MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

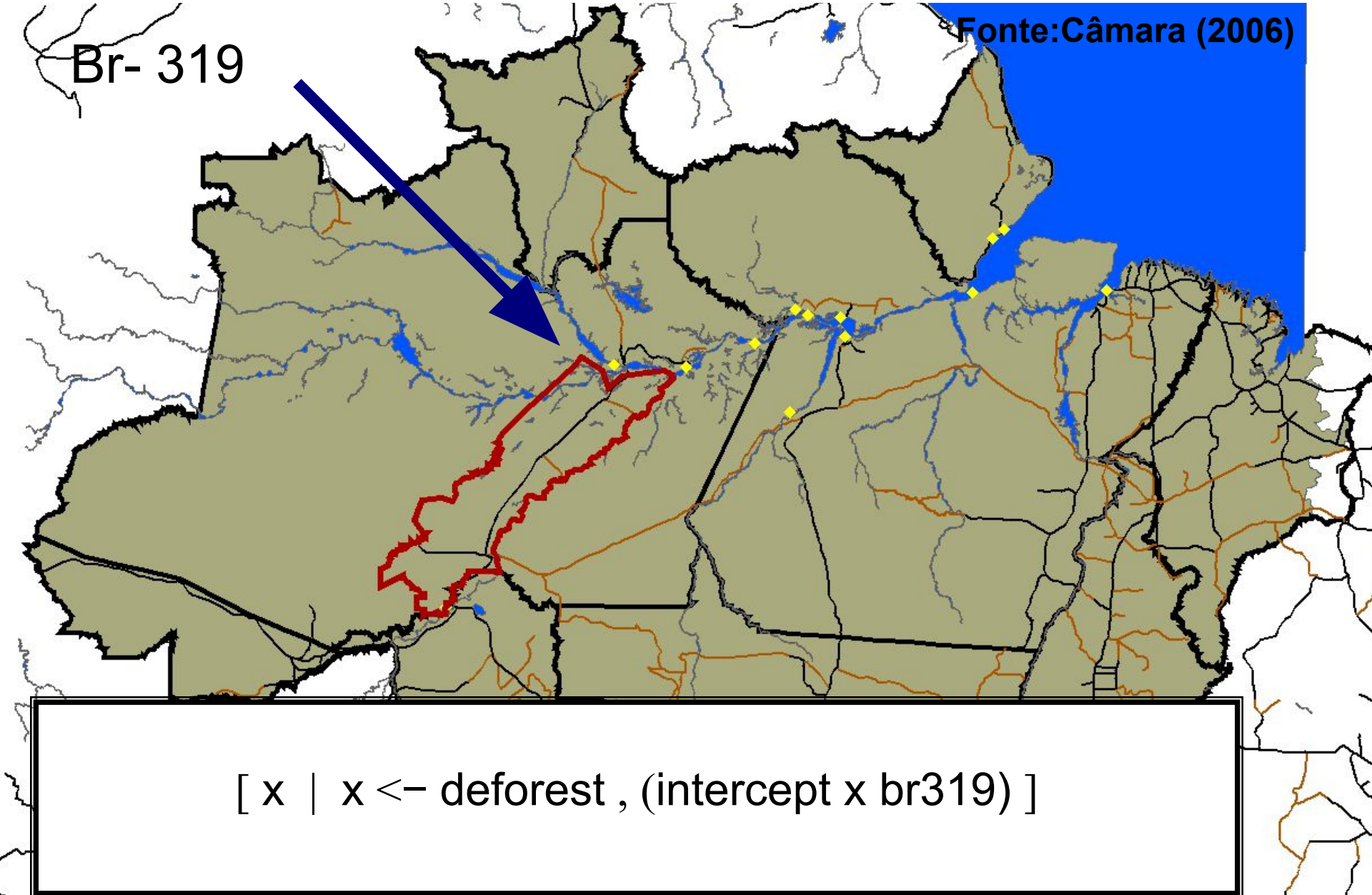
Integration of Functional Programming and Spatial Databases for Gis Application Development

Dissertação de Mestrado em CAP

Sergio Souza Costa

Orientador: Dr. Gilberto Câmara

Retorne o desmatamento ao longo da rodovia BR-319



[x | x <− deforest , (intercept x br319)]

Introdução: Programação Funcional

- **Programação funcional** é um paradigma de programação que trata a computação como uma avaliação de funções matemáticas
  - Principais características : (a) **tudo** é uma função, (b) **não existe** o conceito de variáveis e (c) **não possui** efeitos colaterais
  - Lisp, ML, Scheme, Miranda e Haskell
- **Haskell**
  - Desenvolvido no final da década de 80.
  - Principais características, (Peyton Jones, 2002): (a) Inferência de tipos, (b) avaliação preguiçosa, (c) funções de segunda ordem, (d) casamento de padrões e (e) tipos genéricos.

