Cálculo de Programas
Trabalho Prático
LEI+MiEI — 2021/22

Departamento de Informática
Universidade do Minho

Fevereiro de 2022

Grupo nr.	4
a93290	Joana Maia Teixeira Alves
a93264	Maria Eugénia Bessa Cunha
a93296	Vicente Gonçalves Moreira

1 Preâmbulo

Cálculo de Programas tem como objectivo principal ensinar a programação de computadores como uma disciplina científica. Para isso parte-se de um repertório de *combinadores* que formam uma álgebra da programação (conjunto de leis universais e seus corolários) e usam-se esses combinadores para construir programas *composicionalmente*, isto é, agregando programas já existentes.

Na sequência pedagógica dos planos de estudo dos cursos que têm esta disciplina, opta-se pela aplicação deste método à programação em Haskell (sem prejuízo da sua aplicação a outras linguagens funcionais). Assim, o presente trabalho prático coloca os alunos perante problemas concretos que deverão ser implementados em Haskell. Há ainda um outro objectivo: o de ensinar a documentar programas, a validá-los e a produzir textos técnico-científicos de qualidade.

Antes de abodarem os problemas propostos no trabalho, os grupos devem ler com atenção o anexo A onde encontrarão as instruções relativas ao sofware a instalar, etc.

Problema 1

Num sistema de informação distribuído, uma lista não vazia de transações é vista como um *blockchain* sempre que possui um valor de *hash* que é dado pela raiz de uma Merkle tree que lhe está associada. Isto significa que cada *blockchain* está estruturado numa Merkle tree. Mas, o que é uma Merkle tree?

Uma Merkle tree é uma *FTree* com as seguintes propriedades:

1. as folhas são pares ($hash$, transação) ou simplesmente o $hash$ de uma transação;
2. os nodos são $hashes$ que correspondem à concatenação dos $hashes$ dos filhos;
3. o $hash$ que se encontra na raiz da árvore é designado $Merkle\ \ Root$; como se disse acima, corresponde ao valor de $hash$ de todo o bloco de transações.

(1)

Assumindo uma lista não vazia de transações, o algoritmo clássico de construção de uma $Merkle\ \ Tree$ é o que está dado na Figura 1. Contudo, este algoritmo (que se pode mostrar ser um hilomorfismo de listas não vazias) é demasiadamente complexo. Uma forma bem mais simples de produzir uma $Merkle\ \ Tree$ é através de um hilomorfismo de *LTree*s. Começa-se por, a partir da lista de transações, construir uma *LTree* cujas folhas são as transações:

$$list2LTree :: [a] \rightarrow \mathsf{LTree}\ a$$

Depois, o objetivo é etiquetar essa árvore com os $hashes$,

- *Se a lista for singular, calcular o* $hash$ *da transação.*
- *Caso contrário,*
  1. *Mapear a lista com a função* $hash$*.*
  2. *Se o comprimento da lista for ímpar, concatenar a lista com o seu último valor (que fica duplicado). Caso contrário, a lista não sofre alterações.*
  3. *Agrupar a lista em pares.*
  4. *Concatenar os* $hashes$ *do par produzindo uma lista de (sub-)árvores nas quais a cabeça terá a respetiva concatenação.*
  5. *Se a lista de (sub-)árvores não for singular, voltar ao passo 2 com a lista das cabeças como argumento, preservando a lista de (sub-)árvores. Se a lista for singular, chegamos à* $Merkle\ Root$*. Contudo, falta compor a* $Merkle\ Tree$ *final. Para tal, tendo como resultado uma lista de listas de (sub-)árvores agrupada pelos níveis da árvore final, é necessário encaixar sucessivamente os tais níveis formando a* $Merkle\ Tree$ *completa.*

Figura 1: Algoritmo clássico de construção de uma Merkle tree [4] .

$$lTree2MTree :: Hashable\ a \Rightarrow LTree\ a \rightarrow \underbrace{FTree\ \mathbb{Z}\ (\mathbb{Z}, a)}_{Merkle\ tree}$$

formando uma Merkle tree que satisfaça os três requisitos em (1). Em suma, a construção de um blockchain é um hilomorfismo de *LTree*s

$$computeMerkleTree :: Hashable\ a \Rightarrow [\,a\,] \rightarrow FTree\ \mathbb{Z}\ (\mathbb{Z}, a)$$
$$computeMerkleTree = lTree2MTree \cdot list2LTree$$

1. Comece por definir o gene do anamorfismo que constrói *LTree*s a partir de listas não vazias:

   $$list2LTree :: [\,a\,] \rightarrow LTree\ a$$
   $$list2LTree = [\!(g\_list2LTree)\!]$$

   NB: para garantir que $list2LTree$ não aceita listas vazias deverá usar em $g\_list2LTree$ o inverso $outNEList$ do isomorfismo

   $$inNEList = [singl, cons]$$

2. Assumindo as seguintes funções $hash$ e $concHash$:[1]

   $$hash :: Hashable\ a \Rightarrow a \rightarrow \mathbb{Z}$$
   $$hash = toInteger \cdot (Data.Hashable.hash)$$
   $$concHash :: (\mathbb{Z}, \mathbb{Z}) \rightarrow \mathbb{Z}$$
   $$concHash = add$$

   defina o gene do catamorfismo que consome a *LTree* e produz a correspondente Merkle tree etiquetada com todos os $hashes$:

   $$lTree2MTree :: Hashable\ a \Rightarrow LTree\ a \rightarrow FTree\ \mathbb{Z}\ (\mathbb{Z}, a)$$
   $$lTree2MTree = (\!|\, g\_lTree2MTree \,|\!)$$

3. Defina g_mroot por forma a

   $$mroot :: Hashable\ b \Rightarrow [\,b\,] \rightarrow \mathbb{Z}$$
   $$mroot = (\!|\, g\_mroot \,|\!) \cdot computeMerkleTree$$

   nos dar a Merkle *root* de um qualquer bloco $[\,b\,]$ de transações.

