APPENDIX

Nonstandard Extensions to Scheme

We presume the existence of the following predefined procedures, which are not part of the R^{4}RS standard for Scheme:

`(error` *string value*...`)` Signals an error to the user in some form. The *string* should be a description of the error. There can be any number of *values*, and they are also displayed to the user to further describe what went wrong.

`(filled-triangle` x_0 y_0 x_1 y_1 x_2 y_2`)` Produces a standard-sized square image containing a filled-in triangle with vertices (x_0, y_0), (x_1, y_1), and (x_2, y_2). The coordinate system for the vertices ranges from $(-1, -1)$ in the lower left corner of the image to $(1, 1)$ in the upper right corner.

`(invert` *image*`)` Produces a new image of the same size as *image* and with the same contents as *image* except that black and white are reversed.

`(line` x_0 y_0 x_1 y_1`)` Produces a standard-sized square image containing a line segment from (x_0, y_0) to (x_1, y_1). The coordinate system and size are the same as for `filled-triangle`.

`(overlay` *image*$_1$ *image*$_2$`)` Produces a new image with the same size as *image*$_1$ and *image*$_2$, which must have the same width and height as each other. The contents of the new image is formed by combining the contents of the two existing images, much as though two transparencies were laid together.

`(quarter-turn-right` *image*`)` Produces a new image with the contents of *image* turned 90 degrees clockwise. The width of the new image is the same as *image*'s height, and the height of the new image is the same as *image*'s width.

`(random` *n*`)` Produces a nonnegative integer, chosen in a pseudo-random fashion from the range from 0 up to but not including *n*. The argument *n* must be an exact positive integer. The *n* possible values are returned equally frequently over the long run.

`(stack` $image_1$ $image_2$`)` Produces a new image by stacking the contents of $image_1$ on top of the contents of $image_2$. The width of $image_1$ must equal the width of $image_2$, which becomes the width of the resulting image. The height of the resulting image is the sum of the heights of $image_1$ and $image_2$.

In addition to presuming the above non-R^{4}RS procedures, we make use of several features specified in the R^{4}RS as being "inessential." In other words, these are features that the standard describes but does not require all implementations to provide. We list below the inessential features we use, with some comments on how common it is for an implementation to omit each feature and what impact such an omission would have on using this book:

internal definitions The R^{4}RS permits an implementation to not support nested definitions inside the body of a lambda expression or let expression. Such an implementation would be awkward to use with this book, because we use internal definitions freely. However, such implementations are very rare.

disjointness of `#f` and `()` The R^{4}RS allows a single value to be used as both false and the empty list. It is relatively common for Scheme implementations to make this choice. All our examples will work in such implementations, with the minor exception that wherever we show `#f` in the output, `()` will be displayed instead. (In input, `#f` should still be used.)

exact rationals Not every implementation needs to support exact rational numbers. This area is the one that is likely to cause the most trouble because relatively many implementations have opted out of exact rationals and we use them moderately freely in the early chapters. The book can be used with such an implementation, however, provided you are willing to work around a few difficulties as you encounter them. For example, in most systems that only have inexact "floating point" numbers, you will not be able to get an approximation to the golden ratio that is good to one part in 10^{79} and will get an infinitely looping process if you try. However, simply using a more tolerant tolerance will solve this problem.

the procedures `sqrt`, `expt`, `denominator`, `list-tail`, `vector-fill!`, and `with-output-to-file` These predefined procedures are all labeled "inessential" by the R^{4}RS. Nearly all implementations include `sqrt` and `expt`, so they are unlikely to present a problem. The `denominator` procedure is used only in approximating the golden ratio; it is only likely to be missing in implementations that don't have exact rationals, and in such a case, the exercise could simply be omitted. We define `list-tail` and `vector-fill!` in the text, so it wouldn't matter if they weren't predefined. On the rare system that doesn't support `with-output-to-file`, the examples using it can be rewritten to use the essential procedure `call-with-output-file`.