Exercise 5.3.

Design a machine to compute square roots using Newton's method, as described in section 1.1.7:

```
(define (sqrt x)
  (define (good-enough? guess)
    (< (abs (- (square guess) x)) 0.001))
  (define (improve guess)
    (average guess (/ x guess)))
  (define (sqrt-iter guess)
    (if (good-enough? guess)
        guess
        (sqrt-iter (improve guess))))
  (sqrt-iter 1.0))
```

Begin by assuming that `good-enough?` and `improve` operations are available as primitives. Then show how to expand these in terms of arithmetic operations. Describe each version of the `sqrt` machine design by drawing a data-path diagram and writing a controller definition in the register-machine language.

Answer.

By assuming that `good-enough?` and `improve` are available as primitive operations, we can describe Newton's method concisely in a register machine, as figure 1 shows. The controller sequence for this `sqrt` machine can be described as follows:

```
(controller
   (assign guess (const 1))
 test-guess
   (test (op good-enough?) (reg guess))
   (branch (label sqrt-done))
   (assign guess (op improve) (reg guess))
   (goto (label test-guess))
 sqrt-done)
```

The `sqrt` machine looked simple because we abstract both `good-enough?` and `improve` as primitive operations. If we want to construct the `sqrt` machine without using these primitive operations, we must specify how to implement `good-enough?` and `improve` in terms of simpler operations, such as `square`, `abs` and `average`. Figure 2 shows the data paths for the elaborated `sqrt` machine. The controller for this elaborated `sqrt` machine is shown in figure 3. The controller sequence for this elaborated `sqrt` machine can be describe as follow:

```
(controller
   (assign guess (const 1))
 guess-loop
   (assign t (op square) (reg guess))
   (assign t (op -) (reg x) (reg t))
   (assign t (op abs) (reg t))
 test-guess
   (test (op <) (reg t) (const 0.001))
   (branch (label sqrt-done))
```


Email address: informlarry@gmail.com

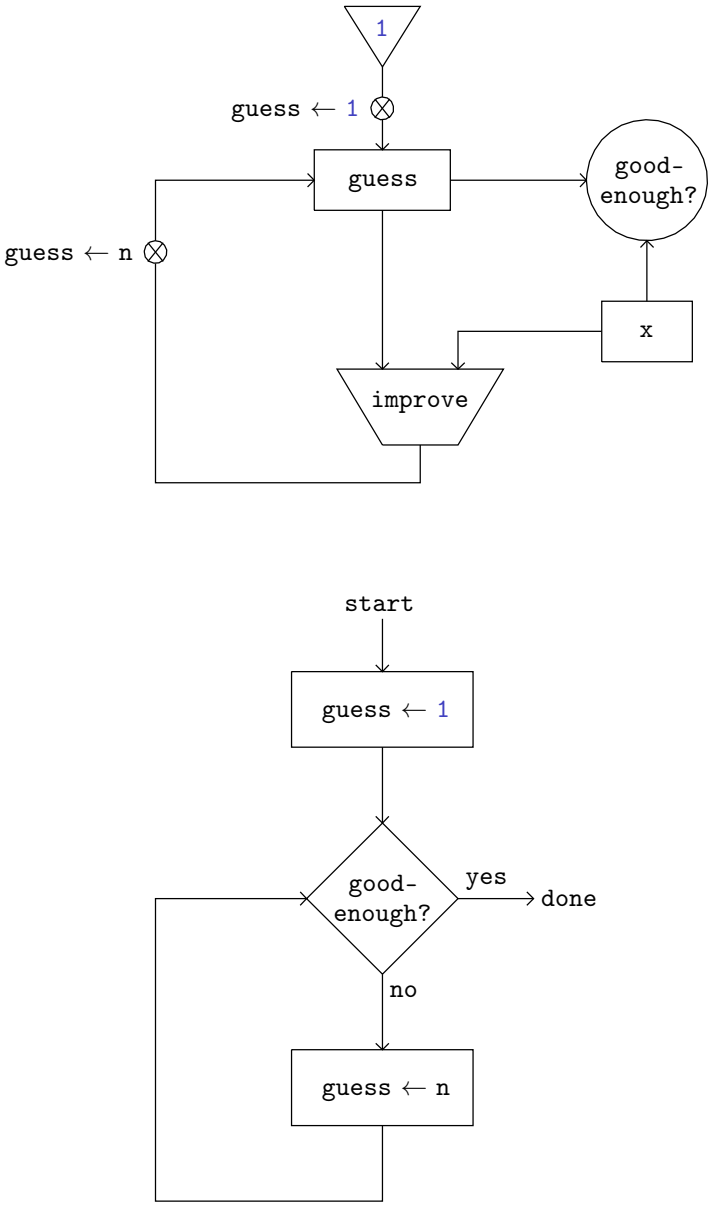

Figure 1. Data paths and controller for the simplified `sqrt` machine.