[1] Para invocar a função $hash$, escreva $Main.hash$.

4. Calcule $mroot\ trs$ da sequência de transações trs da no anexo e verifique que, sempre que se modifica (e.g. fraudulentamente) uma transação passada em trs, $mroot\ trs$ altera-se necessariamente. Porquê? (Esse é exactamente o princípio de funcionamento da tecnologia blockchain.)

Valorização (não obrigatória): implemente o algoritmo clássico de construção de Merkle trees

$$classicMerkleTree :: Hashable\ a \Rightarrow [a] \rightarrow FTree\ \mathbb{Z}\ \mathbb{Z}$$

sob a forma de um hilomorfismo de listas não vazias. Para isso deverá definir esse combinador primeiro, da forma habitual:

$$hyloNEList\ h\ g = cataNEList\ h \cdot anaNEList\ g$$

etc. Depois passe à definição do gene $g\_pairsList$ do anamorfismo de listas

$$pairsList :: [a] \rightarrow [(a, a)]$$
$$pairsList = [\![(g\_pairsList)]\!]$$

que agrupa a lista argumento por pares, duplicando o último valor caso seja necessário. Para tal, poderá usar a função (já definida)

$$getEvenBlock :: [a] \rightarrow [a]$$

que, dada uma lista, se o seu comprimento for ímpar, duplica o último valor.

Por fim, defina os genes $divide$ e $conquer$ dos respetivos anamorfismo e catamorfimo por forma a

$$classicMerkleTree = (hyloNEList\ conquer\ divide) \cdot (\mathsf{map}\ \ Main.hash)$$

Para facilitar a definição do $conquer$, terá apenas de definir o gene $g\_mergeMerkleTree$ do catamorfismo de ordem superior

$$mergeMerkleTree :: FTree\ a\ p \rightarrow [FTree\ a\ c] \rightarrow FTree\ a\ c$$
$$mergeMerkleTree = (\!|\ g\_mergeMerkleTree\ |\!)$$

que compõe a $FTree$ (à cabeça) com a lista de $FTree$s (como filhos), fazendo um "merge" dos valores intermédios. Veja o seguinte exemplo de aplicação da função $mergeMerkleTree$:

```
> l = [Comp 3 (Unit 1, Unit 2), Comp 7 (Unit 3, Unit 4)]
>
> m = Comp 10 (Unit 3, Unit 7)
>
> mergeMerkleTree m l
Comp 10 (Comp 3 (Unit 1,Unit 2),Comp 7 (Unit 3,Unit 4))
```

NB: o *classicMerkleTree* retorna uma Merkle Tree cujas folhas são apenas o $hash$ da transação e não o par ($hash$, transação).

Problema 2

Se se digitar $man\ wc$ na shell do Unix (Linux) obtém-se:

```
NAME
     wc -- word, line, character, and byte count

SYNOPSIS
     wc [-clmw] [file ...]

DESCRIPTION
    The wc utility displays the number of lines, words, and bytes contained in
    each input file,  or standard input (if no file is specified) to the stan-
    dard  output.  A line is defined as  a string of characters delimited by a
    <newline> character.  Characters beyond the final <newline> character will
    not be included in the line count.
    (...)
```

```
The following options are available:
(...)
    -w   The number of words in each input file is written to the standard
         output.
(...)
```

Se olharmos para o código da função que, em C, implementa esta funcionalidade [1] e nos focarmos apenas na parte que implementa a opção `-w`, verificamos que a poderíamos escrever, em Haskell, da forma seguinte:

$$
\begin{array}{l}
\mathit{wc\_w} :: [\mathit{Char}] \rightarrow \mathit{Int} \\
\mathit{wc\_w}\ [\,] = 0 \\
\mathit{wc\_w}\ (c : l) = \\
\quad \textbf{if}\ \neg\ (\mathit{sep}\ c) \wedge \mathit{lookahead\_sep}\ l\ \textbf{then}\ \mathit{wc\_w}\ l + 1\ \textbf{else}\ \mathit{wc\_w}\ l \\
\qquad \textbf{where} \\
\qquad\quad \mathit{sep}\ c = (c \equiv \texttt{' '} \vee c \equiv \texttt{'\textbackslash n'} \vee c \equiv \texttt{'\textbackslash t'}) \\
\qquad\quad \mathit{lookahead\_sep}\ [\,] = \mathit{True} \\
\qquad\quad \mathit{lookahead\_sep}\ (c : l) = \mathit{sep}\ c
\end{array}
$$

Por aplicação da lei de recursividade mútua

$$
\left\{ \begin{array}{l} f \cdot \mathsf{in} = h \cdot \mathsf{F}\ \langle f, g \rangle \\ g \cdot \mathsf{in} = k \cdot \mathsf{F}\ \langle f, g \rangle \end{array} \right. \quad \equiv \quad \langle f, g \rangle = (\!|\langle h, k \rangle|\!) \tag{2}
$$

às funções $\mathit{wc\_w}$ e $\mathit{lookahead\_sep}$, re-implemente a primeira segundo o modelo *worker/wrapper* onde *worker* deverá ser um catamorfismo de listas:

$$
\begin{array}{l}
\mathit{wc\_w\_final} :: [\mathit{Char}] \rightarrow \mathit{Int} \\
\mathit{wc\_w\_final} = \mathit{wrapper} \cdot \underbrace{(\!|[\mathit{g1}, \mathit{g2}]|\!)}_{\mathit{worker}}
\end{array}
$$

Apresente os cálculos que fez para chegar à versão $\mathit{wc\_w\_final}$ de $\mathit{wc\_w}$, com indicação dos genes h, k e $g = [\mathit{g1}, \mathit{g2}]$.