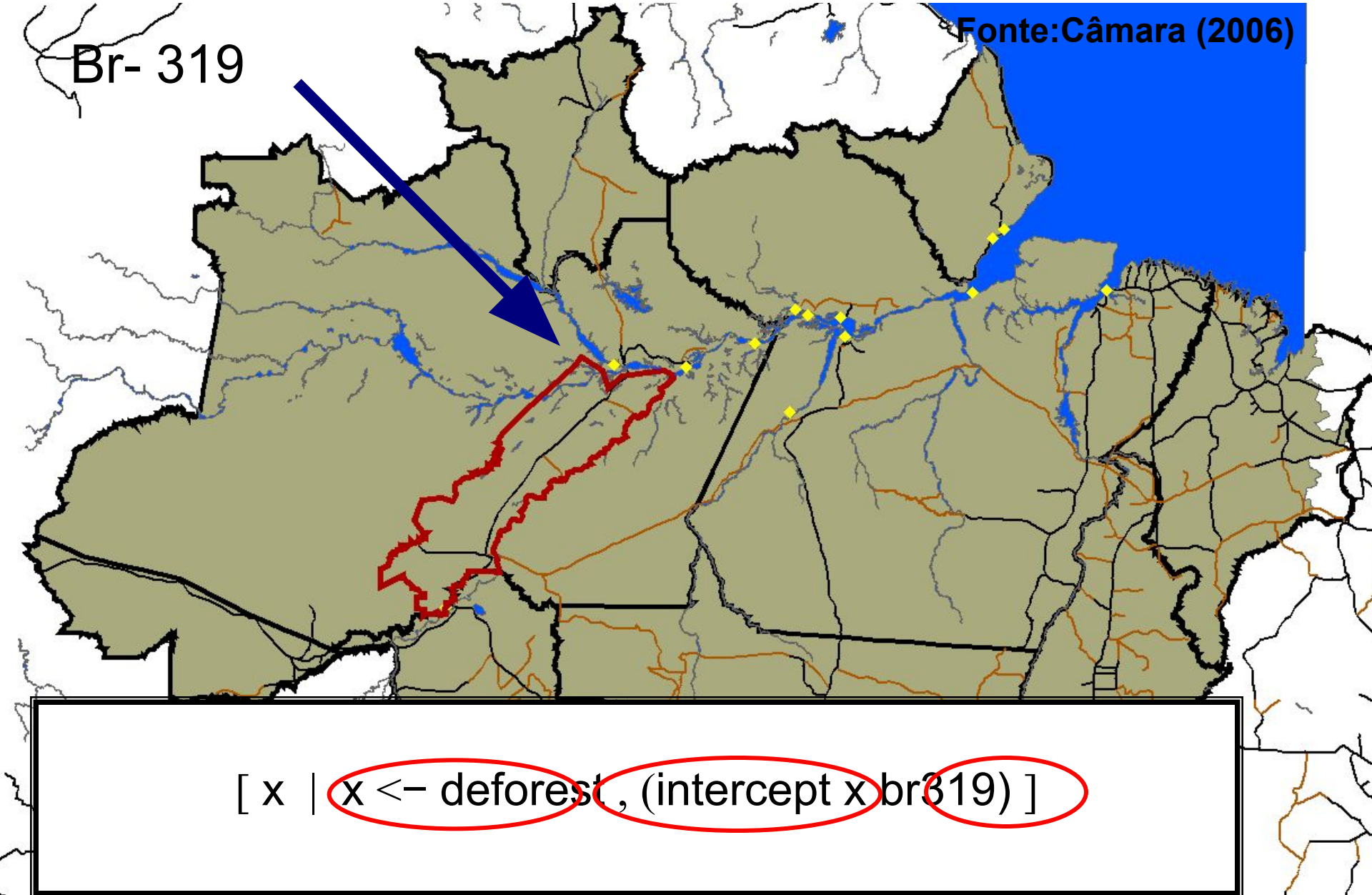
Introdução: Programação Funcional e Geoinformação

- 1995 - Frank e Kuhn- Specifying Open GIS with Functional Languages
- 1999 – Medak - Lifestyles - a new Paradigm in Spatio-Temporal Databases
- 2003 – Winter - Formal information modelling for standardisation in the spatial domain
- 2005 – Frank - Map Algebra Extended with Functors for Temporal Data

Introdução: Programação Funcional e Geoinformação

- Andrew Frank, Winter, Medak e Werner Kuhn apresentou algumas vantagens, como:
  - Semântica definida formalmente
  - Especificação executável
  - Garantia de consistência,
  - Estrutura algébrica extensível
  - Abstração de implementação.

Retorne o desmatamento ao longo da rodovia BR-319




```
[ x | x <- deforest , (intercept x br319) ]
```

Introdução: Motivação

- Aplicativos geográficos (e ou protótipos) compartilham algumas necessidades comuns.
  - Grande parte utilizam funções espaciais básicas, como operações topológicas (*intersects*, *crosses* ...) e operações métricas (*distance*, *area* ...).
  - Utilizam SGBDs (Oracle, MySQL, PostgreSQL ...) para o armazenamento e recuperação de informações espaciais.
- Esses recursos estão disponíveis em bibliotecas imperativas, como C++, C e Java.
  - **Questão:** Implementar todos esses recursos em uma linguagem funcional ?

Introdução: Hipótese

O melhor modo de se beneficiar da programação funcional em aplicativos geográficos, é termos um ambiente de desenvolvimento sobre um outro ambiente de suporte a banco de dados já existente.

- **Validação da hipótese**
  - Desenvolvemos um ambiente de interface com banco de dados espaciais, TerraHS.
  - Desenvolvemos uma aplicação real que foi validado com o uso de banco de dados espacial.

Roteiro

- Introdução
- Revisão (Mônadas e FFI)
- TerraHS
- Algebra de Mapas
- Conclusão

Revisão: Mônadas

- **Questão**: *Como tratar aspectos dependentes de I/O como acesso a bancos de dados em uma linguagem funcional ?*
  - Interação com o usuário
  - Efeitos colaterais
  - Seqüência

- **Solução**: Mônadas:
  - Moggi (1989) apresenta o conceito
  - Wader (1990) aplica em uma linguagem funcional

Revisão: Mônadas

- Definição de Mônadas em Haskell

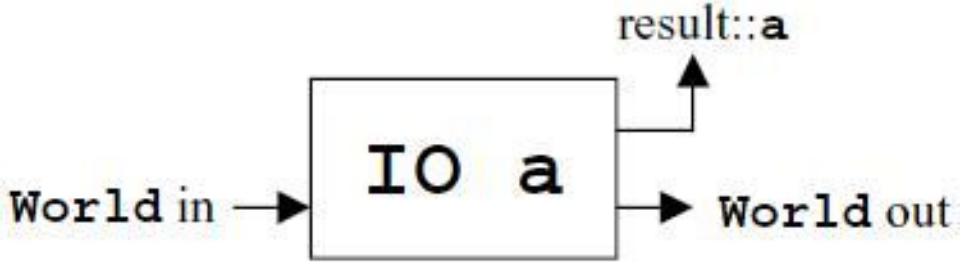
```
class Monad m where

        (>>=)  :: m a -> (a -> m b) -> m b

        (>>)   :: m a -> m b -> m b

        return :: a -> m a

        fail   :: String -> m a
```

- Alguns exemplos de mônadas:
  - Operações de entrada e saída (IO)
  - Exceção (Maybe)

Revisão: Mônadas

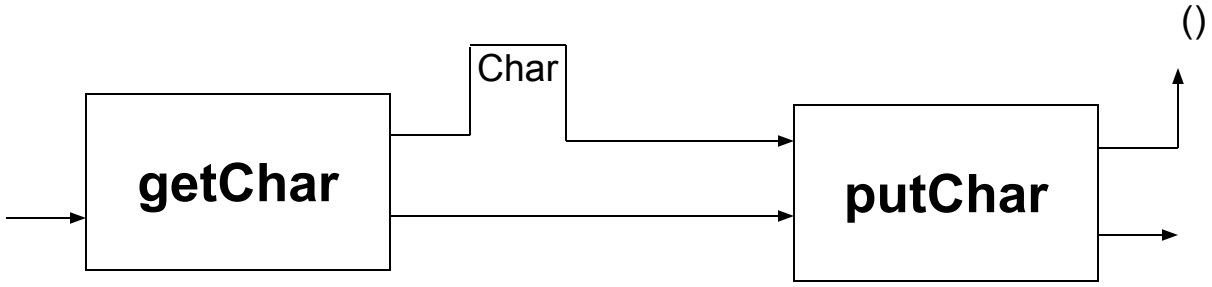
- Mônada IO

type IO a = World → (a, World)



