

LPP - 3CFU : COSA

Sollecitate l'uso delle Type Class (Typeclassopedia)

- * FUNCTOR
- * APPLICATIVE
- * MONAD ←

Perché

Computational effects in
pure functional style

PS PSE

LPP - BCFU : COME

- * Definiamo un linguaggio
 { costanti numeriche , una operazione }
- * [costante numerica] = numero binario

[operazione] =

div :: BN → BN → BN

div N \emptyset _ = N \emptyset

div n d | n (<) d = N \emptyset

| otherwise = succ(div (sub n d) d)

div?

$\text{div} :: BN \rightarrow BN \rightarrow BN$ Tipo Binary Numbers

$\text{div } N\phi - = N\phi$ BN dovrebbe essere istanza di Ord

$\text{div } n \ d \ | \ n (<) d = N\phi$

| otherwise = succ (div (sub n d) d)

necessarie per rendere
BN istanza di Num



\exists una "Migliore" per i nostri obiettivi ?

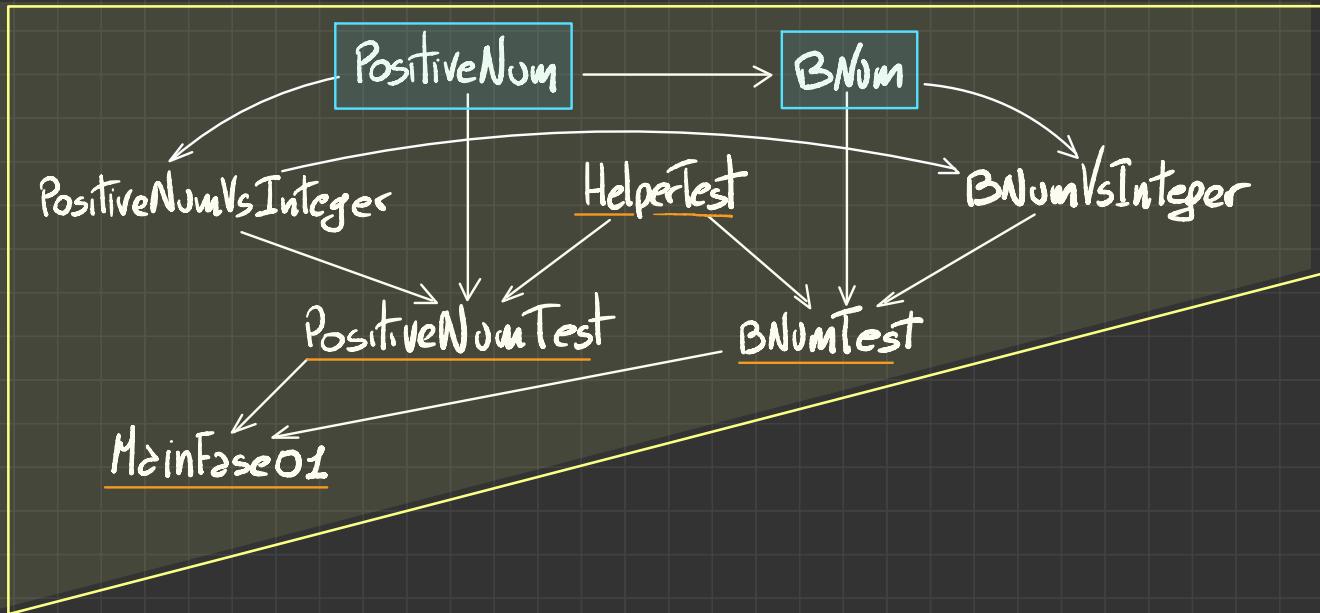
FASI

1m2

Numeri binari

Operazioni

Istanze di Num

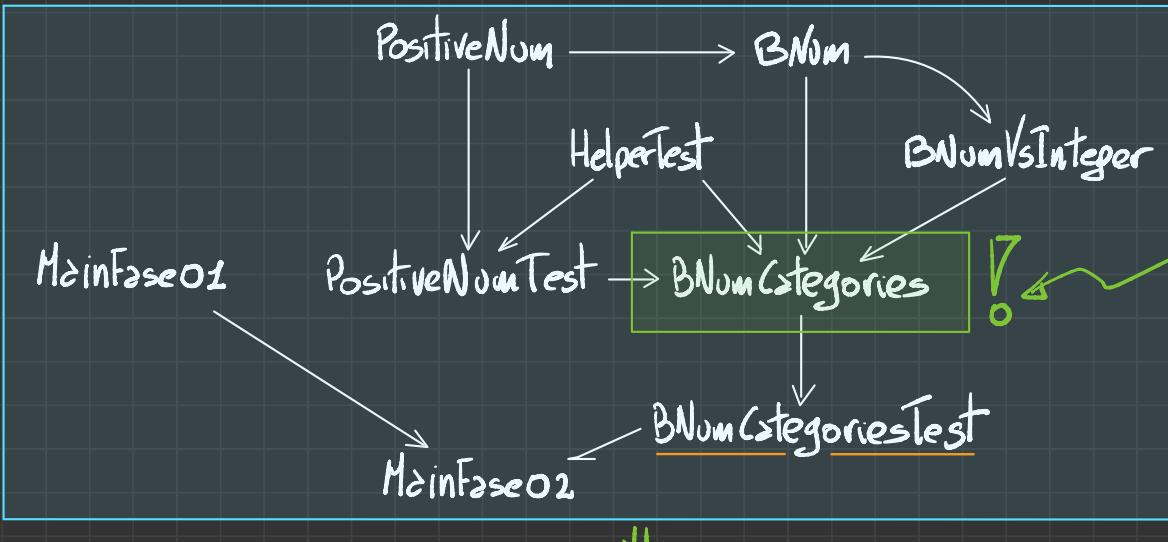


2da

BNum ISTANZA DI

Funzione ?

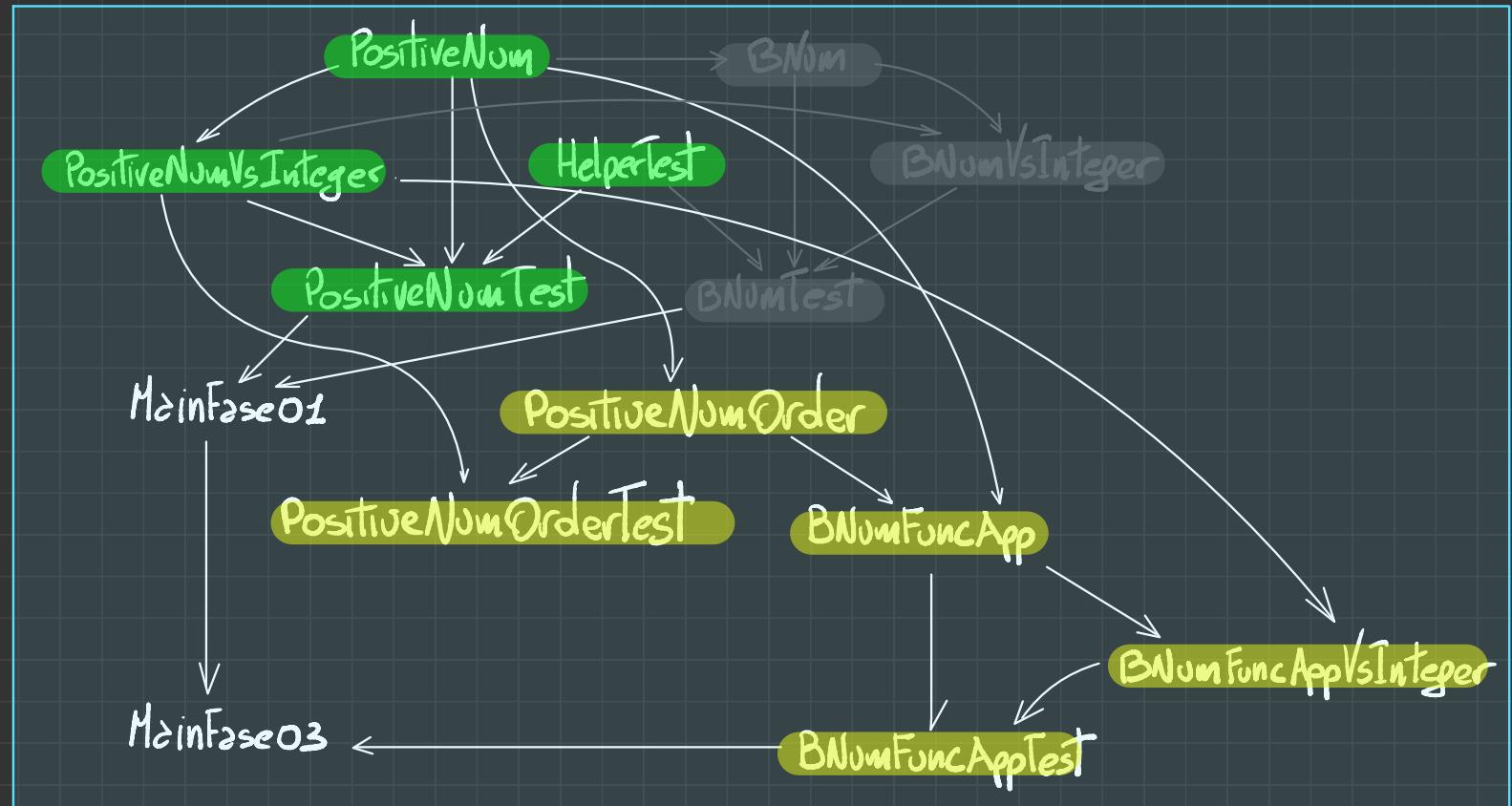
Applicative ?