Problema 3

Neste problema pretende-se gerar o HTML de uma página de um jornal descrita como uma agregação estruturada de blocos de texto ou imagens:

$$\textbf{data}\ \mathit{Unit}\ a\ b = \mathit{Image}\ a \mid \mathit{Text}\ b\ \textbf{deriving}\ \mathit{Show}$$

O tipo *Sheet* (="página de jornal")

$$\textbf{data}\ \mathit{Sheet}\ a\ b\ i = \mathit{Rect}\ (\mathit{Frame}\ i)\ (X\ (\mathit{Unit}\ a\ b)\ (\mathit{Mode}\ i))\ \textbf{deriving}\ \mathit{Show}$$

é baseado num tipo indutivo X que, dado em anexo (pág. 10), exprime a partição de um rectângulo (a página tipográfica) em vários subrectângulos (as caixas tipográficas a encher com texto ou imagens), segundo um processo de partição binária, na horizontal ou na vertical. Para isso, o tipo

$$\textbf{data}\ \mathit{Mode}\ i = \mathit{Hr}\ i \mid \mathit{Hl}\ i \mid \mathit{Vt}\ i \mid \mathit{Vb}\ i\ \textbf{deriving}\ \mathit{Show}$$

especifica quatro variantes de partição. O seu argumento deverá ser um número de 0 a 1, indicando a fracção da altura (ou da largura) em que o rectângulo é dividido, a saber:

- `Hr i` — partição horizontal, medindo i a partir da direita
- `Hl i` — partição horizontal, medindo i a partir da esquerda
- `Vt i` — partição vertical, medindo i a partir do topo
- `Vb i` — partição vertical, medindo i a partir da base

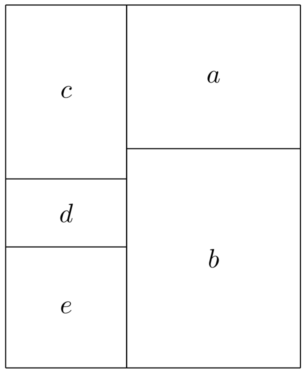
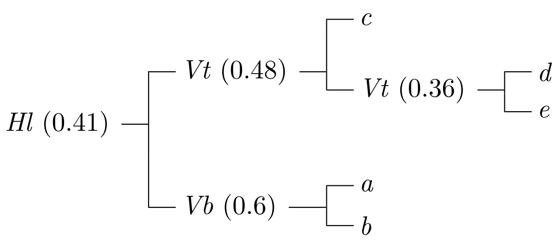


Figura 2: Layout de página de jornal.

Por exemplo, a partição dada na figura 2 corresponde à partição de um rectângulo de acordo com a seguinte árvore de partições:



As caixas delineadas por uma partição (como a dada acima) correspondem a folhas da árvore de partição e podem conter texto ou imagens. É o que se verifica no objecto $example$ da secção B que, processado por $sheet2html$ (secção B) vem a produzir o ficheiro `jornal.html`.

O que se pretende O código em Haskell fornecido no anexo B como "kit" para arranque deste trabalho não está estruturado em termos dos combinadores *cata-ana-hylo* estudados nesta disciplina. O que se pretende é, então:

1. A construção de uma biblioteca "pointfree" [2] com base na qual o processamento ("pointwise") já disponível possa ser redefinido.
2. A evolução da biblioteca anterior para uma outra que permita partições n-árias (para *qualquer* n finito) e não apenas binárias. [3]

Problema 4

Este exercício tem como objectivo determinar todos os caminhos possíveis de um ponto A para um ponto B. Para tal, iremos utilizar técnicas de *brute force* e *backtracking*, que podem ser codificadas no mónade das listas (estudado na aulas). Comece por implementar a seguinte função auxiliar:

1. $pairL :: [a] \rightarrow [(a, a)]$ que dada uma lista l de tamanho maior que 1 produz uma nova lista cujos elementos são os pares (x, y) de elementos de l tal que x precede imediatamente y. Por exemplo:

   $pairL\ [1, 2] \equiv [(1, 2)]$,
   $pairL\ [1, 2, 3] \equiv [(1, 2), (2, 3)]$ e
   $pairL\ [1, 2, 3, 4] \equiv [(1, 2), (2, 3), (3, 4)]$

   Para o caso em que $l = [x]$, i.e. o tamanho de l é 1, assuma que $pairL\ [x] \equiv [(x, x)]$. Implemente esta função como um *anamorfismo de listas*, atentando na sua propriedade:

[2] A desenvolver de forma análoga a outras bibliotecas que conhece (*eg.* *LTree*, etc).

[3] Repare que é a falta desta capacidade expressiva que origina, no "kit" actual, a definição das funções auxiliares da secção B, por exemplo.

- *Para todas as listas l de tamanho maior que 1, a lista* `map` π_1 $(pairL\ l)$ *é a lista original l a menos do último elemento. Analogamente, a lista* `map` π_2 $(pairL\ l)$ *é a lista original l a menos do primeiro elemento.*

De seguida necessitamos de uma estrutura de dados representativa da noção de espaço, para que seja possível formular a noção de *caminho* de um ponto A para um ponto B, por exemplo, num papel quadriculado. No nosso caso vamos ter:

$$\mathbf{data}\ Cell = Free \mid Blocked \mid Lft \mid Rght \mid Up \mid Down\ \mathbf{deriving}\ (Eq, Show)$$
$$\mathbf{type}\ Map = [[Cell]]$$

O terreno onde iremos navegar é codificado então numa *matriz* de células. Os valores *Free* and *Blocked* denotam uma célula como livre ou bloqueada, respectivamente (a navegação entre dois pontos terá que ser realizada *exclusivamente* através de células livres). Ao correr, por exemplo, $putStr\ \$\ showM\ \$\ map_1$ no interpretador irá obter a seguinte apresentação de um mapa:

```
_ _ _
_ X _
_ X _
```

