CFA Limitation: Super-β inlining Inlining a function based on flow information is blocked by the lack of environmental precision in control-flow analysis. Shivers termed the inlining of a function based on flow information *super-β* inlining [27], because it is beyond the reach of ordinary β-reduction. Consider:

```
(let ((f (lambda (x h)
           (if x
               (h)
               (lambda () x)))))
  (f #t (f #f nil)))
```

Nearly any CFA will find that at the call site `(h)`, the only procedure ever invoked is a closure over the lambda term `(lambda () x)`. The lambda term's only free variable, `x`, is in scope at the invocation site. It *feels* safe to inline. Yet, if the compiler replaces the reference to `h` with the lambda term `(lambda () x)`, the meaning of the program will change from `#f` to `#t`. This happens because the closure that gets invoked was closed over an earlier binding of `x` (to `#f`), whereas the inlined lambda term closes over the binding of `x` currently in scope (which is to `#t`). Programs like this mean that functional compilers must be conservative when they inline based on information obtained from a CFA. If the inlined lambda term has a free variable, the inlining could be unsafe.

Specific problem To determine the safety of inlining the lambda term *lam* at the call site $[\![\texttt{(f ...)}]\!]$, we need to know that for every environment ρ in which this call is evaluated, that $\rho[\![\texttt{f}]\!] = (lam, \rho')$ and $\rho(v) = \rho'(v)$ for each free variable v in the term *lam*.[2]

CFA Limitation: Globalization Sestoft identified globalization as a second blindspot of control-flow analysis [25]. Globalization is an optimization that converts a procedure parameter into a global variable when it is safe to do so. Though not obvious, globalization can also be cast as a problem of reasoning about environments: if, for every state of execution, all *reachable* environments which contain a variable are equivalent for that variable, then it is safe to turn that variable into a global.

Specific problem To determine the safety of globalizing the variable v, we need to know that for each reachable state, for any two environments ρ and ρ' reachable inside that state, it must be that $\rho(v) = \rho'(v)$ if $v \in dom(\rho)$ and $v \in dom(\rho')$.

CFA Limitation: Rematerialization Compilers for imperative languages have found that it can be beneficial to rematerialize (to recompute) a value at its point of use if the values on which it depends are still available. On modern hardware, rematerialization can decrease register pressure and improve cache performance. Functional languages currently lack analyses to drive rematerialization. Consider a trivial example:

[2] The symbol ρ denotes a conventional variable-to-value environment map.