Fonte: Peyton Jones (2002)

getChar >>= putChar



Revisão: Foreign Function Interface

- Foreign Function Interface (FFI)
  - Chamadas a códigos externos, no caso apenas de uma biblioteca C.
  - Exportar códigos Haskell para uma biblioteca externa.
  - Tratar *ponteiros de* tipos de dados externos.
  - Exportar ponteiros de tipos de dados Haskell

Revisão: Foreign Function Interface

- Chamadas a códigos externos.
  - Com o mesmo nome

```
foreign import ccall putChar::Char -> IO()
```

  - Especificando um novo nome

```
foreign import ccall “printChar” putChar::Char -> IO()
```

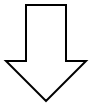
  - Funções puras

```
foreign import ccall unsafe sin::Float -> Float
```

Revisão: Foreign Function Interface

- Tratar *ponteiros de* tipos de dados externos
  - Haskell possui um tipo *Ptr*, para representar referencia a objetos
- Considere a seguinte função:

```
double distance (Point* pt1, Point* pt1)
```



```
data Point = Point

foreign import ccall unsafe

    distance :: Ptr Point -> Ptr Point -> Double
```

Roteiro

- Introdução
- Revisão (Mônadas e FFI)
- TerraHS
- Algebra de Mapas
- Conclusão

TerraHS

- Objetivo principal:
  - Desenvolvimento rápido e concisos de protótipos de aplicativos geográficos.
- Objetivos específicos:
  - Acesso a diferentes bancos de dados espaciais.
  - Acesso a um conjunto de operações espaciais básicas.

TerraHS: Arquitetura do TerraHS

- Arquitetura

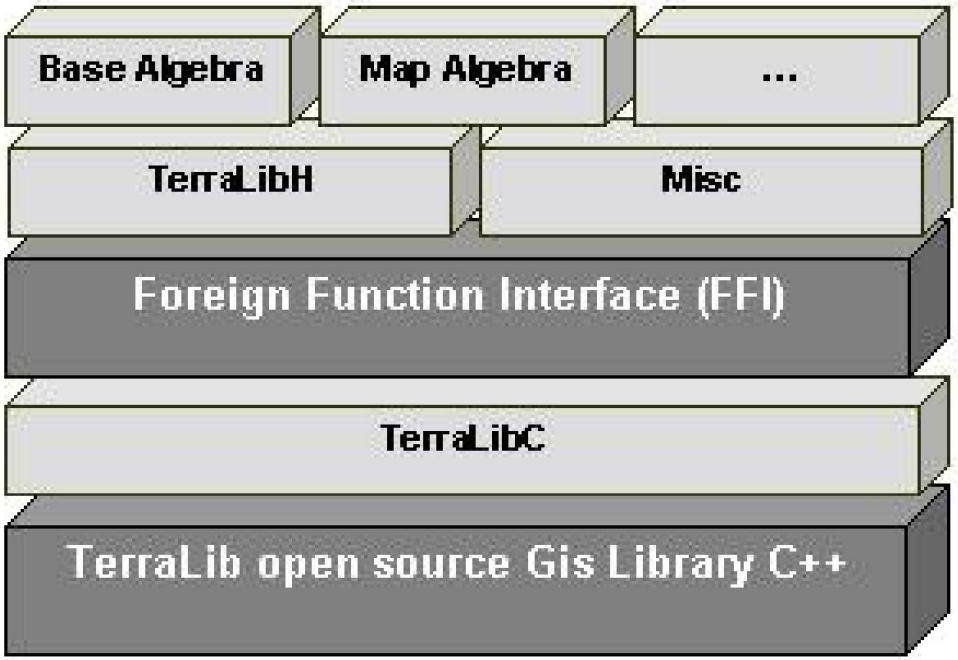


Figura 1: Arquitetura do TerraHS

TerraHS

- Nos provemos os seguintes recursos:
  - Operações Espaciais
  - Tipos de dados Espaciais
  - Acesso a banco de dados

TerraHS: Operações Espaciais

- Operações topologicas
  - Eg. *disjoint, intersects, crosses ...*
- Operações de conjunto
  - Eg. *union, intersection e difference*
- Operações métricas
  - Eg.*distance, length* e *area*

TerraHS: Tipos de Dados Espaciais

- Damos suporte apenas a definição de geo-objetos, onde um geo-objeto é definido por uma tripla:

```
data GeoObject = GeoObject
        (ObjectId,[Attribute], [Geometry])
```

Onde:

- ObjectId: Identifica um geo-objeto e representado por uma cadeia de caracteres.
- Attribute: Descreve os atributos que pode ser de diferentes valores: *inteiro, real* ou *cadeia de caracteres.*
- Geometry: Descreve a parte espacial e é representado por tipos de dados vetoriais ou células.

TerraHS: Tipos de Dados Espaciais

- Exemplo:

