html_node/SRFI-Support.html

[15] Free Software Foundation. 2018. The Emacs Editor: 15.1 Incremental Search. Retrieved 2018-7-5 from https://www.gnu.org/software/emacs/manual/html_node/emacs/Incremental-Search.html

[16] Abdulaziz Ghuloum. 2008. Ikarus page. Retrieved 2018-7-3 from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.130.9554&rep=rep1&type=pdf

[17] Arthur A. Gleckler. 2018. Github home page for Scheme Requests for Implementation. Retrieved 2018-7-5 from http://github.com/scheme-requests-for-implementation/

[18] Arthur A. Gleckler. 2018. SRFI Data. Retrieved 2018-7-5 from https://github.com/scheme-requests-for-implementation/srfi-common/blob/master/admin/srfi-data.scm

[19] Arthur A. Gleckler, David Van Horn, and Mike Sperber. 2017. SRFI History. Retrieved 2018-7-5 from https://srfi.schemers.org/srfi-editors.html

[20] Arthur A. Gleckler, David Van Horn, and Mike Sperber. 2018. Scheme Requests for Implementation document template. Retrieved 2018-7-5 from https://srfi.schemers.org/srfi-template.html

[21] Arthur A. Gleckler, David Van Horn, and Mike Sperber. 2018. Scheme Requests for Implementation home page. Retrieved 2018-7-2 from https://srfi.schemers.org/

[22] Arthur A. Gleckler, David Van Horn, and Mike Sperber. 2018. SRFI Editors. Retrieved 2018-7-5 from https://srfi.schemers.org/srfi-history.html

[23] Chris Hanson. 2018. MIT Scheme home page. Retrieved 2018-7-3 from https://www.gnu.org/software/mit-scheme/

[24] David Van Horn, Donovan Kolbly, Mike Sperber, and Arthur A. Gleckler. 2018. Scheme Requests for Implementation Process. Retrieved 2018-7-5 from https://srfi.schemers.org/srfi-process.html

[25] Aaron W. Hsu. 2018. Chez page. Retrieved 2018-7-3 from https://github.com/arcfide/chez-srfi

[26] Github Inc. 2018. Github Help: About pull requests. Retrieved 2018-7-5 from https://help.github.com/articles/about-pull-requests/

[27] Takashi Kato. 2018. Sagittarius Scheme Users' Reference, Chapter 9, Supporting SRFIs. Retrieved 2018-8-23 from http://ktakashi.github.io/sagittarius-online-ref/section9.html

[28] Shiro Kawai. 2006. Scheme48 entry on practical-scheme.net. Retrieved 2018-7-13 from https://practical-scheme.net/wiliki/schemexref.cgi?Scheme48

[29] Shiro Kawai. 2007. SCM entry on practical-scheme.net. Retrieved 2018-7-13 from https://practical-scheme.net/wiliki/schemexref.cgi?SCM

[30] Shiro Kawai. 2007. SigScheme entry on practical-scheme.net. Retrieved 2018-7-13 from https://practical-scheme.net/wiliki/schemexref.cgi?SigScheme

[31] Shiro Kawai. 2012. Mosh entry on practical-scheme.net. Retrieved 2018-8-23 from https://practical-scheme.net/wiliki/schemexref.cgi?Mosh

[32] Shiro Kawai. 2012. STklos entry on practical-scheme.net. Retrieved 2018-7-13 from https://practical-scheme.net/wiliki/schemexref.cgi?STklos

[33] Shiro Kawai. 2012. Ypsilon entry on practical-scheme.net. Retrieved 2018-7-13 from https://practical-scheme.net/wiliki/schemexref.cgi?Ypsilon
[34] Shiro Kawai. 2014. SLIB entry on practical-scheme.net. Retrieved 2018-7-13 from https://practical-scheme.net/wiliki/schemexref.cgi?SLIB
[35] Shiro Kawai. 2018. Gauche page. Retrieved 2018-8-23 from http://practical-scheme.net/gauche/man/gauche-refe/Standard-conformance.html#Standard-conformance
[36] Richard Kelsey. 1998. Preliminary Call for Participation, Scheme Strawman Workshop. Retrieved 2018-7-2 from https://web.archive.org/web/19990428131650/http://www.neci.nj.nec.com/homepages/kelsey/workshop.html
[37] Mike Montague. 2017. Foment. Retrieved 2018-7-5 from https://github.com/leftmike/foment/wiki/Foment
[38] Derek Robert Price and Ximbiot. 2015. CVS - Concurrent Versions System. Retrieved 2018-7-5 from https://www.nongnu.org/cvs/
[39] et al. Richard Kelsey, Jonathan Rees. 2001. Scheme48 manual (1.9.2). Retrieved 2018-8-23 from http://s48.org/
[40] Manuel Serrano. 2017. Bigloo page. Retrieved 2018-7-3 from https://www-sop.inria.fr/indes/fp/Bigloo/bigloo-1.html
[41] Alex Shinn. 2018. personal communication.
[42] Alex Shinn. 2018. Chibi Scheme page. Retrieved 2018-7-3 from https://github.com/ashinn/chibi-scheme/tree/master/lib/srfi
[43] Alex Shinn. 2018. Snow! Retrieved 2018-7-5 from http://snow-fort.org/
[44] Olin Shivers, Brian D. Carlstrom, Martin Gasbichler, and Mike Sperber. 2006. SCSH page. Retrieved 2018-7-3 from https://scsh.net/docu/docu.html
[45] Michael Sperber. 2018. personal communication.
[46] Michael Sperber. 2018. personal communication.
[47] Chicken team. 2018. Eggs Unlimited. Retrieved 2018-7-5 from http://wiki.call-cc.org/chicken-projects/egg-index-4.html
[48] Racket team. 2018. Racket. Retrieved 2018-7-5 from https://racket-lang.org/
[49] Racket team. 2018. Racket Packages. Retrieved 2018-7-5 from https://pkgs.racket-lang.org/
[50] Racket team. 2018. Racket page. Retrieved 2018-8-23 from https://docs.racket-lang.org/srfi/

Received July 2018; accepted August 2018