Para facilitar o teste das implementações pedidas abaixo, disponibilizamos no anexo B a função $testWithRndMap$. Por exemplo, ao correr $testWithRndMap$ obtivemos o seguinte mapa aleatoriamente:

```
 _ _ _ _ _ _ X _ _ X
 _ X _ _ _ X _ _ _ _
 _ _ _ _ _ X _ _ _ _
 _ X _ _ _ _ _ _ _ X
 _ _ _ _ _ _ X _ X _
 _ _ _ _ _ _ _ _ _ _
 _ X X _ _ X _ _ _ _
 _ _ _ _ _ _ _ _ X _
 _ _ _ _ _ X _ _ X _
 _ _ X _ _ _ _ _ _ X
Map of dimension 10x10.
```

De seguida, os valores *Lft*, *Rght*, *Up* e *Down* em *Cell* denotam o facto de uma célula ter sido alcançada através da célula à esquerda, direita, de cima, ou de baixo, respectivamente. Tais valores irão ser usados na representação de caminhos num mapa.

2. Implemente agora a função $markMap :: [Pos] \rightarrow Map \rightarrow Map$, que dada uma lista de posições (representante de um *caminho* de um ponto A para um ponto B) e um mapa retorna um novo mapa com o caminho lá marcado. Por exemplo, ao correr no interpretador,

$$putStr\ \$\ showM\ \$\ markMap\ [(0,0),(0,1),(0,2),(1,2)]\ map_1$$

deverá obter a seguinte apresentação de um mapa e respectivo caminho:

```
> _ _
^ X _
^ X _
```

representante do caso em que subimos duas vezes no mapa e depois viramos à direita. Para implementar a função $markMap$ deverá recorrer à função $toCell$ (disponibilizada no anexo B) e a uma função auxiliar com o tipo $[(Pos, Pos)] \rightarrow Map \rightarrow Map$ definida como um *catamorfismo de listas*. Tal como anteriormente, anote as propriedades seguintes sobre $markMap$:[4]

- *Para qualquer lista l a função $markMap\ l$ é idempotente.*
- *Todas as posições presentes na lista dada como argumento irão fazer com que as células correspondentes no mapa deixem de ser Free.*

[4] Ao implementar a função $markMap$, estude também a função $subst$ (disponibilizada no anexo B) pois as duas funções tem algumas semelhanças.

Finalmente há que implementar a função $scout :: Map \to Pos \to Pos \to Int \to [[Pos]]$, que dado um mapa m, uma posição inicial s, uma posição alvo t, e um número inteiro n, retorna uma lista de caminhos que começam em s e que têm tamanho máximo $n + 1$. Nenhum destes caminhos pode conter t como elemento que não seja o último na lista (i.e. um caminho deve terminar logo que se alcança a posição t). Para além disso, não é permitido voltar a posições previamente visitadas e se ao alcançar uma posição diferente de t é impossivel sair dela então todo o caminho que levou a esta posição deve ser removido (*backtracking*). Por exemplo:

$scout\ map_1\ (0,0)\ (2,0)\ 0 \equiv [[(0,0)]]$
$scout\ map_1\ (0,0)\ (2,0)\ 1 \equiv [[(0,0),(0,1)]]$
$scout\ map_1\ (0,0)\ (2,0)\ 4 \equiv [[(0,0),(0,1),(0,2),(1,2),(2,2)]]$
$scout\ map_2\ (0,0)\ (2,2)\ 2 \equiv [[(0,0),(0,1),(1,1)],[(0,0),(0,1),(0,2)]]$
$scout\ map_2\ (0,0)\ (2,2)\ 4 \equiv [[(0,0),(0,1),(1,1),(2,1),(2,2)],[(0,0),(0,1),(1,1),(2,1),(2,0)]]$

3. Implemente a função

$$scout :: Map \to Pos \to Pos \to Int \to [[Pos]]$$

recorrendo à função $checkAround$ (disponibilizada no anexo B) e de tal forma a que $scout\ \ m\ \ s\ \ t$ seja um catamorfismos de naturais *monádico*. Anote a seguinte propriedade desta função:

- *Quanto maior for o tamanho máximo permitido aos caminhos, mais caminhos que alcançam a posição alvo iremos encontrar.*

Anexos

A Documentação para realizar o trabalho

Para cumprir de forma integrada os objectivos Rdo trabalho vamos recorrer a uma técnica de programação dita "literária" [2], cujo princípio base é o seguinte:

Um programa e a sua documentação devem coincidir.

Por outras palavras, o código fonte e a documentação de um programa deverão estar no mesmo ficheiro.

O ficheiro `cp2122t.pdf` que está a ler é já um exemplo de programação literária: foi gerado a partir do texto fonte `cp2122t.lhs`[5] que encontrará no material pedagógico desta disciplina descompactando o ficheiro `cp2122t.zip` e executando:

```
$ lhs2TeX cp2122t.lhs > cp2122t.tex
$ pdflatex cp2122t
```

em que `lhs2tex` é um pre-processador que faz "pretty printing" de código Haskell em LaTeX e que deve desde já instalar executando

```
$ cabal install lhs2tex --lib
$ cabal install --ghc-option=-dynamic lhs2tex
```

NB: utilizadores do macOS poderão instalar o *cabal* com o seguinte comando:

```
$ brew install cabal-install
```

Por outro lado, o mesmo ficheiro `cp2122t.lhs` é executável e contém o "kit" básico, escrito em Haskell, para realizar o trabalho. Basta executar

```
$ ghci cp2122t.lhs
```

Abra o ficheiro `cp2122t.lhs` no seu editor de texto preferido e verifique que assim é: todo o texto que se encontra dentro do ambiente

```
\begin{code}
...
\end{code}
```

é seleccionado pelo GHCi para ser executado.

A.1 Como realizar o trabalho

Este trabalho teórico-prático deve ser realizado por grupos de 3 (ou 4) alunos. Os detalhes da avaliação (datas para submissão do relatório e sua defesa oral) são os que forem publicados na página da disciplina na *internet*.