Categorie ≈ Haskell
Funzione TypeClass

322

Numeri binari istanze di Num e Ord



Lte

INTERPRETI "ovvi"

BNonFuncApp

BNonFuncAppVsInteger

Language

HelperTest



EvalBasic

EvalException

EvalState



ad hoc \Leftrightarrow riprogettati "da zero"

Definizioni inedette \Rightarrow Modularizzazione

5 ↗

INTERPRETI "FATTORIZZATI" E MONDOLARI

Monad ID

Eval Basic ID Monadic

Monad Exception

BName FuncApp
BName FuncApp Vs Integer
Language HelperTest

Eval Exception Monadic

Monad State ID

Eval State ID Monadic

⚠ Gestione Errore
○ Conteggio passi

Eval State ID EXCEPTION Monadic

6TA

DA HASKELL A PROOF-ASSISTANT PER DIMOSTRARE PROPRIETÀ

Tipo & Istanza di:

Num $\Leftrightarrow \left\{ \begin{array}{l} + : \alpha \rightarrow \alpha \rightarrow \alpha \\ - : \alpha \rightarrow \alpha \rightarrow \alpha \\ * : \alpha \rightarrow \alpha \rightarrow \alpha \\ \text{negate} : \alpha \rightarrow \alpha \end{array} \right\} \cup \text{div}$

Ord \Leftrightarrow è istanza di Eq

Ord $\Leftrightarrow \leq : \alpha \rightarrow \alpha \rightarrow \text{Bool}$

! Compare $: \alpha \rightarrow \alpha \rightarrow \text{Ordering}$

Eq $\Leftrightarrow = : \alpha \rightarrow \alpha \rightarrow \text{Bool}$
 $\neq : \alpha \rightarrow \alpha \rightarrow \text{Bool}$

Interpreti modulari conseguono di:

class Functor f

fmap $: (\alpha \rightarrow \beta) \rightarrow f\alpha \rightarrow f\beta$
+ functor laws

class Applicative f

pure $: \alpha \rightarrow f\alpha$

$\langle *\rangle : f(\alpha \rightarrow \beta) \rightarrow f\alpha \rightarrow f\beta$

t.c. $fmap g x = pure g \langle *\rangle x$

class Monad f

$\gg = : f\alpha \rightarrow (\alpha \rightarrow f\beta) \rightarrow f\beta$

return = pure

t.c. return identità su, da etc.

hs-to-coq ?

(Tutorial: Verify Haskell Programs with hs-to-coq)