Programación Declarativa 2020-2
Todavía mas mónadas (`Identity`, `Continuation`)

Javier Enríquez Mendoza Favio E. Miranda Perea

Facultad de Ciencias UNAM

4 de junio de 2020

Identity

La mónada identidad no incorpora ninguna estrategia computacional.

Simplemente aplica la función ligada a un valor sin ninguna modificación.

Desde el punto de vista computacional, no hay una razón para usar la mónada identidad.

En su lugar se podría simplemente ... aplicar la función.

Esta mónada sin embargo tiene un uso especial en las transformaciones monádicas que se discutirán mas adelante.

Definición

La mónada identidad se define como sigue:

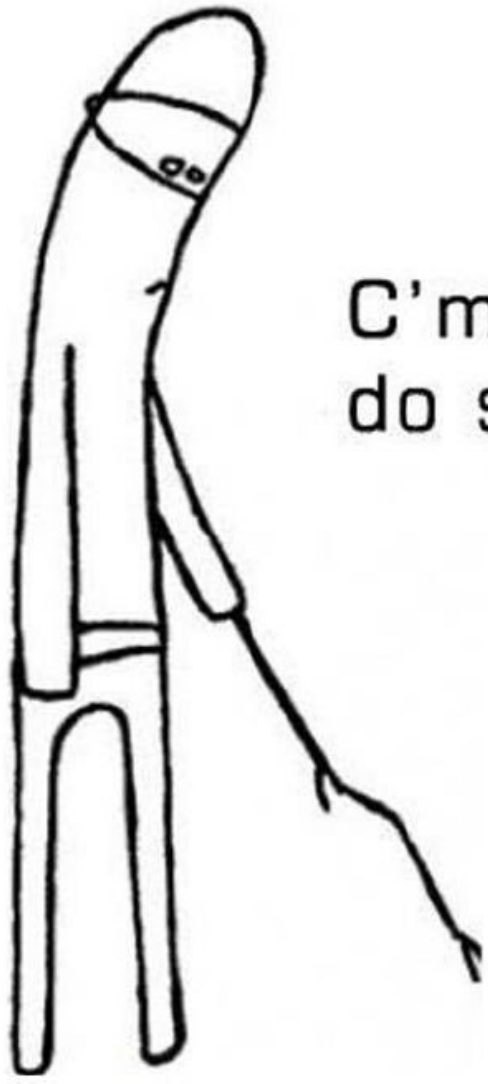
```
newtype Identity a = Identity {runIdentity :: a}

instance Monad Identity where
  return              = Identity
  (Identity x) >>= f = f x
```

Se utiliza la etiqueta `runIdentity` porque sigue un estilo que representa explícitamente los valores de mónadas como cálculos.

De esta forma los cómputos son construidos con los operadores monádicos y para extraer el valor se utiliza la función `runIdentity`.

Se puede ver en la definición de la instancia que la mónada `Identity` no hace nada.



C'mon,
do something...

Continuation

Las continuaciones representan el futuro de un cómputo, como una función de un resultado intermedio al resultado final.

En la estrategia CPS (Continuation Passing Style) los cómputos son construidos como una secuencia de continuaciones anidadas que termina en `id`.

De esta forma se tiene acceso a todo el proceso de evaluación y no solo al resultado de esta.

Manipular estas continuaciones podría representar la manipulación de los cómputos subsecuentes.

Tales como interrumpir un computo en medio de su ejecución, abortar una porción del cómputo o entrelazar ejecuciones de cómputos.

Las continuaciones son utilizadas como estructuras de control complejas, para manejar errores y para crear subrutinas.

Definición

La mónada `Continuation` adapta CPS a la estructura mónadica. Y está definida de la siguiente forma:

```
newtype Cont r a =
        Cont { runCont :: ((a -> r) -> r) }

instance Monad (Cont r) where
 return a = Cont $ \k -> k a
 (Cont c) >>= f =
          Cont $ \k -> c (\a -> runCont (f a) k)
```

Esta mónada representa cómputos en CPS que producen un resultado intermedio de tipo `a` y `r` es el tipo del resultado final del cómputo.

callCC

callCC es le mecanismo con el cual se escapa de una continuación y así se interrumpe la ejecución del cómputo.

Abortar la ejecución permite terminar el cómputo y regresar un valor inmediatamente.