Recomenda-se uma abordagem participativa dos membros do grupo em todos os exercícios do trabalho, para assim poderem responder a qualquer questão colocada na *defesa oral* do relatório.

Em que consiste, então, o *relatório* a que se refere o parágrafo anterior? É a edição do texto que está a ser lido, preenchendo o anexo C com as respostas. O relatório deverá conter ainda a identificação dos membros do grupo de trabalho, no local respectivo da folha de rosto.

Para gerar o PDF integral do relatório deve-se ainda correr os comando seguintes, que actualizam a bibliografia (com BibTeX) e o índice remissivo (com `makeindex`),

```
$ bibtex cp2122t.aux
$ makeindex cp2122t.idx
```

e recompilar o texto como acima se indicou. Dever-se-á ainda instalar o utilitário QuickCheck, que ajuda a validar programas em Haskell:

```
$ cabal install QuickCheck --lib
```

Para testar uma propriedade QuickCheck *prop*, basta invocá-la com o comando:

[5] O sufixo 'lhs' quer dizer *literate Haskell*.

```
> quickCheck prop
+++ OK, passed 100 tests.
```

Pode-se ainda controlar o número de casos de teste e sua complexidade, como o seguinte exemplo mostra:[6]

```
> quickCheckWith stdArgs { maxSuccess = 200, maxSize = 10 } prop
+++ OK, passed 200 tests.
```

Qualquer programador tem, na vida real, de ler e analisar (muito!) código escrito por outros. No anexo B disponibiliza-se algum código Haskell relativo aos problemas que se seguem. Esse anexo deverá ser consultado e analisado à medida que isso for necessário.

Stack O Stack é um programa útil para criar, gerir e manter projetos em Haskell. Um projeto criado com o Stack possui uma estrutura de pastas muito específica:

- Os módulos auxiliares encontram-se na pasta *src*.
- O módulo principal encontra-se na pasta *app*.
- A lista de dependências externas encontra-se no ficheiro *package.yaml*.

Pode aceder ao GHCi utilizando o comando:

```
stack ghci
```

Garanta que se encontra na pasta mais externa **do projeto**. A primeira vez que correr este comando as depêndencias externas serão instaladas automaticamente. Para gerar o PDF, garanta que se encontra na diretoria *app*.

A.2 Como exprimir cálculos e diagramas em LaTeX/lhs2tex

Como primeiro exemplo, estudar o texto fonte deste trabalho para obter o efeito:[7]

$$
\begin{array}{ll}
& id = \langle f, g \rangle \\
\equiv & \qquad \{ \text{ universal property } \} \\
& \left\{ \begin{array}{l} \pi_1 \cdot id = f \\ \pi_2 \cdot id = g \end{array} \right. \\
\equiv & \qquad \{ \text{ identity } \} \\
& \left\{ \begin{array}{l} \pi_1 = f \\ \pi_2 = g \end{array} \right. \\
\Box &
\end{array}
$$

Os diagramas podem ser produzidos recorrendo à *package* LaTeX xymatrix, por exemplo:

$$
\begin{tikzcd}
\mathbb{N}_0 \arrow[d, "(\!|g|\!)"'] & & 1 + \mathbb{N}_0 \arrow[ll, "\mathsf{in}"'] \arrow[d, "id + (\!|g|\!)"] \\
B & & 1 + B \arrow[ll, "g"]
\end{tikzcd}
$$

[6]Como já sabe, os testes normalmente não provam a ausência de erros no código, apenas a sua presença (cf. arquivo online). Portanto não deve ver o facto de o seu código passar nos testes abaixo como uma garantia que este está livre de erros.

[7]Exemplos tirados de [3].

B Código fornecido

Problema 1

Sequência de transações para teste:

$$
\begin{array}{l}
trs = [(\texttt{"compra"}, \texttt{"20211102"}, -50),\\
\quad (\texttt{"venda"}, \quad \texttt{"20211103"}, 100),\\
\quad (\texttt{"despesa"}, \texttt{"20212103"}, -20),\\
\quad (\texttt{"venda"}, \quad \texttt{"20211205"}, 250),\\
\quad (\texttt{"venda"}, \quad \texttt{"20211205"}, 120)]\\
trsmoded = [(\texttt{"compra"}, \texttt{"20211102"}, -50),\\
\quad (\texttt{"venda"}, \quad \texttt{"20211103"}, 100),\\
\quad (\texttt{"despesa"}, \texttt{"20212103"}, -20),\\
\quad (\texttt{"venda"}, \quad \texttt{"20211205"}, 250),\\
\quad (\texttt{"venda"}, \quad \texttt{"20211206"}, 120)]
\end{array}
$$

$$
\begin{array}{l}
getEvenBlock :: [a] \rightarrow [a]\\
getEvenBlock\ l = \mathbf{if}\ (even\ (length\ l))\ \mathbf{then}\ l\ \mathbf{else}\ l +\!\!+ [last\ l]\\
firsts = [\pi_1, \pi_1]
\end{array}
$$

Problema 2

$$
\begin{array}{l}
wc\_test = \texttt{"Here is a sentence, for testing.\textbackslash nA short one."}\\
sp\ c = (c \equiv \texttt{' '} \vee c \equiv \texttt{'\textbackslash n'} \vee c \equiv \texttt{'\textbackslash t'})
\end{array}
$$

Problema 3

Tipos:

$$
\begin{array}{l}
\mathbf{data}\ X\ u\ i = XLeaf\ u \mid Node\ i\ (X\ u\ i)\ (X\ u\ i)\ \mathbf{deriving}\ Show\\
\mathbf{data}\ Frame\ i = Frame\ i\ i\ \mathbf{deriving}\ Show
\end{array}
$$