Minas Gerais - MG

População	20.595.499
Área	586.528,293
IDH	0,773
Clima	Tropical Aw

TerraHS: Acesso a Banco de Dados

- Banco de dados TerraLib

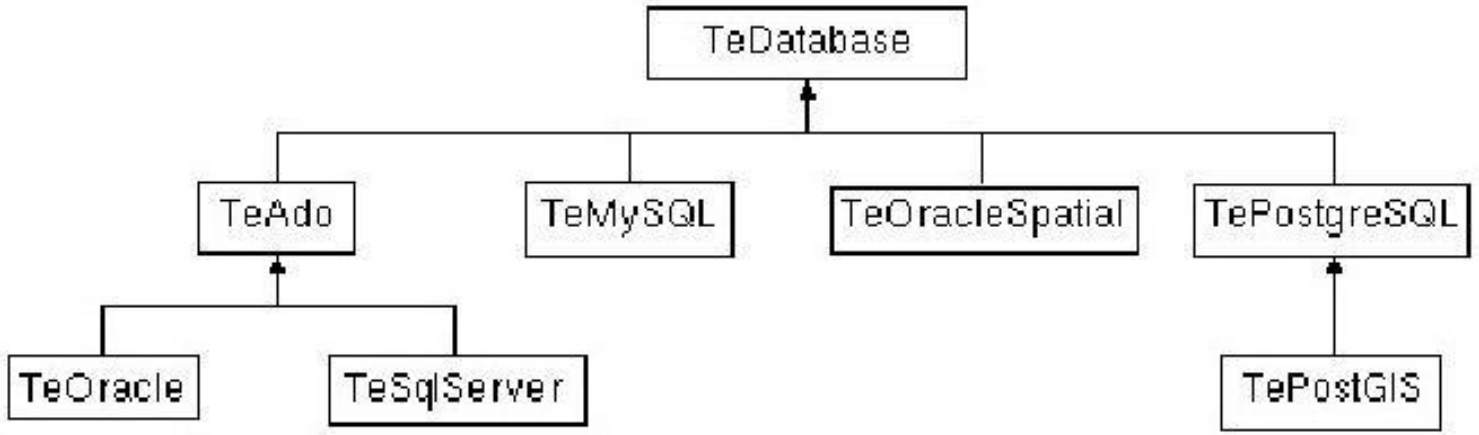


Figura 2: Drivers TerraLib - Fonte: Vinhas (2005)

```
data TeDatabase = TeMySQL String String String String
                | TePostgreSQL String String String String
                | TeAdo String
```

TerraHS: Acesso a Banco de Dados

- Operações básicas de acesso a banco de dados
  - open : Abre uma conexão com um banco de dados
  - Close: Encerra uma conexão com um banco de dados
  - retrieve: Retorna um conjunto de geo-objetos de um layer TerraLib.
  - store: Armazena um conjunto de geo-objetos em um layer TerraLib.
  - errorMessage: Retorna uma mensagem de erro

TerraHS: Exemplo de um programa em TerraHS

- Exemplo:

```
import Algebras.Base
import TerraHS.TerraLib

host = “sputnik”
user = “Sergio”
password = “terrahs”
dbname = “Amazonia”
main:: IO()
main = do
      db <- open (TeMySQL host user password dbname)
      goSet1 <- retrieve db “desmatamento”
      goSet2 <- op goSet1
      store db “newlayer” goSet2
      close db
```

Roteiro

- Introdução
- Revisão (Mônadas e FFI)
- TerraHS
- Algebra de Mapas
- Conclusão

Álgebra de Mapas

- Objetivo principal:
  - Demonstra o desenvolvimento completo de um aplicativo real em programação funcional.
- Caso de teste:
  - Álgebra de mapas proposta em (Câmara, Palomo et al , 2005).
- Vantagens esperadas:
  - Genérica
  - Concisa

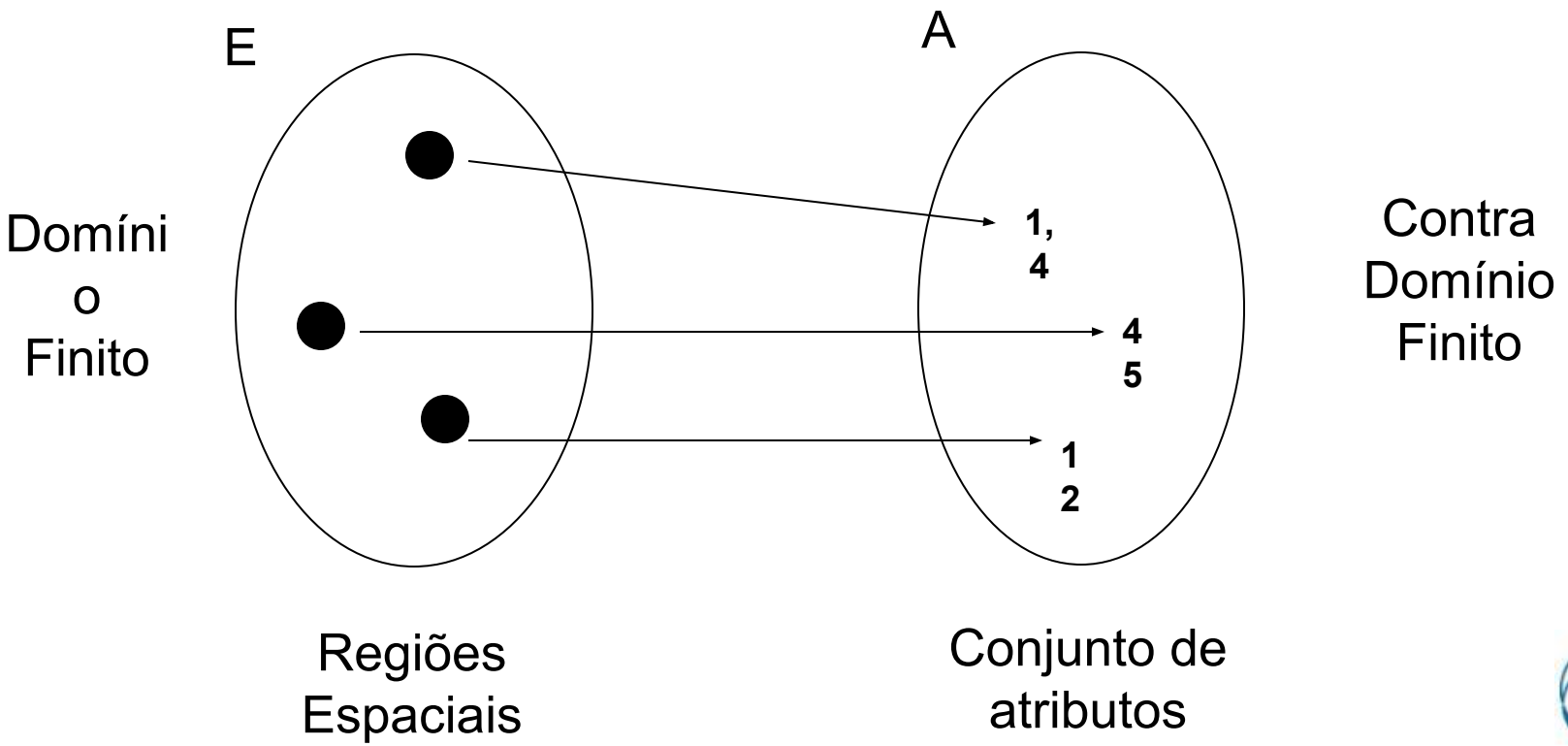
Álgebra de Mapas

- Implementação do tipo de dado.
  - Definido como um tipo abstrato de dados
- Implementação das operações
  - Operações não espaciais
  - Operações espaciais
- Validação com acesso a banco de dados

Álgebra de Mapas : Tipo de Dado (*Map*)

