Scheme Reports Working Group 1 Progress

Alex Shinn
alexshinn@gmail.com

1. Introduction

Earlier revisions of the Scheme standard were made by small groups of established implementors, where consensus was required to make any change, thus ensuring a conservative standard. The R6RS broke the convention by requiring only a simple majority for a feature to be added, while at the same time greatly expanding its scope. This resulted in a very large standard with many improvements and clarifications on previous standards, but at the same time some mistakes and questionable design decisions. This combined with the sheer size of R6RS resulted in a backlash from the community - many implementors simply refused to provide R6RS compatibility, factioning the community.

In their wisdom, the Steering Committee decided that the next standard would be split into two separate but compatible languages. One would be a small language, very close to R5RS but addressing some of its deficiencies and providing a module system; the other would be a large language, a superset of the small language, similar to R6RS in size and scope. The size of the small language would not only appeal to purists, but leave less room for newly introduced bugs. The large language would appeal to people who want to be able to write in a language with a broader set of features.

Working group 1 was formed to create the small language specification, suitable for use in education, programming-language research, embedded systems and extension languages. The language should be mostly compatible and comparable in size with the R5RS, in particular following the design precept found in the R5RS introduction:

> Programming languages should be designed not by piling feature on top of feature, but by removing the weaknesses and restrictions that make additional features appear necessary.

This quotation is often overused in debates among Schemers to attack anything new which they dislike, but that doesn't make it any less important a precept. We are tasked with designing a language, not collecting a list of features, and we must consider the consequences of any changes we make. It's much easier for later standards to add new features than to remove them.

2. Process

Compared to previous standards, there are relatively few implementors in working group 1. This is arguably a good thing, because users drive the future of scheme, and because they will hopefully be more objective and less tied to their implementation's way of doing things. To address the concern that we may make design mistakes owing to our lack of experience, we are requiring proof of implementation for any change, and will also count on the fact that the eyes of the entire Scheme community will be reviewing the finished result.[1]

Also different from previous standards, we are using a competitive proposal system. Members are free to make proposals on any issue, written up on a publicly viewable wiki, or to propose existing specifications such as SRFIs, and to propose any number of variations as well. Votes are recorded on the wiki[2] as a simple ranking of each member's preferences, in order. The

2010 Workshop on Scheme and Functional Programming August 21-22, 2010, Montreal, Quebec, Canada.

[1] WG2 will deal with larger and more experimental changes, for which the members may wish to make use of the SRFI process as was done during the R6RS drafting.

[2] Votes are optionally accepted via email.

results are tallied using the ranked pairs method, so no strategy is involved. This allows for multiple proposals to be put forth without detracting from each other or forcing an all-or-nothing situation.

In addition, many simple issues that don't warrant a full proposal would typically be yes/no votes, but the preferential system allows for easy addition of options on the fly. For instance, most issues allow default options of "module" means "yes, but I think it should be in a separate module" and "wg2" meaning "no, but I think it should be included in WG2."

At different points, calls for final votes will be made on batches of issues where discussion has died down or no progress seems to be being made. Once finalized, the winning proposals will be written up formally and included in the draft standard.

A sample of the voting process is given in the appendix.

3. Issues

From past standards, SRFIs, and comments on R6RS, a list of issues was logged into a bug tracking system. There are currently about 70 issues, and it is anticipated that over 100 issues will be logged before the final draft. For the major issues we underwent a fact-finding period, detailing the background behind the issues on the wiki, before proceeding to write proposals.

At that point, from the issues and proposals an initial ballot of 47 items[3], was created, and voting has confirmed consensus on about 30 of the items. Email discussion continues on the other items, and new items will be added as the originals are finalized.

A summary of the most notable issues under discussion follows.

3.1 Major Controversial Issues

These issues are still a matter of active debate and the proposals may change substantially before standardization.

- **Module System -** static R6RS-like or syntactic Chez-like

  A very natural approach to modules is to treat *module* and *import* as new special forms, similar to **lambda**, which can be composed in any manner and expanded from macros. This would give us a more expressive module system than R6RS, at the expense of precluding alternative approaches to module systems and risking less adoption.

  The R6RS module system, on the other hand, described only the static syntax of a module, leaving implementation details unspecified. This is a natural compromise because it's easy for any Scheme to support, so the alternate module system proposal calls for a similar static system to R6RS, without versions, and minor adjustments leaving room for future extension.

