Exercise 3.63.

Louis Reasoner asks why the `sqrt-stream` procedure was not written in the following more straightforward way, without the local variable `guesses`:

```
(define (sqrt-stream x)
  (cons-stream 1.0
               (stream-map (lambda (guess)
                             (sqrt-improve guess x))
                           (sqrt-stream x))))
```

Alyssa P. Hacker replies that this version of the procedure is considerably less efficient because it performs redundant computation. Explain Alyssa's answer. Would the two versions still differ in efficiency if our implementation of delay used only `(lambda () <exp>)` without using the optimization provided by `memo-proc` (section 3.5.1)?

Answer.

In our original `sqrt-stream` procedure, we made these guesses be a stream whose first element is 1 and the rest of which are the successive improved guesses. This implicit definition works because, at any point, enough of the `guesses` stream has been generated so that we can feed it back into the definition to produce the next guess. Remember that we formerly have exploited this strategy to construct the `integers` and `fibs` stream in section 3.5.2. For example, the `sqrt-stream` of 2 is generated in the following process:

	1	1.5	1.41666666	1.41421568	1.41421356	...	= guess
	1.5	1.41666666	1.41421568	1.41421356	1.41421356	...	= (stream-cdr guesses)
1	1.5	1.41666666	1.41421568	1.41421356	1.41421356	...	= guess

Louis's `sqrt-stream` program however, maps the `sqrt-improve` procedure successively onto itself to get better and better guesses. Thus, `(sqrt-stream 2)` is defined to be a stream whose first element is 1 and the rest of which is the improvement of itself shifted by one place. To capture the image of itself shifed by one place, Louis forces `sqrt-stream` to call to itself once again. Notice that here what returned by `cons-stream` is simply the number 1 together with a promise to improve its `stream-cdr`. No other information about the stream will be preserved in every reduction step. Hence, anyone other than the first element in the stream will be reproduced starting from 1 again and again and this leads to redundant computation:

							1	...	= (sqrt-stream 2)
						1	1.5	...	= (sqrt-stream 2)
					1	1.5	1.41666666	...	= (sqrt-stream 2)
				1	1.5	1.41666666	1.41421568	...	= (sqrt-stream 2)
			1	1.5	1.41666666	1.41421568	1.41421356	...	= (sqrt-stream 2)
		⋰	⋮	⋮	⋮	⋮	⋮		⋮
1	1.5	...	...	...	...	...	...	...	= (sqrt-stream 2)

Once the optimization provided by `memo-proc` was eliminated, our original `sqrt-stream` procedure would lost all its formerly computed results and arise the same redundant computation as Louis's procedure does.

Email address: informlarry@gmail.com