Programación Declarativa 2020-2
Transformadores Monádicos

Javier Enríquez Mendoza Favio E. Miranda Perea

Facultad de Ciencias UNAM

4 de junio de 2020

Combinaciones

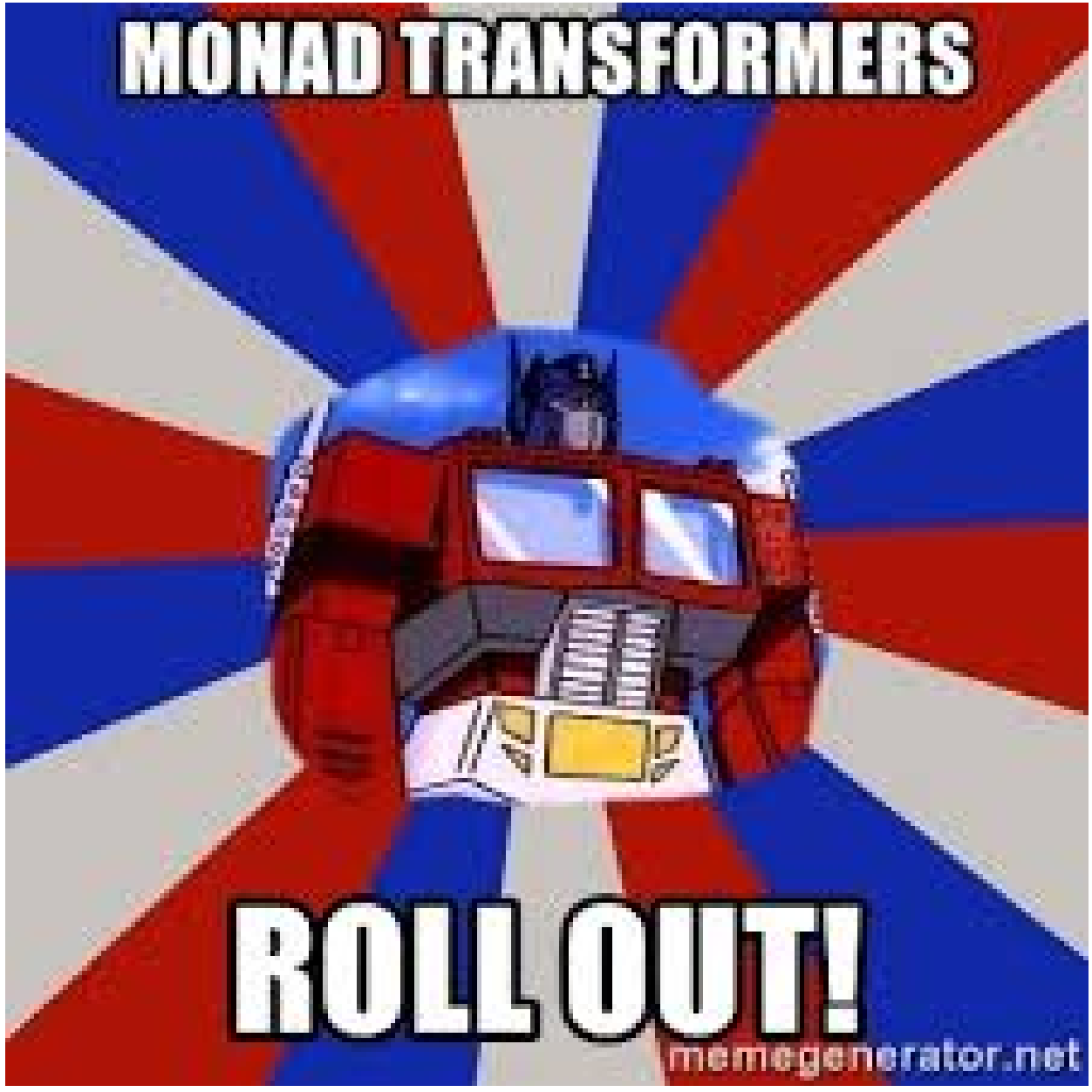
Hemos jugado ya con algunas mónadas y vimos de lo que son capaces.

Estudiamos el comportamiento de mónadas como : `IO`, `Maybe`, `List`, `State`, `Identity` y `Continuation`.

Con todas estas mónadas la pregunta que seguramente viene a la mente es:

¿Como se pueden combinar?

Y la respuesta es usando transformadores monádicos.


MONAD TRANSFORMERS
ROLL OUT!
memegenerator.net

Transformadores Monádicos

Los transformadores mónadicos pueden verse como anidar mónadas una dentro de otra.

De tal forma que tengamos disponibles los cómputos de ambas.

Y de esa forma se puedan realizar programas mas complejos.

Entonces la pregunta es:

¿Como voy a usar una mónada cuando ya estoy usando otra?

Envolver

La idea es justo envolver una mónada en un contenedor monádico que nos permita simular el comportamiento de otra mónada.

Un transformador monádico es un tipo wrapper.

Está parametrizado por otro tipo monádico

De esta forma se pueden ejecutar cómputos de la mónada interna al mismo tiempo que se agrega el comportamiento del nuevo tipo monádico.

Por convención el nombre del transformador es el mismo que el de la mónada pero con una `T` al final, por ejemplo `MaybeT`.

MonadTans

Para definir transformadores monádicos Haskell tiene la clase `MonadTrans`.

Esta clase está definida de esta forma:

```
class MonadTrans t where
    lift :: (Monad m) ==> m a -> t m a
```

Hay que notar que el transformador está parametrizado por un tipo monádico, esto se puede ver en el tipo de su único método, es decir, espera una mónada para funcionar.

