

Exercise 4.19.

Ben Bitdiddle, Alyssa P. Hacker, and Eva Lu Ator are arguing about the desired result of evaluating the expression


```
(let ((a 1))
  (define (f x)
    (define b (+ a x))
    (define a 5)
    (+ a b))
(f 10))
```

Ben asserts that the result should be obtained using the sequential rule for `define`: `b` is defined to be `11`, then `a` is defined to be `5`, so the result is `16`. Alyssa objects that mutual recursion requires the simultaneous scope rule for internal procedure definitions, and that it is unreasonable to treat procedure names differently from other names. Thus, she argues for the mechanism implemented in exercise 4.16. This would lead to `a` being unassigned at the time that the value for `b` is to be computed. Hence, in Alyssa's view the procedure should produce an error. Eva has a third opinion. She says that if the definitions of `a` and `b` are truly meant to be simultaneous, then the value `5` for `a` should be used in evaluating `b`. Hence, in Eva's view `a` should be `5`, `b` should be `15`, and the result should be `20`. Which (if any) of these viewpoints do you support? Can you devise a way to implement internal definitions so that they behave as Eva prefers?¹

Answer.

Personally, I advocate the viewpoint of Alyssa.

To implement internal definitions as Eva prefers, we need a way to determine the dependency among defined variables. Theoretically, this can be done by exploiting the strategy of *Topological Sort*, but putting it down in Lisp would be a complicated work.

*. Creative Commons  2014, Lawrence X. Amlord (颜世敏, aka 颜序).
Email address: informlarry@gmail.com

1. The MIT implementors of Scheme support Alyssa on the following grounds: Eva is in principle correct—the definitions should be regarded as simultaneous. But it seems difficult to implement a general, efficient mechanism that does what Eva requires. In the absence of such a mechanism, it is better to generate an error in the difficult cases of simultaneous definitions (Alyssa's notion) than to produce an incorrect answer (as Ben would have it).