# Syntax:

Our syntax is basically the regular lambda calculus syntax with some standard additions (*if-then-else*, *let-in*, *fix*) plus native support for monads (*bind* and *return*).

We also natively support constructs for manipulating resources. We define a resource as anything data structure that controls a limited resource that needs to be freed as soon as possible. Resources range from memory to files to network connections to lots of other things that could be easily added by modifying the definition of *r*.

# Types:

Our language has a type system that complements the syntax of a standard lambda calculus with primitive types.

Since we natively support the state monad, we also add typing constructs for representing stateful computations and resources.

We carry as a parameter of the state monad the current capabilities (resources) that its statement supports.

# Typing Rules:

Our typing rules ensure the usual properties of the lambda calculus and its types. In addition we also support