El método lift envuelve este tipo en la nueva mónada, no confundir con el combinador `liftM`.

Todas las mónadas predefinidas en Haskell tienen también su instancia ya definida de la clase `MonadTrans`

Identity

De esta forma el transformador `Maybe r IO` es una combinación de las mónadas `Maybe` y `IO` sobre el tipo `r`.

Se puede generar una versión no transformada de la mónada a partir del transformador.

Esto se logra aplicando el transformador a la mónada `Identity`

Por ejemplo `StateT r Identity` es lo mismo que `State r`

Y creían que `Identity` no servia para nada.

MonadIO

La combinación de mónadas con `IO` es algo muy común cuando se esta programando en Haskell.

Es por eso que `IO` tiene su propia versión de `MonadTrans` que está optimizada para ser mas eficiente.

Ésta se llama `MonadIO` y está definida como sigue:

```
class (Monad m) ==> MonadIO m where
    liftIO :: IO a -> m a
```

Es subclase de `Monad` y lo que hace es envolver la mónada `m` en `IO`

Es equivalente a `IOT a m`

Capas

La parte mas bonita de los transformadores es que son al mismo tiempo mónadas.

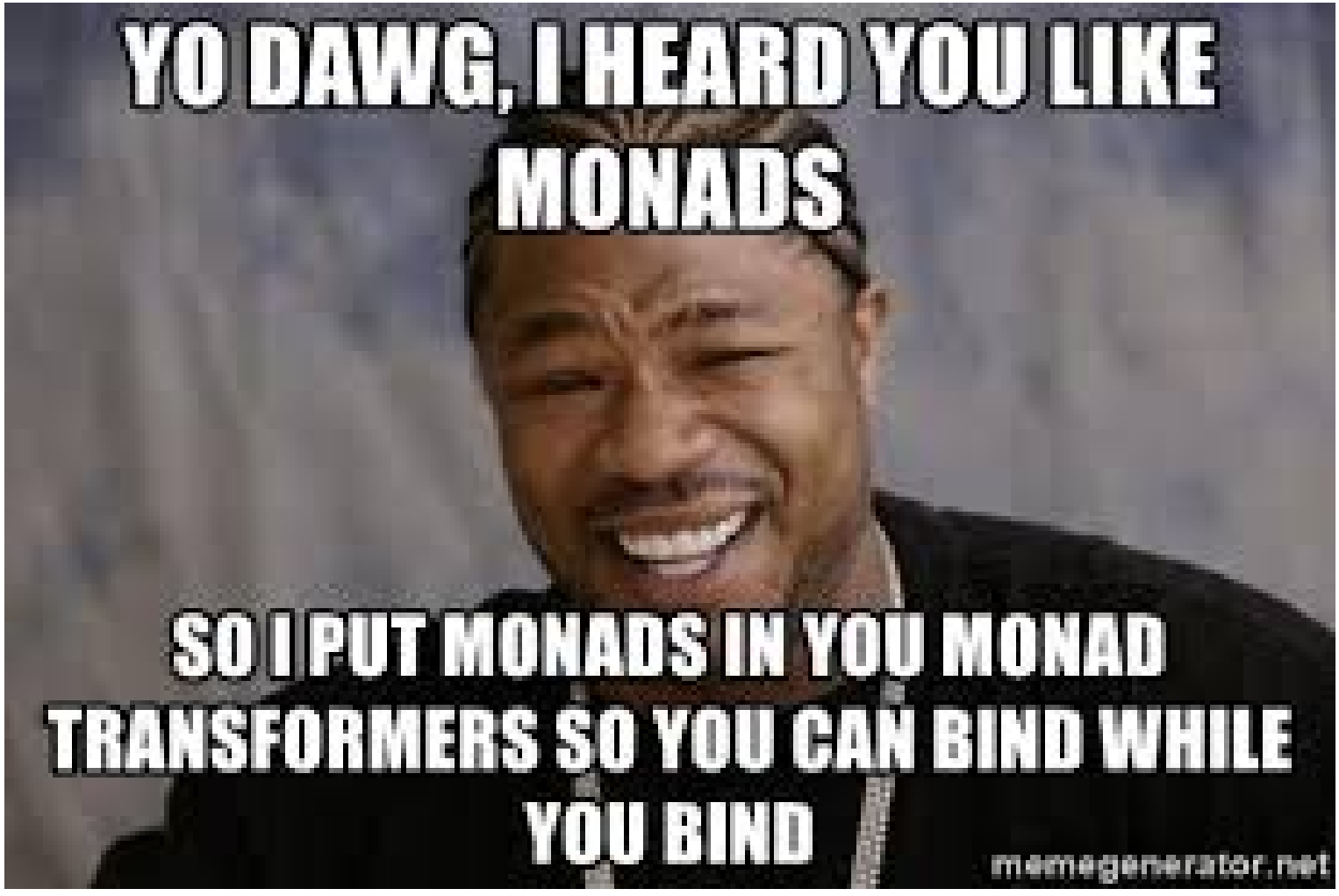
Esto significa que podemos envolver un transformador en otro, hasta el nivel que queramos.

Esto nos permite anidar todas las mónadas necesarias.

Por ejemplo el transformador `MaybeT r (ReaderT s Continuation)`

Esto se puede ver como si se estuvieran apilando mónadas.

Pero ... Un gran poder conlleva una gran responsabilidad.


YO DAWG, I HEARD YOU LIKE MONADS
SO I PUT MONADS IN YOU MONAD TRANSFORMERS SO YOU CAN BIND WHILE YOU BIND
memegenerator.net