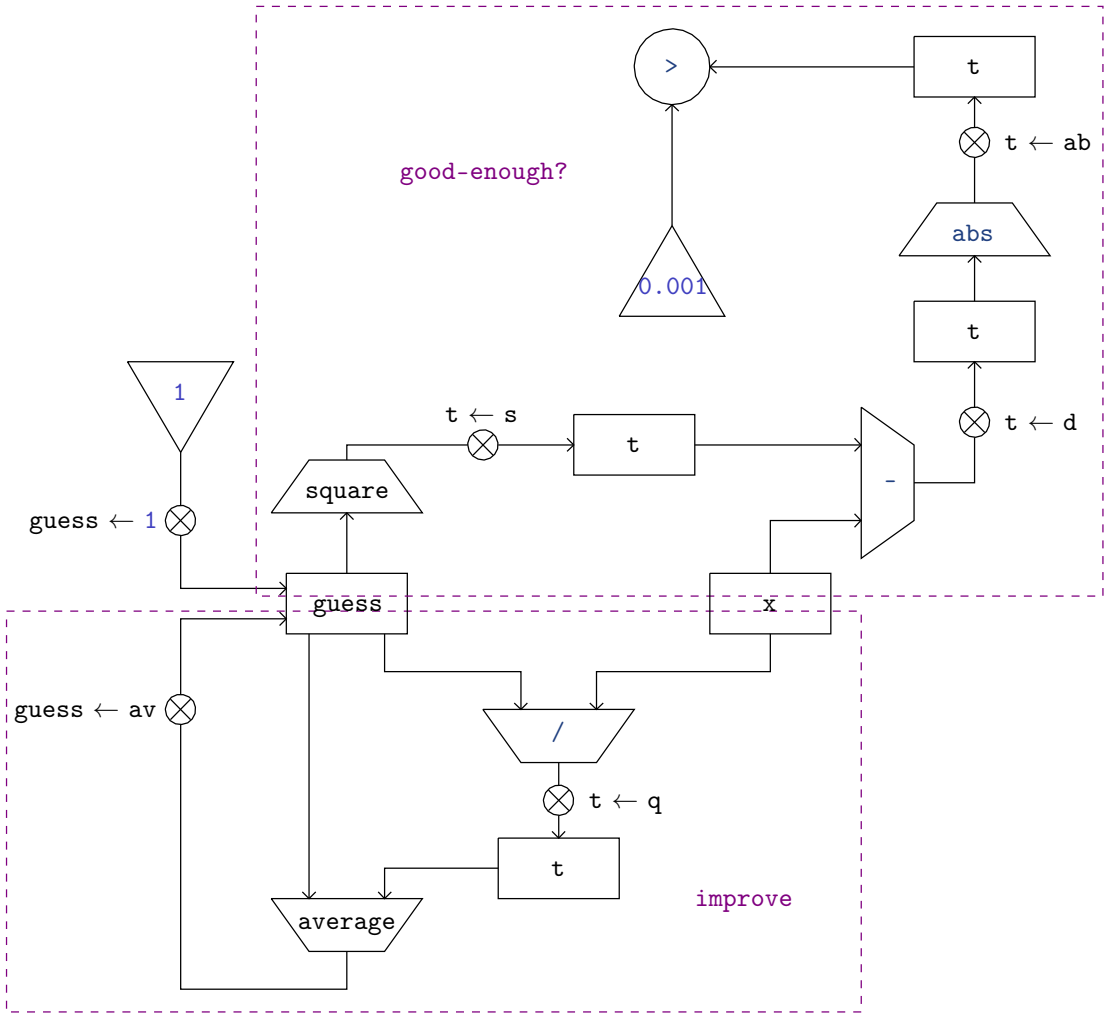

Figure 2. Data paths for the elaborated `sqrt` machine.