- **User-Defined Types -** procedural or syntactic

  The ability to create new types or records is unanimously desired, but there is disagreement on whether the means of doing so should be procedural or syntactic. On the one hand, syntactic forms could be built on top of procedural forms, on the other hand purely syntactic forms could in some cases be more efficient and could integrate easier with foreign object systems.

  This same debate occurred among the R6RS editors, with the unfortunate result of providing both record types in such a way that they could not always be interchanged. Most WG1 members are opposed to providing both on the grounds that it's not minimalist.

- **Parameters -** mutable or immutable

  SRFI-39 style parameters (first class dynamic bindings) are generally desired, and are typically defined as the mechanism by which *current-input-port*, etc. are provided and are hence a likely candidate for WG1. Even though WG1 will not be specifying threads or any other form of concurrency, it is felt that the behavior of parameters in the presence of concurrent code should be specified up front. The debate is whether parameters should be mutable (beyond *parameterize*) or not, and if so are the mutations seen between separate threads.

- **Binary Data -** blobs or uniformly-typed vectors

  Binary data is essential for FFIs, efficient binary I/O, and compact storage of data. The two general approaches are to provide a number of different homogeneous vector types as in SRFI-4, or to provide a single abstract blob type with multiple accessors for extracting different sized values from offsets into it. A hybrid solution has also been proposed, with a base blob type on which you can layer homogeneous vectors.

[3] Later increased to 51.

- **Hash Tables -** R6RS or SRFI-69

  SRFI-69 hash tables have the known issue that they provide no way to use address-based hash functions for *eq?* comparisons in a moving GC. The R6RS hash tables address this by making *eq?* and *eqv?* hash tables special, and hiding the hash function used. Neither of these are particularly popular, but this is a non-essential feature and easy to push to WG2.
- **SRFI-38 -** reading and writing shared structures

  SRFI-38 provides procedures to read and write shared structures with the common `#0=(1 . #0#)` notation. We're still undecided whether these should be provided, dropped, or required for the core read/write.

3.2 Major Uncontroversial Issues

These are areas we've been able to agree on.

- **Exceptions**

  There were originally two exception proposals, which went through a number of changes before converging to a single proposal. Although the cooperation is encouraging, the resulting proposal is too minimal for direct use and is wanting for syntactic convenience forms.

  current-error-port and *error* will be provided.

  The types of exceptions and corresponding accessors is still under active discussion. Restarts will likely be a topic for WG2.
- **Character Set**

  We have a well thought-out Unicode proposal which specifies the Unicode semantics of character and string operations in the presence of Unicode, without requiring full Unicode support.
- **Basic File System Operations**

  file-exists? and *delete-file* are essential for any file-oriented programming. There was strong consensus that they be included, though in a separate module.
- **syntax-rules escapes**

  The common template escape of (... *x*) to expand into just *x* will be accepted.

  SRFI-46 style escapes as well as tail patterns are still undecided.
- ***syntax-error***

  A frequent complaint against **syntax-rules** is the inability to provide meaningful compile-time errors, so the addition of *syntax-error* is welcome by most members.
- **Internal define-syntax**

  There's no good reason to allow internal define but not internal define-syntax. This is a case where it's a good thing most members are users - the only negative vote was from an implementor who didn't want to complicate his implementation.
- ***letrec****∗*

  Taken directly from R6RS and generally approved, this will also be used to specify the semantics of internal **define**.

3.3 Syntactic Changes

Relatively minor issues, but syntax changes stand out.

- **Nested and Sexp Comments**

  Both SRFI-30 style #|nested comments |# and SRFI-62 style #; sexp-comments will be added to the language.
- **No Brackets**

  The left and right bracket characters will remain reserved, which means you can use them to whatever purpose in your own implementation but should avoid them in portable code.
- **Character Literal Syntax**

  New character names such as #\tab in addition to the existing #\space and #\newline will be added, along with numeric escapes. The names are convenient and widely supported already, and the numeric escapes useful and well-defined in the context of our planned Unicode friendliness. The exact list of mnemonics has yet to be determined.
- **String and Symbol Literal Syntax**

  By the same rationale as the characters, strings and symbols will have new escape syntax for mnemonics such as `"\t"` and numeric escapes.
- **Case-Sensitivity**

  This is currently a close vote, though if finalized at the time of writing the language would be case-sensitive. Methods of changing this behavior are still a matter of debate.