- Definição
  - Um *Map* é representado por uma função discreta:

$$m :: E \rightarrow A$$



Álgebra de Mapas : Tipo de Dado (*Map*)

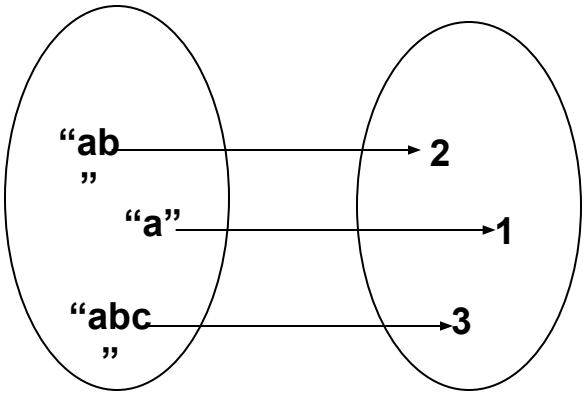
- Implementação em Haskell

  data Map a b = Map ((a → b), [a])

- Operações

module Maps (new_map,evaluate, domain, num , values, fun)

m1 :: (Map String Int)
m1 = new_map [”ab”,”abc”,”a”] length



values m1
⇒ [2,1,3]
evaluate m1 “ab”
⇒ Just 2
evaluate m1 “ad”
⇒ Nothing

Álgebra de Mapas : Operações não Espaciais

- Operações com único argumento
  - Exemplo: matemáticas (*log*, *exp*, *sin*, ...) ou funções de transformação.
- Implementação em Haskell
  - Assinatura:

```
map_single  :: (b → c) → (m a b) → (m a c)]
```

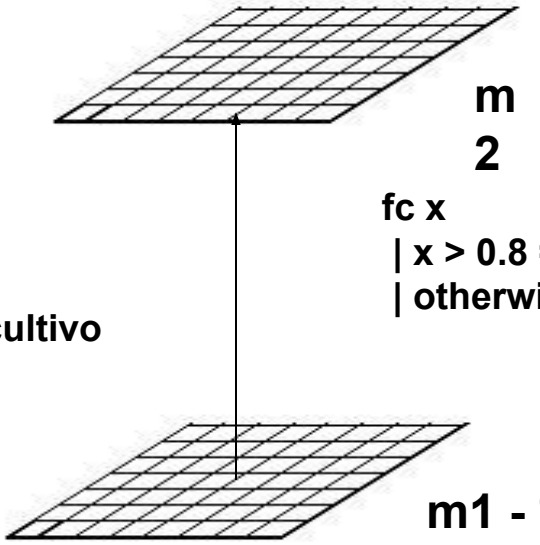
  - Axioma

```
map_single g m = new_map (domain m)  ( g . (fun m))
```

Álgebra de Mapas : Operações não Espaciais

- Operações com único argumento: Exemplo

values m2
⇒ [“AC”, “AC”, “AP”,”AC”,”AP”]



m
2

fc x
| x > 0.8 = "AP"
| otherwise = "AC"

AP: Área de proteção
AC: Área de cultivo

m2 = map_single fc m1

m1 - % de floresta

values m1
⇒ [0.6, 0.3, 0.9,0.4,0.85]

Álgebra de Mapas : Operações não Espaciais

- Operações com múltiplos argumentos
  - Exemplos: sum, product, or, ...
- Implementação em Haskell
  - Assinatura

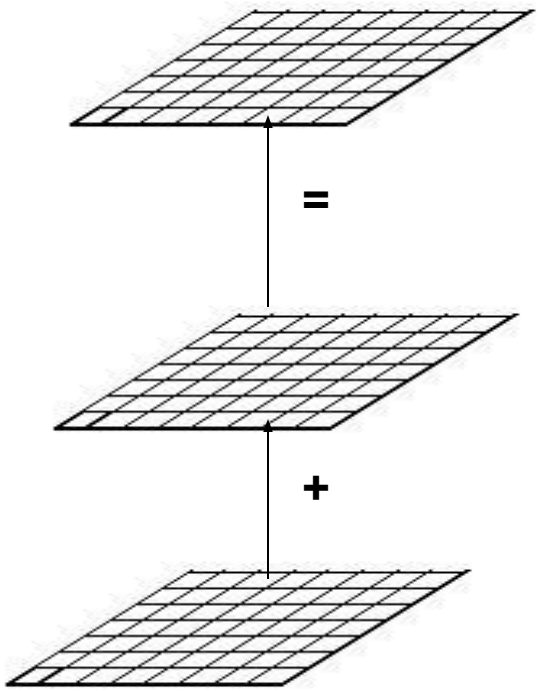
```
map_multiple:: ([b] → c) → [(m a b)] → (m a b)→(m a c)
```

  - Axioma

```
map_multiple fn mlist mref =
      new_map (domain mref) (\x →  fn (map_r mlist x))
```

Álgebra de Mapas : Operações não Espaciais

- Operações com múltiplos argumentos: Exemplo



m3 = map_multiple sum [m1,m2] m1
values m3
⇒ [6, 9, 18]

values m2
⇒ [4, 5, 10]

values m1
⇒ [2, 4, 8]