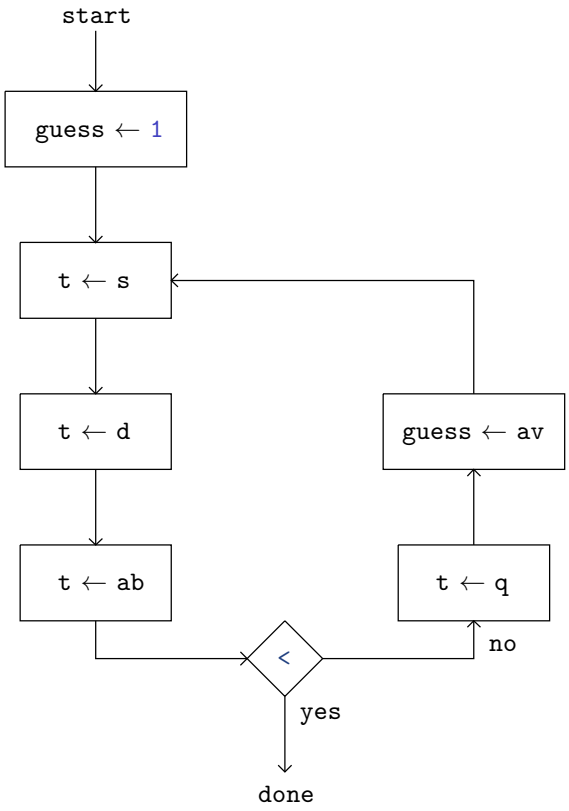

Figure 3. Controller for the elaborated `sqrt` machine.

```
   (assign t (op /) (reg x) (reg guess))
   (assign guess (op average) (reg t) (reg guess))
   (goto (label guess-loop))
 sqrt-done)
```

The `sqrt` machine can be constructed on a even elementary level if we further expand `square`, `abs` and `average` to arithmetic operations. Figure 4 shows the data path for the refined machine. The controller for this refined `sqrt` machine is shown in figure 5. The controller sequence for this refined `sqrt` machine can be described as follows:

```
(controller
   (assign guess (const 1))
 guess-loop
   (assign t (op *) (reg guess) (reg guess))
   (assign t (op -) (reg x) (reg t))
 test-t
   (test (op <) (reg t) (const 0))
   (branch (assign t (op *) (reg t) (const -1)))
 test-guess
   (test (op <) (reg t) (const 0.001))
   (branch (label sqrt-done))
   (assign t (op /) (reg x) (reg guess))
   (assign t (op +) (reg guess) (reg t))
   (assign guess (op /) (reg t) (const 2))
   (goto (label guess-loop))
 sqrt-done)
```

