```
(define (new-if predicate then-clause else-clause)
  (cond (predicate then-clause)
        (else else-clause)))
(define (sqrt-iter guess x)
  (new-if (good-enough? guess x)
          guess
          (sqrt-iter (improve guess x) x)))
```

When Alyssa attempts to use the `new-if` procedure to compute square roots, the result will be an infinite loop due to `new-if` not being a special form. When `sqrt-iter` is initially applied to `guess` and `x`, the values of `guess` and `x` will be substituted into a `new-if` procedure. This `new-if` procedure is not a special form, and instead will be evaluated in applicative-order. This means the interpreter will evaluate the operands, which includes another `sqrt-iter` procedure. Before the interpreter can apply the first `new-if`, there is another `new-if` that must be evaluated after the initial guess is improved. This continues in an infinite loop, even if the initial guess is good enough, because `new-if` must evaluate the operands before applying the procedure which would yield the expected result of allowing an exit when the guess is good enough. The built-in `if` procedure is a special form and avoids this because it will simply evaluate the predicate and won't evaluate the recursive call until it knows the guess is not good enough. The `new-if` will always evaluate the recursive call before entering case analysis which means it can never reach the predicate step, and can never return a value. Alyssa's examples worked because the operands included only primitives, no recursive calls, so it didn't matter that `new-if` was evaluated in applicative-order.