3.4 Compatibility with WG2

These are features of the large language which may require cooperation from the small language, and need to be kept in mind.

- **Module Phasing**

With only **syntax-rules**, phasing is a moot issue, but assuming the addition of low-level macros we may need some way to provide for control of phase distinctions in the module syntax.

- **Conditional Compilation**
  R6RS implementations quickly realized that even a large standard wasn't enough for portability, and came up with a hackish mechanism of using file names to choose implementation-specific code. An alternative would be something like SRFI-0's *cond-expand*, with implementation names as features, possibly restricted to the module top-level.
- **Threads**
  WG2 is likely to provide threads. As mentioned in the parameters discussion, concurrency has far-reaching implications and the consequences need to be considered in advance.
- **Binary I/O**
  Binary I/O is likely to introduce ports for which the R5RS character operations are undefined. In addition, the types of *current-input-port* and *current-output-port* and means by which those types may be converted have implications for WG1.
- **Keywords**
  This is an issue that comes up often. It is likely WG2 will introduce some way to handle named function parameters. This may involve any of a new type, new syntax, or extensions to the **lambda** form.

4. Future Work

With the major issues cataloged and researched, and concrete proposals under discussion, the next step will be to begin finalizing votes on batches of issues so they can be formally specified in the initial draft, due in another 6 months. With a concrete draft available for review we expect a large amount of feedback from the community, using the last 6 months of the effort for bug-fixes and editing leading up to the final draft.

We will also provide working implementations of all proposals, including most likely a meta-circular evaluator and full support from a small Scheme implementation.

A. Voting Sample

As an example vote we can consider the current ballots for the user-defined types proposals. There are six options:

- SRFI-9 - the SRFI-9 define-record-type syntax
- SRFI-99 - the SRFI-99 extensions to SRFI-9
- Hsu - a syntactic proposal by one of the WG1 members
- SnellPym - a procedural proposal by another WG1 member
- WG2 - a vote to leave this issue to WG2
- None - Scheme should not have user-defined types

WG1 is a large group with 16 members. Of these, there are currently 8 active voters, one of whom abstained on this issue. The remaining seven votes were

```
(define votes
 '((srfi-99 srfi-9 hsu snellpym)
   (hsu)
   (hsu)
   (snellpym srfi-9)
   (srfi-9 srfi-99)
   (snellpym hsu srfi-9 wg2)
   (snellpym srfi-9 hsu)))
```

where for each user the unlisted options are all given equal precedence after the last listed option. We then compute a tally of the number of times each option is preferred over each other option, in this case:

```
'(((srfi-9 . hsu) . 3)
  ((srfi-9 . snellpym) . 4)
  ((srfi-9 . srfi-99) . 4)
  ((srfi-9 . wg2) . 4)
  ((hsu . srfi-9) . 1)
  ((hsu . snellpym) . 3)
  ((hsu . srfi-99) . 3)
  ((hsu . wg2) . 3)
  ((snellpym . srfi-9) . 0)
  ((snellpym . hsu) . 0)
  ((snellpym . srfi-99) . 1)
  ((snellpym . wg2) . 1)
  ((srfi-99 . srfi-9) . 1)
  ((srfi-99 . hsu) . 1)
  ((srfi-99 . snellpym) . 1)
  ((srfi-99 . wg2) . 1)
  ((wg2 . srfi-9) . 0)
  ((wg2 . hsu) . 0)
  ((wg2 . snellpym) . 1)
  ((wg2 . srfi-99) . 1))
```

We then sort these tallies, and lock in turn each pair ranking the winner over the loser, unless this would result in a circularity in the locked pairs so far. This results in the following graph

```
'((srfi-9 hsu srfi-99 wg2 snellpym)
  (hsu srfi-99 wg2 snellpym)
  (snellpym wg2 srfi-99)
  (srfi-99 wg2))
```

which when topologically sorted gives the current linearized ranking of the options:

```
'(srfi-9 hsu snellpym srfi-99 wg2)
```

Thus if the vote were finalized at this point, SRFI-9 would be chosen as the record system for WG1, and written up in the draft standard.

Acknowledgments

Thanks to the members of the Steering Committee, Arthur Gleckler for his early work and preparation of the group, the chair of WG2 John Cowan, as well as all the members of WG1.