R5RS: Legacy Scheme

Version 6.8

January 24, 2017

The The Revised^5 Report on the Algorithmic Language Scheme defines a dialect of Scheme. We use *R^{5}RS* to refer to both the standard and the language defined by the standard.

The default dialect of Lisp provided by `racket` and other Racket tools differs from R^{5}RS in many ways, but Racket includes tools and libraries for running R^{5}RS programs.

See §23 “Dialects of Racket and Scheme” for general information about different dialects of Scheme within Racket.

Contents

1 Running R^{5}RS Programs

Racket provides several layers of support for programs written according to R^{5}RS:

- DrRacket provides an R5RS language, which can be selected via the Language|Choose Language... menu item. See Choose Language... in the DrRacket documentation for more information.
- The `plt-r5rs` executable runs an R^{5}RS program or provides a read-eval-print loop for evaluating R^{5}RS expressions and definitions. See §2 "`plt-r5rs`" (later in this manual) for more information.
- The `r5rs` library implemented R^{5}RS procedures and syntactic forms. It can also be used with `#lang` to create a module whose body is implemented in an R^{5}RS-like language. See §3 "R^{5}RS Module Language" (later in this manual) for more information.
- The `r5rs/init` library extends `r5rs` to set parameters (such as case-insensitive symbol reading) for R^{5}RS loading or an R^{5}RS read-eval-print loop. See §4 "R^{5}RS Initialization Library" (later in this manual) for more information.

2 plt-r5rs

The `plt-r5rs` executable runs an R^{5}RS program from a file that is supplied on the command line. If no program file is provided as a command-line argument, then a read-eval-print loop is started.

Before starting a read-eval-print loop, an initialization file is loaded, if it exists. The file is the same as the file reported by `(find-system-path 'init-file)`, but with the characters `racket` in the filename replaced by `pltr5rs`. For example, on Unix, the file is `"~/.pltr5rsrc"`.

By default, `plt-r5rs` departs from R^{5}RS conformance in one crucial way: the names of pre-defined functions cannot be redefined at the top level. This restriction enables better run-time performance. Use the `--no-prim` command-line flag—before a file to load, if any—to obtain the standard behavior for primitive bindings (at the cost of performance).

3 R5RS Module Language

`#lang r5rs` package: `r5rs-lib`

As a library, `r5rs` provides the syntactic forms and procedures defined by R5RS. When used as a language via `#lang`, the program is read with the following parameterizations:

```
(read-case-sensitive #f)
(read-accept-infix-dot #f)
(read-curly-brace-as-paren #f)
(read-square-bracket-as-paren #f)
```

The `r5rs` bindings can be imported into a top-level environment, and then evaluation in that top-level environment corresponds to R5RS. Use `(namespace-require/copy 'r5rs)` with an empty namespace to maximize conformance with R5RS; Using `(namespace-require 'r5rs)`, in contrast, creates primitive bindings as imports, which is the same as using `plt-r5rs` without the `--no-prim` flag. More simply, use `(scheme-report-environment 5)`. See also `r5rs/init`, which sets reader and printer parameters to increase conformance.

Using `r5rs` via `#lang` creates a module whose body is implemented with an R5RS-like language. The main difference from R5RS is that, as a module language, `r5rs` does not allow redefinition of top-level bindings, and expressions evaluated through `load` and `eval` cannot automatically access bindings defined within the module.

Changed in version 6.0.1.4 of package `r5rs-lib`: When an identifier bound by `letrec` is referenced before it is initialized, an exception is raised, instead of producing `#<undefined>`.

3.1 Non-R5RS Bindings from `r5rs`

In addition to the bindings defined by R5RS, the `r5rs` library provides the following bindings from `racket/base` (which are not legal identifiers in R5RS syntax, so there is no danger of collisions in R5RS programs):

```
#%app #%datum #%top #%top-interaction #%require #%provide
```

It also provides `racket`'s `#%plain-module-begin` as `#%module-begin`. Note that `#%require` can be used to import Racket libraries into an otherwise R5RS program, and `#%provide` can be used to export from a module that is implemented in an R5RS-like language.

3.2 Notes on R^{5}RS Functions

The `cons` of `r5rs` corresponds to `racket/base`'s `mcons`. Similarly, `cdr` is `mcdr`, and `map` is `compatibility/mlist`'s `mmap`, and so on.

An R^{5}RS *environment* is implemented as a `racket/base` *namespace*. Also, relative to `racket/base`, the `expr` passed to `eval` is constructed using mutable pairs.

The `scheme-report-environment` function returns a namespace containing the bindings of `r5rs`. Procedure values are installed into the namespace using `namespace-require/copy`, so that they can be redefined.

The `null-environment` function returns a namespace containing the syntactic forms of `r5rs`, not including `#%module-begin` (which is not useful outside of a module).

4 R5RS Initialization Library

`(require r5rs/init)` package: `r5rs-lib`

The `r5rs/init` module re-exports `r5rs`, and also sets parameters as follows:

```
(read-case-sensitive #f)
(read-accept-infix-dot #f)
(read-curly-brace-as-paren #f)
(read-square-bracket-as-paren #f)
(print-mpair-curly-braces #f)
```

The side-effect of setting these parameters is useful when the module is `required` before loading an R5RS program, so that the reader and printer behave more as specified in R5RS. In particular, the `plt-r5rs` executable initializes by importing `r5rs/init`.