Programação Funcional

TAD (Tipo Abstrato de Dados)

Introdução e Conceitos Básicos

- Definição de TAD (Tipo Abstrato de Dados)
- Importância da abstração de dados em Haskell
- Separação da implementação dos detalhes da estrutura de dados

Definição de TAD em Haskell

- Uso de tipos algébricos de dados para implementar TADs em Haskell
- Uma lista e uma pilha é uma estrutura de dados comum que pode ser facilmente representada usando um TAD em Haskell.

Exemplos simples de TADs como lista e pilha

Lista

```
1  data Lista a = Vazia | Cons a (Lista a)
2
```

- Nessa definição, temos dois construtores de dados: "Vazia", que representa uma lista vazia, e "Cons", que representa um elemento seguido de outra lista.
- O parâmetro "a" é um tipo de valor genérico, permitindo que a lista contenha elementos de qualquer tipo.

Pilha

```
1  data Pilha a = Vazia | Push a (Pilha a)
2
```

- Nessa definição, também temos dois construtores de dados: "Vazia", que representa uma pilha vazia, e "Push", que representa um elemento no topo da pilha seguido de outra pilha.
- Assim como na definição da lista, o parâmetro "a" permite que a pilha contenha elementos de qualquer tipo.

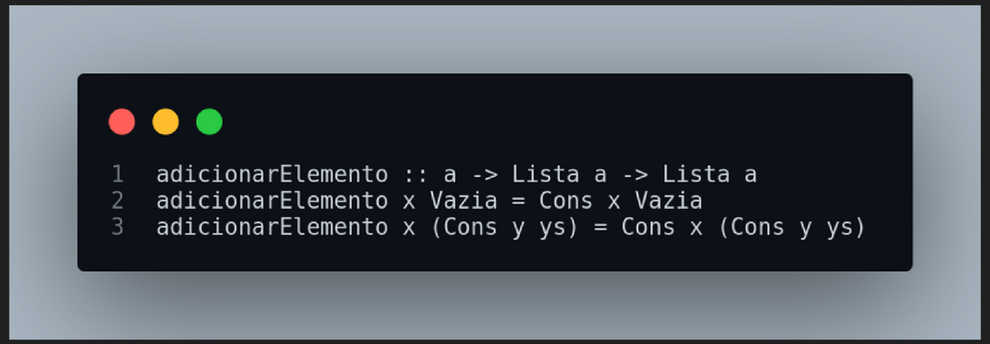
Operações comuns em TADs

- Adicionar um elemento: Para uma lista, podemos definir a função "adicionarElemento" que recebe um elemento e uma lista como parâmetros e retorna a lista com o elemento adicionado.
- Remover um elemento: Para uma pilha, podemos ter a função "removerElemento" que remove o elemento no topo da pilha e retorna a pilha modificada.
- Acessar elementos:Por exemplo, para uma lista, podemos ter a função "acessarElemento" que recebe um índice e uma lista, e retorna o elemento correspondente ao índice.
- Verificar vazio: Para uma pilha, podemos ter a função "estaVazia" que retorna um valor booleano indicando se a pilha está vazia.
- Tamanho: Para uma lista, podemos ter a função "tamanhoLista" que retorna a quantidade de elementos presentes na lista.

Definição de Funções para Manipular TADs

- As funções que manipulam os TADs são definidas fora da declaração do tipo de dado, mas operam sobre os valores desses tipos. Essas funções podem ser definidas usando padrões de correspondência (pattern matching) para tratar diferentes casos dos construtores de dados.

- Considerando o TAD da lista definido anteriormente, podemos implementar a função "adicionarElemento" da seguinte forma:



```
1  adicionarElemento :: a -> Lista a -> Lista a
2  adicionarElemento x Vazia = Cons x Vazia
3  adicionarElemento x (Cons y ys) = Cons x (Cons y ys)
```

Exemplos Práticos de Operações sobre TADs

```haskell
-- Adicionar um elemento à lista
adicionarElemento :: a -> Lista a -> Lista a
adicionarElemento x Vazia = Cons x Vazia
adicionarElemento x (Cons y ys) = Cons x (Cons y ys)

-- Remover um elemento da lista
removerElemento :: Eq a => a ->

```

Benefícios dos TADs em Haskell

- Modularidade e Reutilização de Código
- Abstração da Implementação
- Casos de Uso Específicos

Exemplos e Aplicações de TADs em Haskell

Árvores Binárias

```haskell
data ArvoreBinaria a = Folha a | No (ArvoreBinaria a) (ArvoreBinaria a)
```

Conjuntos

```haskell
data Conjunto a = Vazio | Inserir a (Conjunto a)
```

Grafos

```haskell
data Grafo a = Vertice a [Grafo a]
```

Bibliotecas e Ferramentas Relacionadas a TADs em Haskell

- Data.Set e Data.Map
- Data.Graph
- FGL (Functional Graph Library)

Conclusão

Dúvidas?