R^{7}RS Large

Status Report

Daphne Preston-Kendal
European Lisp Symposium 2024
6 May 2024

The story so far …

- Report on Scheme (1975), Revised Report (1978), Sussman & Steele
- R^{n}RS specifications up to R^{5}RS (1998) devised by unanimous consensus
- R^{6}RS (2007) not as widely adopted, frequently criticized
- 2009: Scheme Steering Committee resolves to split the language in two
- R^{7}RS Small (2013) more conservative revision of R^{5}RS, adopted in some form by all maintained R^{5}RS implementations

R^{7}RS Small (2013)

Headline features compared to R^{5}RS

- Portable library system (R^{6}RS interoperable)
- Exception raising and handling (R^{6}RS compatible)
- Record type definitions (R^{6}RS interoperable)
- Clean(ish) split between binary and textual (Unicode) I/O (R^{6}RS interoperable)
- Parameters (variable-like boxes with dynamic scope)

R^{7}RS Large

- Began 2014
- 2022 split into two or three parts:
  - Foundations: core language semantics, hopefully done by end of 2025
  - Batteries: useful standard libraries that are unlikely to go out of date, finished soon after
  - Environments: OS interfaces, no target completion date yet
- ‘R^{6}RS compatible’

Recent work: Macros

R^{5}RS macros

```
(define-syntax swap!
  (syntax-rules ()
    ((_ x y)
     (let ((temp x))
       (set! x y)
       (set! y temp)))))
```

- Pattern matching and substitution with fully automatic hygiene
- Good for simple macros; otherwise a Turing tarpit (no compile-time Scheme evaluation)
- No way to choose to break hygiene

R^{6}RS macros

```
(define-syntax swap!
  (lambda (stx)
    (syntax-case stx ()
      ((_ x y)
       #'(let ((temp x))
           (set! x y)
           (set! y temp))))))
```

- `syntax-case`, extension of `syntax-rules` allowing interleaving Scheme evaluation with pattern-based expansion

R⁶RS macros

Breaking hygiene

```
(define-syntax with-return
  (λ (stx)
    (syntax-case stx ()
      ((k body₀ body₁ …)
       (let ((return-id
               (datum->syntax #'k 'return)))
         #`(call/cc
             (λ (#,return-id)
               body₀ body₁ ...)))))))
```

R^{6}RS macros

Identifier macros

```
(define-syntax fast-concatenate
  (λ (stx)
    (syntax-case stx (map)
      ((_ (map f ls_0 ls_1 ...))
       #'(append-map f ls_0 ls_1 ...))
      ((_ ls)
       #'(concatenate ls))
      (id
       (identifier? #'id)
       #'concatenate))))
```

Criticisms of R⁶RS `syntax-case`

- Pattern matching as the only portable way to destructure macro input
- High-level syntactic system with no low-level procedural counterpart
  - R7RS Large solution: `unwrap-syntax` procedure
- Identifier macros mean macros cannot tell whether identifiers they receive are variables or macros
  - R6RS and R7RS editors’ reply: code walking macros are inherently broken; identifier syntax which doesn’t behave like a variable is bad style anyway
- Others we don’t understand: too ‘large’, reader extensions, etc.

New in R^{7}RS Large: Identifier properties

- Like classical Lisp symbol properties, but respect lexical scoping and the library system
    - Properties imported when their corresponding libraries are exported
    - Properties have full lexical shadowing behaviour
- Available only at expand time (but also by extension through `eval`)

Identifier property use cases

- Attach information to bindings which useful to programmers, e.g. documentation and debug info
- Families of macros which communicate information to one another
- Establish context-specific usages for identifiers
  - With identifier properties and `unwrap-syntax`, `syntax-case` can be expressed portably in terms of lower level primitives for the first time
  - Racket `match` alike (like Emacs Lisp `pcase`) with extensible patterns

New in R^7RS Large: Syntax parameters

```
(define-syntax-parameter return
  (erroneous-syntax "return must be used inside with-return"))

(define-syntax with-return
  (syntax-rules ()
    ((_ body0 body1 ...)
     (call/cc
       (λ (return-proc)
         (syntax-parameterize
             ((return (identifier-syntax return-proc))
           body0 body1 ...)))))))
```

- An alternative to fully breaking hygiene: adjust an existing, known transformer binding shared between macro author and macro user

Future work

Procedural and Valued Fascicles

- Storage management: ephemerons (the ‘right’ weak pair primitive) and guardians (quasi-deterministic, generation-friendly finalization)
- Challenge: compaction-friendly hash-based data structure primitives for `eq?` and friends

Looking further on

- Condition system including guaranteed exception raising (small language has *lots* of undefined behaviours)
  - Maybe restarts like Common Lisp
- Delimited control operators – complementing and/or extending `call/cc`, not replacing it
- Maybe threading (hard to require on some platforms)

Foundations challenges

- User enthusiasm for a larger core portable Scheme language is high
- Implementer enthusiasm: ???
- Volunteer effort

Batteries

- Data structures, algorithmic primitives, etc. you expect in a functional language in `(current-year)`
- Portable in terms of the Foundations – ‘alternative preludes’ encouraged
- Conservative in scope

Environments

- Scope and overall approach still very unclear
- Potentially a huge project
- Must-haves in my own personal view (not necessarily others'):
  - TCP networking – hopefully simple TLS too
  - Cross-platform pathname and filesystem stuff
  - Some limited process control

Question Time

dpk@nonceword.org