

### Exercise 3.39.

Which of the five possibilities in the parallel execution shown above remain if we instead serialize execution as follows:

```
(define x 10)

(define s (make-serializer))

(parallel-execute (lambda () (set! x ((s (lambda () (* x x))))))
  (s (lambda () (set! x (+ x 1)))))
```

### Answer.


We see that one of the concurrent processes— $P_1$ , which sets  $x$  to the square of  $x$ , is not completely serialized. Whereas the other one— $P_2$ , which increments  $x$ , is serialized thoroughly.

By serializing the computation of the square of  $x$  in  $P_1$ , we ensure that the value of  $x$  will not be changed by  $P_2$  between the two times that  $P_1$  accesses it. This eliminates the value 110 from the five possibilities. However, this partial serialization on  $P_1$  cannot guarantee us that  $P_2$  won't interfere between the computation and assignment of  $P_1$ . Thus,  $x$  will still be possibly set to 100 after execution.

On the other hand, since the process  $P_2$  is fully serialized, no interference will occur between the computation and assignment of it. Therefore, the possibility that  $x$  will be set to 11 is eliminated. So, after parallel execution is complete,  $x$  will be left with one of three possible values:

- 101:  $P_1$  sets  $x$  to 100 and then  $P_2$  increments  $x$  to 101.
- 121:  $P_2$  increments  $x$  to 11 and then  $P_1$  sets  $x$  to the square of  $x$ .
- 100:  $P_1$  access  $x$  (twice), then  $P_2$  sets  $x$  to 11, then  $P_1$  sets  $x$ .

---

\*. Creative Commons  2013, Lawrence X. Amlord (颜世敏, aka 颜序).