Álgebra de Mapas: Operações Espaciais

- Operações espaciais
  - Funções de alta ordem que usam predicados espaciais

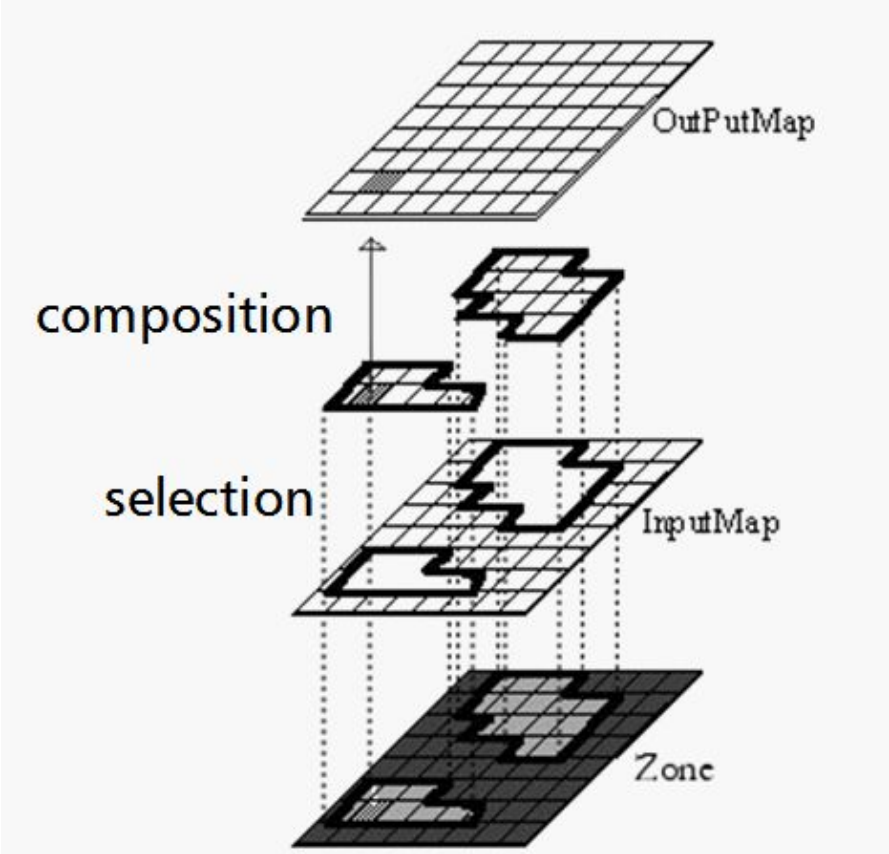


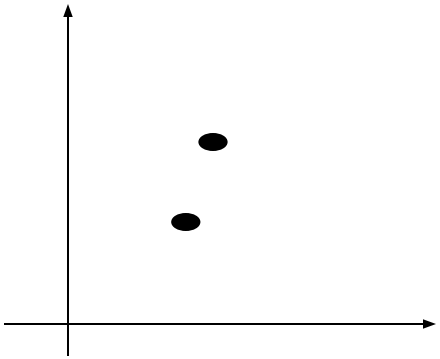
Figura 2: Operação de seleção- Fonte: Câmara, Palomo et al (2005)

Álgebra de Mapas: Operações Espaciais

- Operação de seleção

map_select :: (m a b) → (a → c→ Bool) → c → (m a b)

```
map_select m pred obj = new_map sel_dom (fun m)
where
   sel_dom = [elem | elem ← (domain m) , (pred elem obj)]
```



m3 = map_select m1 intersects m2

m1: conjunto de pontos
m2= apenas uma linha
m3= mapa resultante

Álgebra de Mapas: Operações Espaciais

- Operação de composição

map_compose :: ([b] → b) → (m a b) → b

map_compose f m = (f (values m))

⇒ 16

map_compose sum m1

values m1
⇒ [2, 6, 8]

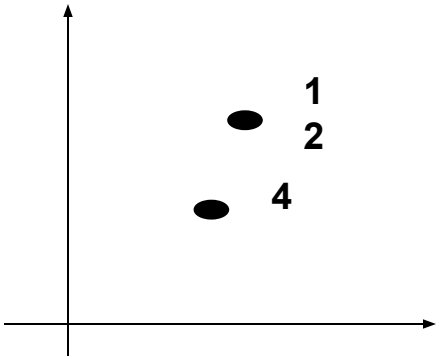
Álgebra de Mapas: Operações Espaciais

- Operação espacial

```
map_spatial :: ([b] → b) → (m a b) → (a → c → Bool)
→ (m c b) → (m c b)
```

```
map_spatial fn m pred mref = new_map (domain mref)
      (\x → map_compose fn (map_select m pred x))
```

```
values m3
⇒ [16]
```



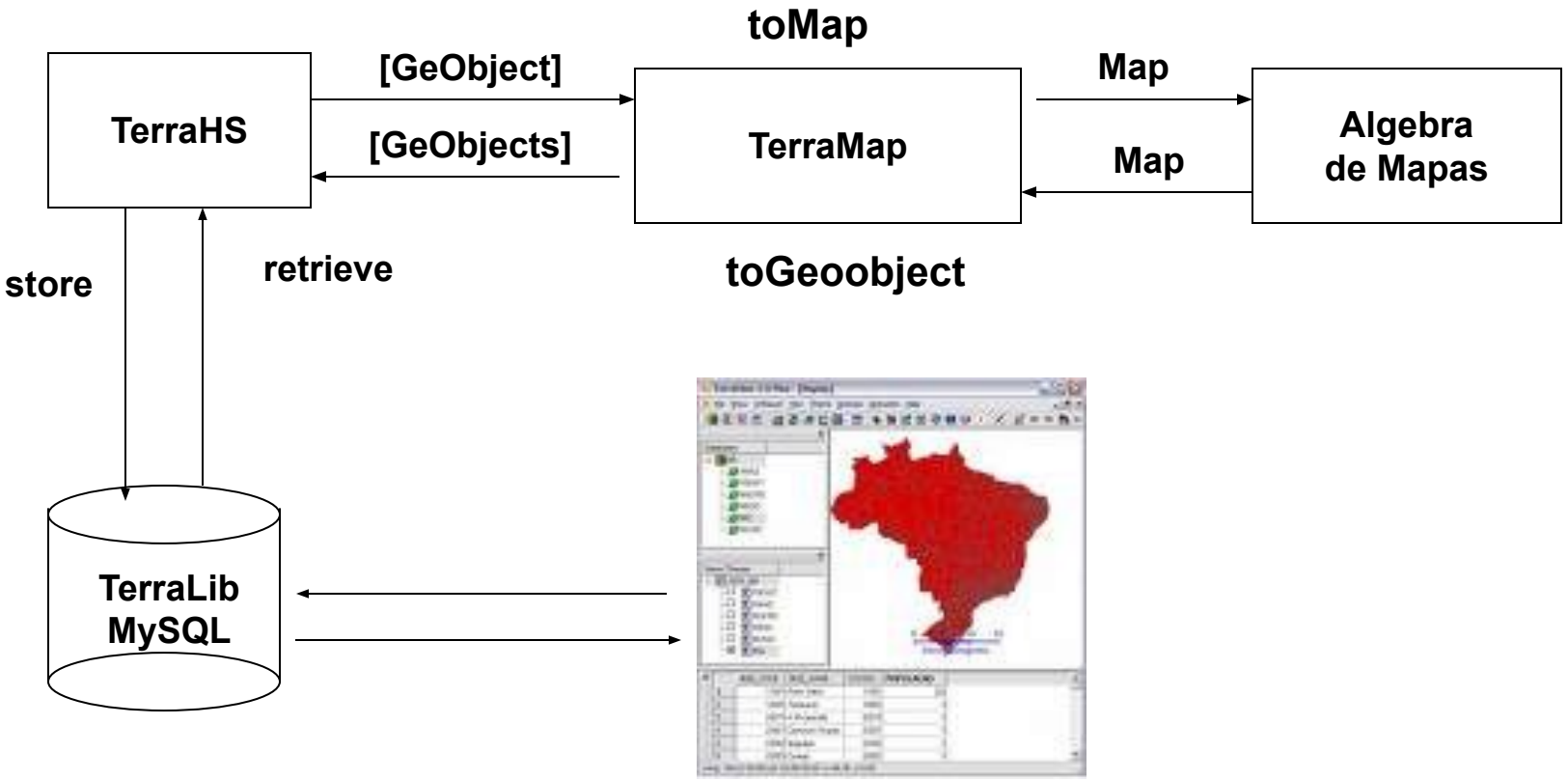

```
m3 = map_spatial sum m1 intersects m2
```

```
values m1
⇒ [5,12,4,2]
```