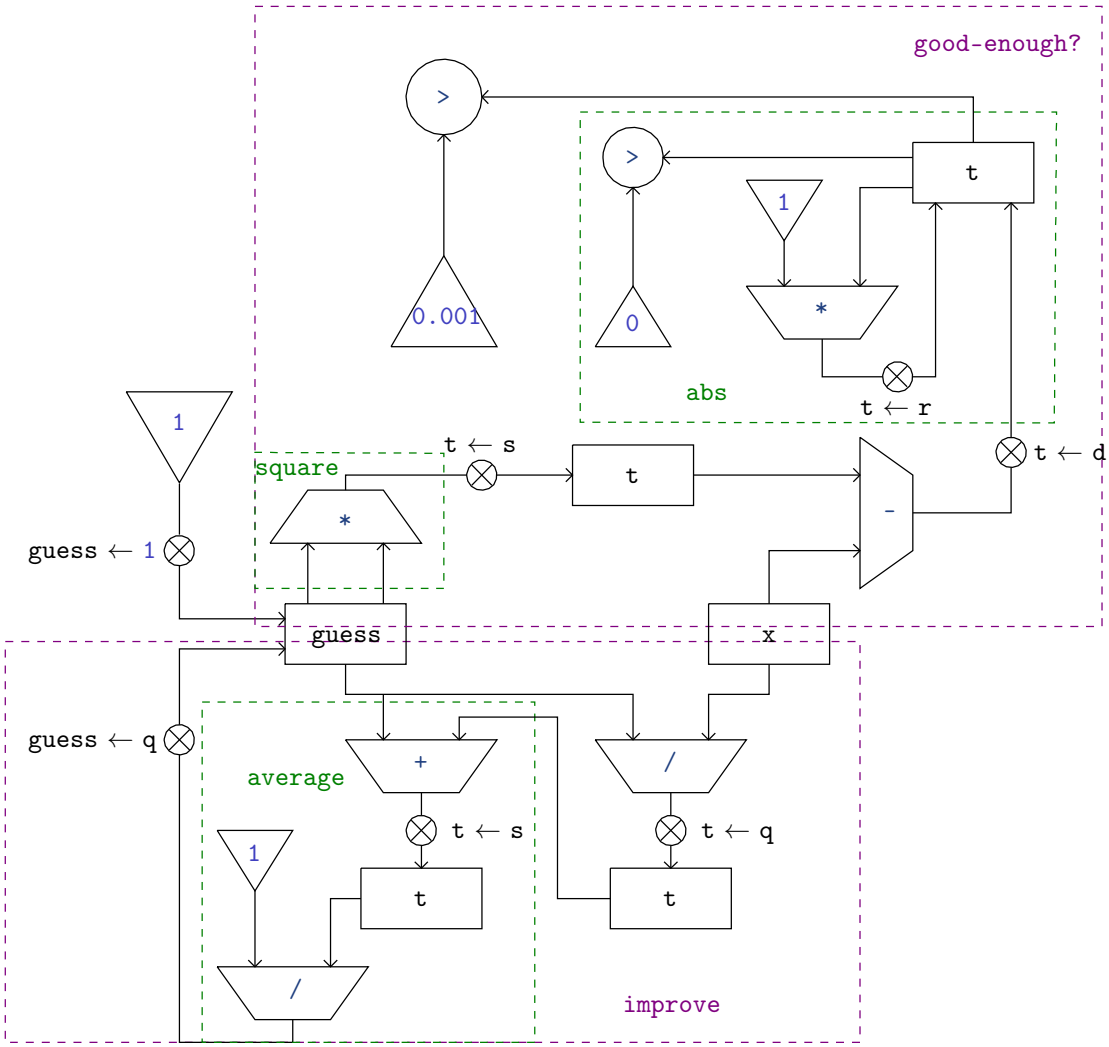

Figure 4. Data paths for the refined `sqrt` machine.

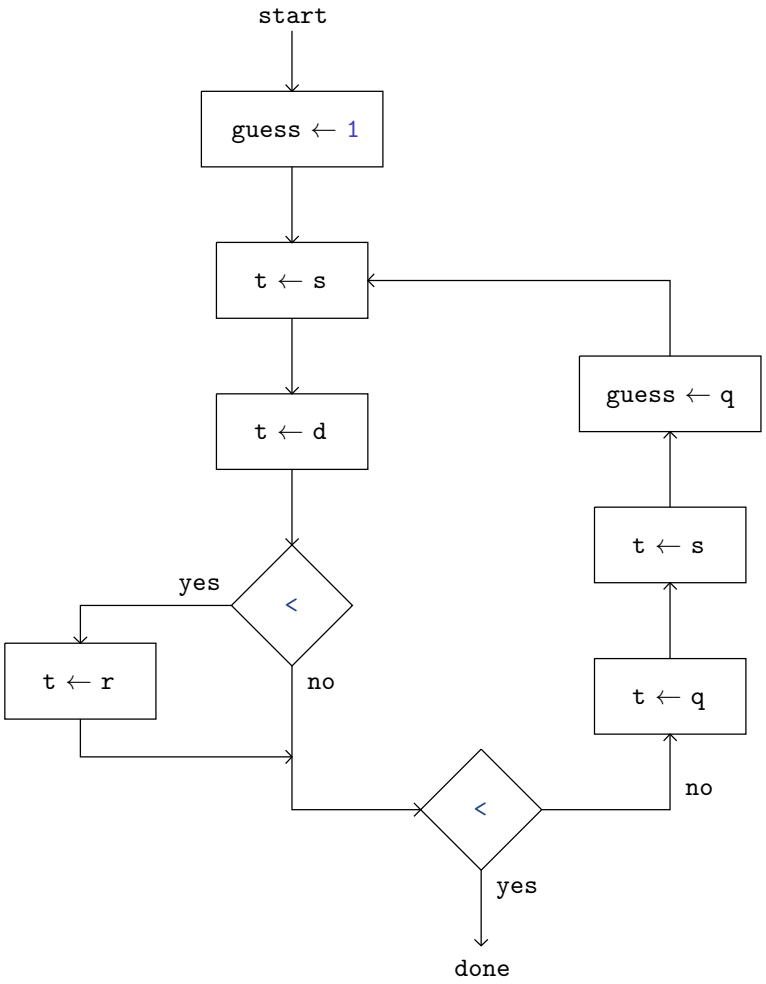

Figure 5. Controller for the refined `sqrt` machine.