Funções da API[8]

$$
\begin{array}{l}
printJournal :: Sheet\ String\ String\ Double \rightarrow \mathsf{IO}\ ()\\
printJournal = write \cdot sheet2html\\
write :: String \rightarrow \mathsf{IO}\ ()\\
write\ s = \mathbf{do}\ writeFile\ \texttt{"jornal.html"}\ s\\
\quad putStrLn\ \texttt{"Output HTML written into file `jornal.html'"}
\end{array}
$$

Geração de HTML:

$$
\begin{array}{l}
sheet2html\ (Rect\ (Frame\ w\ h)\ y) = htmlwrap\ (x2html\ y\ (w, h))\\
x2html :: X\ (Unit\ String\ String)\ (Mode\ Double) \rightarrow (Double, Double) \rightarrow String\\
x2html\ (XLeaf\ (Image\ i))\ (w, h) = img\ w\ h\ i\\
x2html\ (XLeaf\ (Text\ txt))\ \_ = txt\\
x2html\ (Node\ (Vt\ i)\ x1\ x2)\ (w, h) = htab\ w\ h\ (\\
\quad tr\ (td\ w\ (h * i)\ (x2html\ x1\ (w, h * i))) +\!\!+\\
\quad tr\ (td\ w\ (h * (1 - i))\ (x2html\ x2\ (w, h * (1 - i))))\\
\quad)\\
x2html\ (Node\ (Hl\ i)\ x1\ x2)\ (w, h) = htab\ w\ h\ (\\
\quad tr\ (td\ (w * i)\ h\ (x2html\ x1\ (w * i, h)) +\!\!+\\
\qquad td\ (w * (1 - i))\ h\ (x2html\ x2\ (w * (1 - i), h)))
\end{array}
$$

[8] API (="Application Program Interface").

```
  )
x2html (Node (Vb i) x1 x2) m = x2html (Node (Vt (1 - i)) x1 x2) m
x2html (Node (Hr i) x1 x2) m = x2html (Node (Hl (1 - i)) x1 x2) m
```

Funções auxiliares:

```
twoVtImg a b = Node (Vt 0.5) (XLeaf (Image a)) (XLeaf (Image b))
fourInArow a b c d =
  Node (Hl 0.5)
    (Node (Hl 0.5) (XLeaf (Text a)) (XLeaf (Text b)))
    (Node (Hl 0.5) (XLeaf (Text c)) (XLeaf (Text d)))
```

HTML:

```
htmlwrap = html · hd · (title "CP/2122 - sheet2html") · body · divt
html = tag "html" [] · ("<meta charset=\"utf-8\" />"++)
title t = (tag "title" [] t++)
body = tag "body" ["BGCOLOR" ↦ show "#F4EFD8"]
hd = tag "head" []
htab w h = tag "table" [
  "width" ↦ show2 w, "height" ↦ show2 h,
  "cellpadding" ↦ show2 0, "border" ↦ show "1px"]
tr = tag "tr" []
td w h = tag "td" ["width" ↦ show2 w, "height" ↦ show2 h]
divt = tag "div" ["align" ↦ show "center"]
img w h i = tag "img" ["width" ↦ show2 w, "src" ↦ show i] ""
tag t l x = "<" ++ t ++ " " ++ ps ++ ">" ++ x ++ "</" ++ t ++ ">\n"
  where ps = unwords [concat [t, "=", v] | (t, v) ← l]
a ↦ b = (a, b)
show2 :: Show a ⇒ a → String
show2 = show · show
```

Exemplo para teste:

```
example :: (Fractional i) ⇒ Sheet String String i
example =
  Rect (Frame 650 450)
    (Node (Vt 0.01)
      (Node (Hl 0.15)
        (XLeaf (Image "cp2122t_media/publico.jpg"))
        (fourInArow "Jornal Público" "Domingo, 5 de Dezembro 2021" "Simulação para efe
      (Node (Vt 0.55)
        (Node (Hl 0.55)
          (Node (Vt 0.1)
            (XLeaf (Text
            "Universidade do Algarve estuda planta capaz de eliminar a doença do sol
            (XLeaf (Text
              "Organismo (semelhante a um fungo) ataca de forma galopante os montado
          (XLeaf (Image
              "cp2122t_media/1647472.jpg")))
        (Node (Hl 0.25)
          (twoVtImg
              "cp2122t_media/1647981.jpg"
              "cp2122t_media/1647982.jpg")
          (Node (Vt 0.1)
              (XLeaf (Text "Manchester United vence na estreia de Rangnick"))
              (XLeaf (Text "O Manchester United venceu, este domingo, em Old Trafford,
```

Problema 4

Exemplos de mapas:

$$
\begin{aligned}
map_1 &= [[\mathit{Free}, \mathit{Blocked}, \mathit{Free}], [\mathit{Free}, \mathit{Blocked}, \mathit{Free}], [\mathit{Free}, \mathit{Free}, \mathit{Free}]] \\
map_2 &= [[\mathit{Free}, \mathit{Blocked}, \mathit{Free}], [\mathit{Free}, \mathit{Free}, \mathit{Free}], [\mathit{Free}, \mathit{Blocked}, \mathit{Free}]] \\
map_3 &= [[\mathit{Free}, \mathit{Free}, \mathit{Free}], [\mathit{Free}, \mathit{Blocked}, \mathit{Free}], [\mathit{Free}, \mathit{Blocked}, \mathit{Free}]]
\end{aligned}
$$