m1: conjunto de pontos
m2= apenas uma linha
m3= mapa resultante

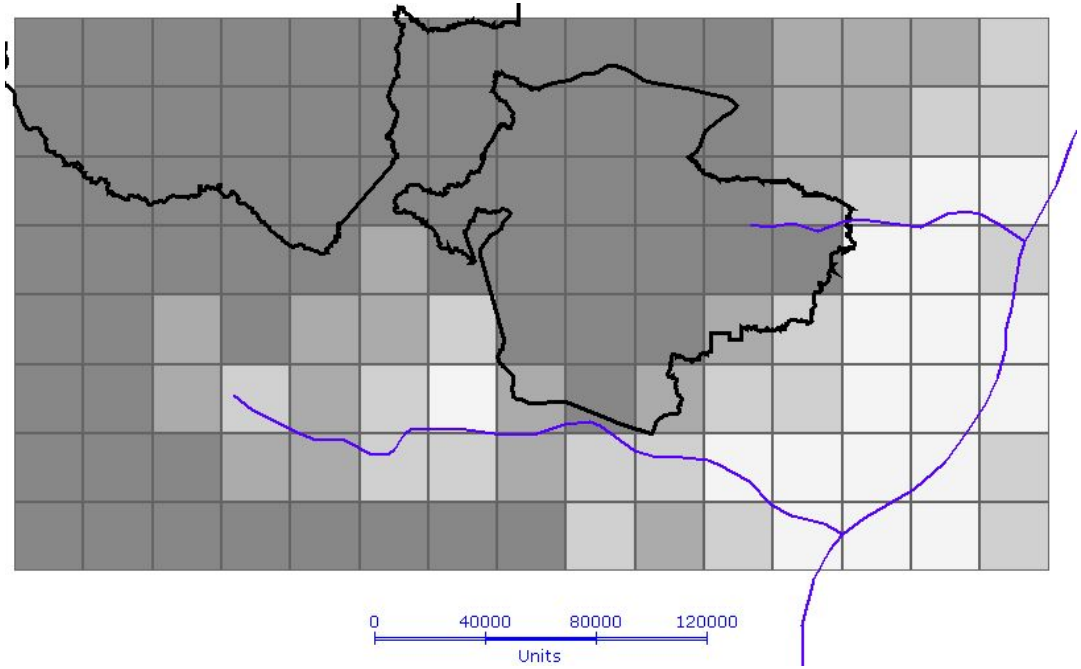
Álgebra de Mapas: Validação

- Compartilhando informações com o TerraView



Álgebra de Mapas: Validação

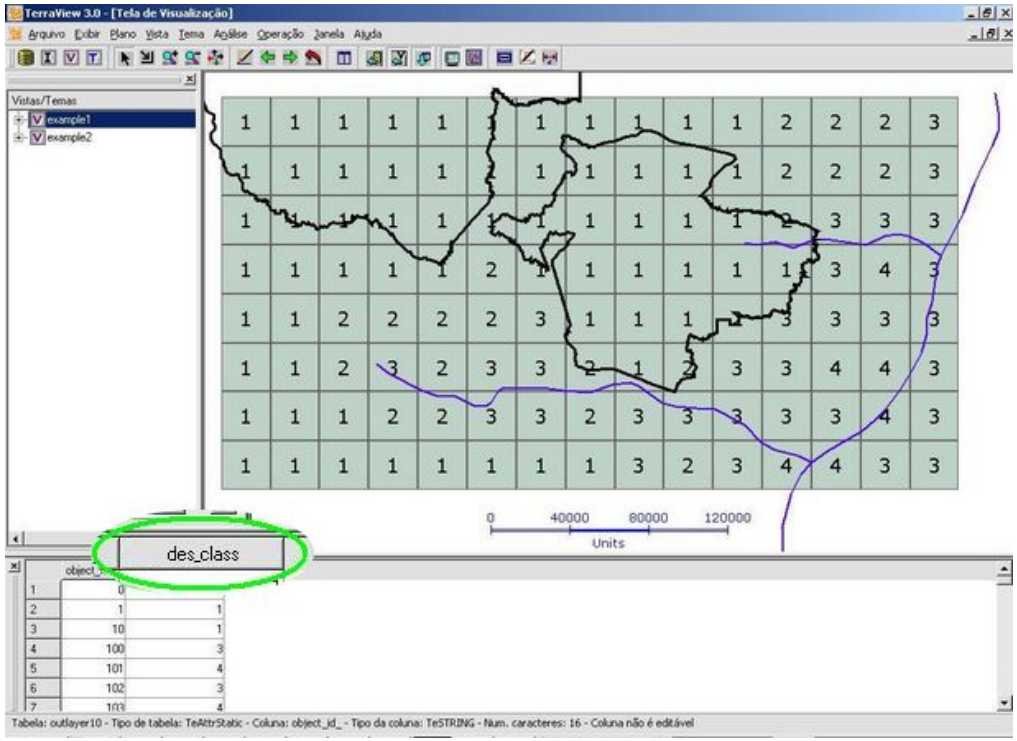
- Dados de teste (Aguiar, 2006):
  - Áreas de proteção ambiental – *prot_areas* (polígonos)
  - Estradas – *road_map* (linhas)
  - Percentual de desmatamento - *def_map* (espaço celular)



Álgebra de Mapas: Validação

Dado um mapa de desmatamento e uma função de classificação, retorne o mapa classificado