Para esto Haskell utiliza la clase MonadCont definida como sigue:

```
class (Monad m) ==> MonadCont m where
  callCC :: ((a -> m b) -> m a) -> m a
```

La instancia para la mónada Continuation es la siguiente:

```
instance MonadCont (Cont r) where
  callCC f = Cont $
    \k -> runCont (f (\a -> Cont $ \_ -> k a)) k
```

Ejemplo

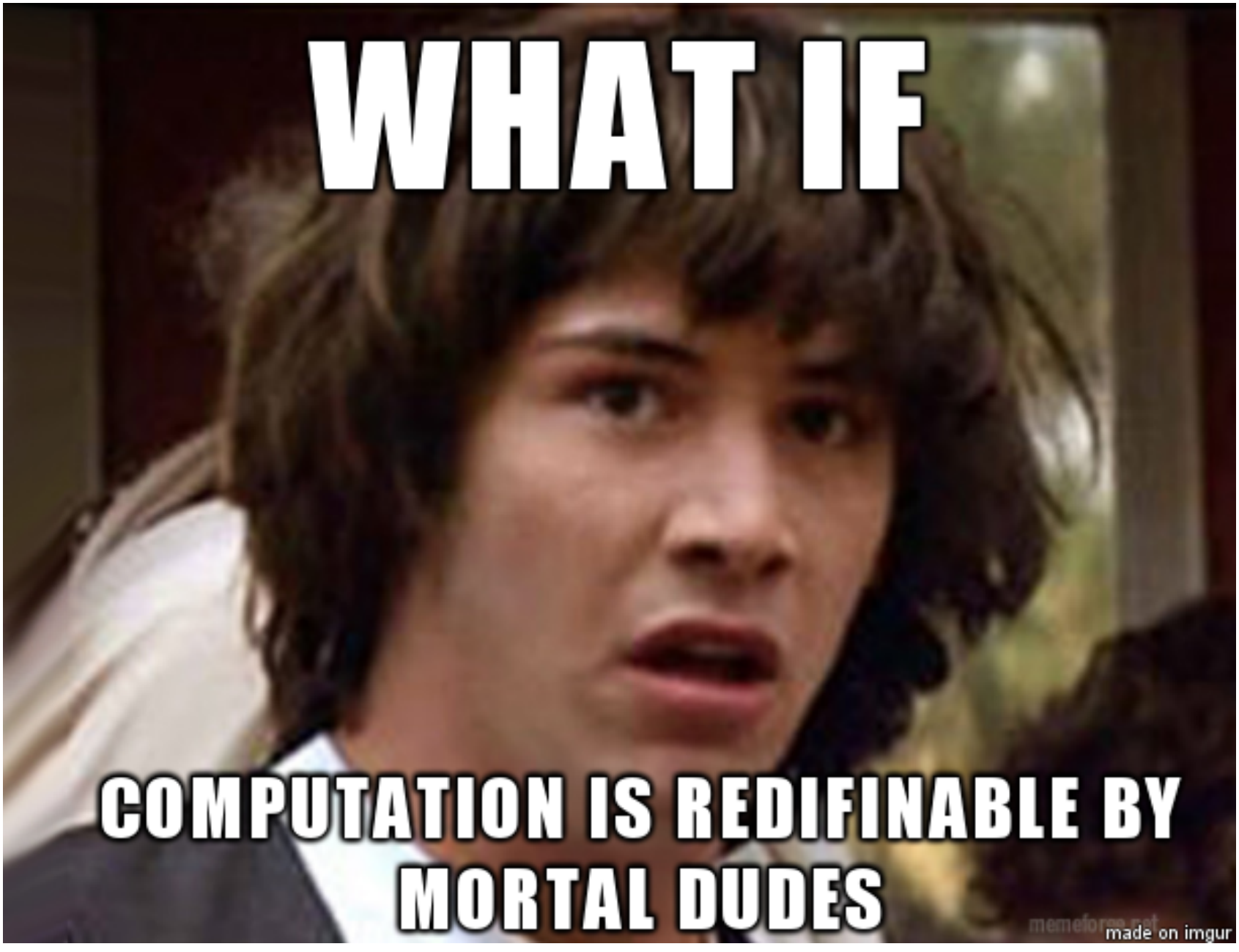
Vamos a usar la mónada `Continuation` para escribir una función que calcula la longitud de una lista.

```
longitud :: [a] -> Cont r Int
longitud l = return (length l)
```

La función longitud captura la continuación actual, en este caso el único cálculo pendiente es calcular la longitud de la lista.

Este cálculo no se realiza hasta que la continuación es aplicada usando `runCont`.

La ejecución del código muestra el resultado después de aplicar la continuación.


WHAT IF
COMPUTATION IS REDIFINABLE BY MORTAL DUDES
made on imgur