Código para impressões de mapas e caminhos:

$$
\begin{aligned}
&showM :: Map \to String \\
&showM = unlines \cdot (\mathsf{map}\ \ showL) \cdot reverse \\
&showL :: [Cell] \to String \\
&showL = (\![f_1, f_2]\!)\ \mathbf{where} \\
&\quad f_1 = \underline{\texttt{""}} \\
&\quad f_2 = \widehat{(+\!\!+)} \cdot (fromCell \times id) \\
&fromCell\ Lft = \texttt{"\ \ >\ \ "} \\
&fromCell\ Rght = \texttt{"\ \ <\ \ "} \\
&fromCell\ Up = \texttt{"\ \ \^{}\ \ "} \\
&fromCell\ Down = \texttt{"\ \ v\ \ "} \\
&fromCell\ Free = \texttt{"\ \ \_\ \ "} \\
&fromCell\ Blocked = \texttt{"\ \ X\ \ "} \\
&toCell\ (x, y)\ (w, z)\ |\ x < w = Lft \\
&toCell\ (x, y)\ (w, z)\ |\ x > w = Rght \\
&toCell\ (x, y)\ (w, z)\ |\ y < z = Up \\
&toCell\ (x, y)\ (w, z)\ |\ y > z = Down
\end{aligned}
$$

Código para validação de mapas (útil, por exemplo, para testes QuickCheck):

$$
\begin{aligned}
&ncols :: Map \to Int \\
&ncols = [\underline{0}, length \cdot \pi_1] \cdot outList \\
&nlines :: Map \to Int \\
&nlines = length \\
&isValidMap :: Map \to Bool \\
&isValidMap = \widehat{(\wedge)} \cdot \langle isSquare, sameLength \rangle\ \mathbf{where} \\
&\quad isSquare = \widehat{(\equiv)} \cdot \langle nlines, ncols \rangle \\
&\quad sameLength\ [\,] = True \\
&\quad sameLength\ [x] = True \\
&\quad sameLength\ (x1 : x2 : y) = length\ x1 \equiv length\ x2 \wedge sameLength\ (x2 : y)
\end{aligned}
$$

Código para geração aleatória de mapas e automatização de testes (envolve o mónade IO):

$$
\begin{aligned}
&randomRIOL :: (Random\ a) \Rightarrow (a, a) \to Int \to \mathsf{IO}\ [a] \\
&randomRIOL\ x = (\![f_1, f_2]\!)\ \mathbf{where} \\
&\quad f_1 = \underline{return\ [\,]} \\
&\quad f_2\ l = \mathbf{do}\ r1 \leftarrow randomRIO\ x \\
&\qquad r2 \leftarrow l \\
&\qquad return\ \$\ r1 : r2 \\
&buildMat :: Int \to Int \to \mathsf{IO}\ [[Int]] \\
&buildMat\ n = (\![f_1, f_2]\!)\ \mathbf{where} \\
&\quad f_1 = \underline{return\ [\,]} \\
&\quad f_2\ l = \mathbf{do}\ x \leftarrow randomRIOL\ (0 :: Int, 3 :: Int)\ n \\
&\qquad y \leftarrow l \\
&\qquad return\ \$\ x : y \\
&testWithRndMap :: \mathsf{IO}\ () \\
&testWithRndMap = \mathbf{do} \\
&\quad dim \leftarrow randomRIO\ (2, 10) :: \mathsf{IO}\ Int \\
&\quad out \leftarrow buildMat\ dim\ dim
\end{aligned}
$$

```
  map ← return $ map (map table) out
  putStr $ showM map
  putStrLn $ "Map of dimension " ++ (show dim) ++ "x" ++ (show dim) ++ "."
  putStr "Please provide a target position (must be different from (0,0)): "
  t ← readLn :: IO (Int, Int)
  putStr "Please provide the number of steps to compute: "
  n ← readLn :: IO Int
  let paths = hasTarget t (scout map (0, 0) t n) in
    if length paths ≡ 0
    then putStrLn "No paths found."
    else putStrLn $ "There are at least " ++ (show $ length paths) ++
    " possible paths. Here is one case: \n" ++ (showM $ markMap (head paths) map )
table 0 = Free
table 1 = Free
table 2 = Free
table 3 = Blocked
hasTarget y = filter (λl → elem y l)
```

Funções auxiliares $subst :: a \to Int \to [a] \to [a]$, que dado um valor x e um inteiro n, produz uma função $f : [a] \to [a]$ que dada uma lista l substitui o valor na posição n dessa lista pelo valor x:

```
subst :: a → Int → [a] → [a]
subst x = (|[f1, f2]|) where
  f1 = λl → x : tail l
  f2 f (h : t) = h : f t
```

$checkAround :: Map \to Pos \to [Pos]$, que verifica se as células adjacentes estão livres:

```
type Pos = (Int, Int)
checkAround :: Map → Pos → [Pos]
checkAround m p = concat $ map (λf → f m p)
  [checkLeft, checkRight, checkUp, checkDown]
checkLeft :: Map → Pos → [Pos]
checkLeft m (x, y) = if x ≡ 0 ∨ (m !! y) !! (x − 1) ≡ Blocked
  then [] else [(x − 1, y)]
checkRight :: Map → Pos → [Pos]
checkRight m (x, y) = if x ≡ (ncols m − 1) ∨ (m !! y) !! (x + 1) ≡ Blocked
  then [] else [(x + 1, y)]
checkUp :: Map → Pos → [Pos]
checkUp m (x, y) = if y ≡ (nlines m − 1) ∨ (m !! (y + 1)) !! x ≡ Blocked
  then [] else [(x, y + 1)]
checkDown :: Map → Pos → [Pos]
checkDown m (x, y) = if y ≡ 0 ∨ (m !! (y − 1)) !! x ≡ Blocked
  then [] else [(x, y − 1)]
```

QuickCheck

Lógicas:

```
infixr 0 ⇒
(⇒) :: (Testable prop) ⇒ (a → Bool) → (a → prop) → a → Property
p ⇒ f = λa → p a ⇒ f a
infixr 0 ⇔
(⇔) :: (a → Bool) → (a → Bool) → a → Property
p ⇔ f = λa → (p a ⇒ property (f a)) .&&. (f a ⇒ property (p a))
infixr 4 ≡
```