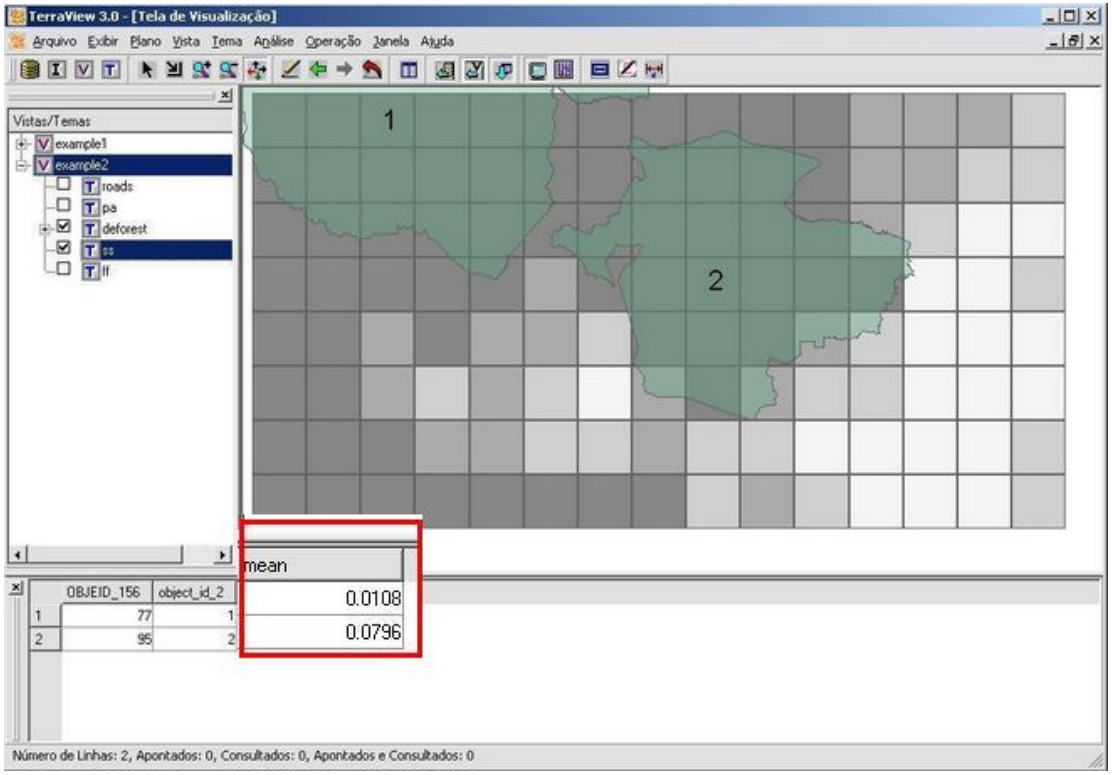
desf_class = map_single classify def_map



Álgebra de Mapas: Validação

Calcule a média de desmatamento por área de proteção.

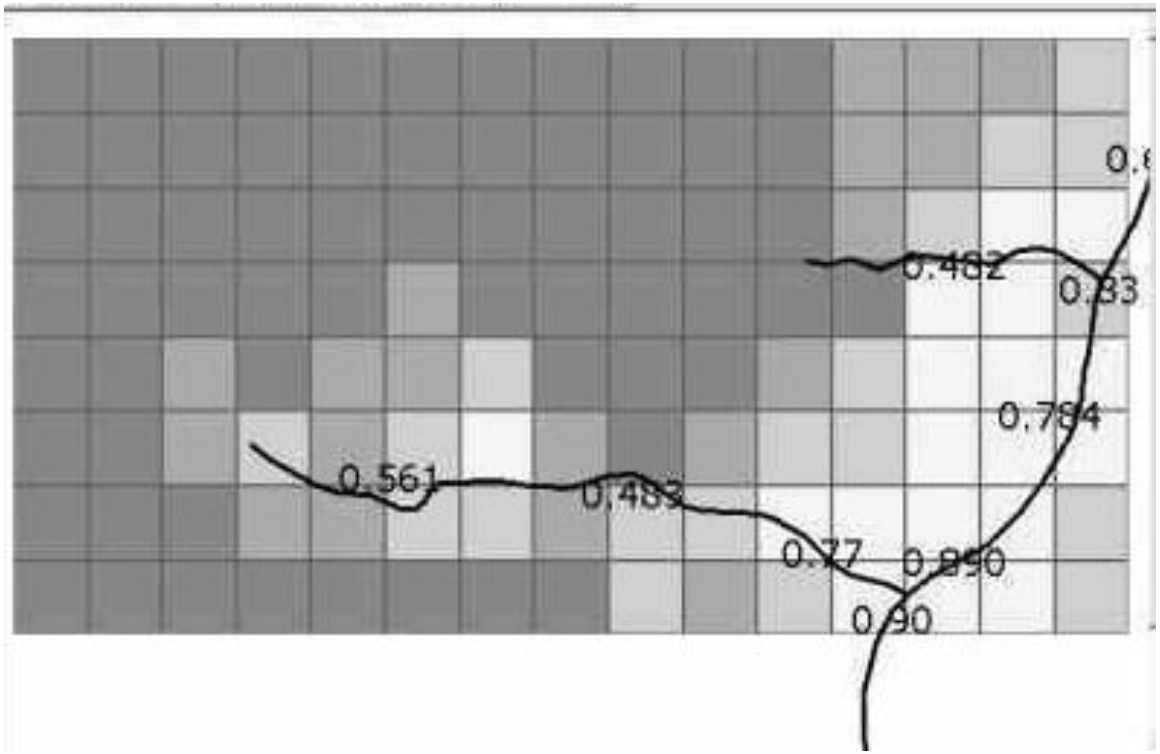
def_prot = map_spatial mean def_map within prot_areas



Álgebra de Mapas: Validação

Calcule a média de desmatamento ao logo das estradas

road_def = map_spatial mean def_map intersect road_map



Roteiro

- Introdução
- Breve Revisão (Mônadas e FFI)
- TerraHS
- Algebra de Mapas
- Conclusão

Conclusão

- Com o TerraHS, permitimos o uso de programação funcional em problemas reais de geoinformação
- Com o uso de uma linguagem funcional, mostramos que uma álgebra de mapas pode ser desenvolvida mais genérica e concisa.
- Linguagens funcionais e linguagens imperativas podem ser usadas de forma conjunta, aproveitando assim as vantagens de ambos paradigmas.

Conclusão

- Trabalhos futuros
  - **Acesso a banco de dados:** Versões futuras do TerraHS, deverá conter maiores recursos de acesso a banco de dados.
  - **Álgebras espaço-temporais:** Futuros trabalhos podem contemplar o desenvolvimento ou a validação de algumas álgebras espaço-temporais já propostas, como Guting (2005) ou Medak (2001)
  - **Integração com software gráficos:** Com o uso de uma interface gráfica pode-se ocultar do usuário as parte mais complexas do Haskell.

Agradecimentos

Obrigados a todos os presentes !!