$(\equiv) :: Eq\ b \Rightarrow (a \rightarrow b) \rightarrow (a \rightarrow b) \rightarrow (a \rightarrow Bool)$
$f \equiv g = \lambda a \rightarrow f\ a \equiv g\ a$

infixr $4 \leqslant$
$(\leqslant) :: Ord\ b \Rightarrow (a \rightarrow b) \rightarrow (a \rightarrow b) \rightarrow (a \rightarrow Bool)$
$f \leqslant g = \lambda a \rightarrow f\ a \leqslant g\ a$

infixr $4 \wedge$
$(\wedge) :: (a \rightarrow Bool) \rightarrow (a \rightarrow Bool) \rightarrow (a \rightarrow Bool)$
$f \wedge g = \lambda a \rightarrow ((f\ a) \wedge (g\ a))$

instance *Arbitrary Cell* **where**
-- 1/4 chance of generating a cell 'Block'.
$arbitrary =$ **do** $x \leftarrow chooseInt\ (0, 3)$
$return\ \$\ f\ x$ **where**
$f\ x =$ **if** $x < 3$ **then** *Free* **else** *Blocked*

C Soluções dos alunos

Os alunos devem colocar neste anexo as suas soluções para os exercícios propostos, de acordo com o "layout"que se fornece. Não podem ser alterados os nomes ou tipos das funções dadas, mas pode ser adicionado texto, diagramas e/ou outras funções auxiliares que sejam necessárias.

Valoriza-se a escrita de *pouco* código que corresponda a soluções simples e elegantes.

Problema 1

Alínea 1 - Gerar LTree a partir da lista de transações.

Para a resolução desta alínea, foi-nos pedido que utilizássemos um tipo de dado correspondente a uma lista não vazia (NEList). Para isso começamos por calcular, a partir do inNEList, o isomorfismo outNEList, assim como as restantes funções de catamorfismo, anamorfismo... etc.

Cálculo do outNEList:

$$
\begin{array}{ll}
& outNEList \cdot inNEList = id \\
\equiv & \quad \{ \text{ Def inNEList = [singl,cons] } \} \\
& outNEList \cdot [singl, cons] = id \\
\equiv & \quad \{ \text{ Fusão-+ (20) } \} \\
& [outNEList \cdot singl, outNEList \cdot cons] = id \\
\equiv & \quad \{ \text{ Universal-+ (27) } \} \\
& \left\{ \begin{array}{l} id \cdot i_1 = outNEList \cdot singl \\ id \cdot i_2 = outNEList \cdot cons \end{array} \right. \\
\equiv & \quad \{ \text{ Natural-id (1), Troca de membros } \} \\
& \left\{ \begin{array}{l} outNEList \cdot singl = i_1 \\ outNEList \cdot cons = i_2 \end{array} \right. \\
\equiv & \quad \{ \text{ Igualdade Extensional (71) } \} \\
& \left\{ \begin{array}{l} outNEList \cdot singl\ a = i_1\ a \\ outNEList \cdot cons\ (a, b) = i_2\ (a, b) \end{array} \right. \\
\equiv & \quad \{ \text{ Def singl, Def cons } \} \\
& \left\{ \begin{array}{l} outNEList\ [a] = i_1\ a \\ outNEList\ (a : b) = i_2\ (a, b) \end{array} \right.
\end{array}
$$

Listas vazias:

$$
\begin{array}{l}
outNEList\ [a] = i_1\ a \\
outNEList\ (h : t) = i_2\ (h, t) \\
baseNEList\ f\ g = id + f \times g \\
recNEList\ \ f\ = id + id \times f \\
cataNEList\ g = g \cdot recNEList\ (cataNEList\ g) \cdot outNEList \\
anaNEList\ g\ = inNEList \cdot recNEList\ (anaNEList\ g) \cdot g \\
hyloNEList\ h\ g = cataNEList\ h \cdot anaNEList\ g
\end{array}
$$

Depois de calculada estas funções, começamos por desenvolver o diagrama do anamorfismo que constrói LTrees a partir de listas não vazias. Depois do diagrama concluído e a tipagem dos dados verificada, calculamos o gene do anamorfismo.

Diagrama do anamorfismo a resolver

$$\begin{tikzcd}
\mathit{LTree}\ A & & & A + (\mathit{LTree}\ A \times \mathit{LTree}\ A) \arrow[lll, "inLTree"'] \\
A^* \arrow[u, "list2LTree"] \arrow[r, "outNEList"] \arrow[rrr, bend right, "g_list2LTree"'] & A + (A \times A^*) \arrow[rr, "id+divLista \cdot cons"] & & A + (A^* \times A^*) \arrow[u, "id+list2LTree^2"']
\end{tikzcd}$$

Como podemos ver no diagrama, o gene será uma composição de duas funções: a função outNEList já calculada, e uma função auxiliar intermédia. Esta função intermédia é definida como uma alternativa onde, no caso desta receber uma lista singular (etiqueta 1/esquerda), não efetua modificações nesta. Caso receba um par constituído pela cabeça e a cauda da lista (etiqueta 2/direita), começa por construir a lista original, passando esta para a função divLista, que irá dividir a lista em duas partes iguais, gerando um par de listas.

Gene do anamorfismo:

$$g\_list2LTree = (id + divLista \cdot cons) \cdot outNEList$$
$$divLista\ l = ((take\ ((length\ l)\ `div`\ 2)\ l), (drop\ ((length\ l)\ `div`\ 2)\ l))$$

Alínea 2 - Gerar a MerkleTree a partir da LTree.

Para a resolução desta alínea, repetimos o mesmo processo, desenvolvendo um diagrama do catamorfismo a ser resolvido, tendo em atenção os vários tipos de dados gerados por cada função.

Diagrama do catamorfismo a resolver:

$$\begin{tikzcd}
\mathit{LTree}\ A \arrow[d, "lTree2MTree"'] & A + (\mathit{LTree}\ A)^2 \arrow[l, "\mathsf{in}=[Leaf\,,Fork]"'] \arrow[d, "id+lTree2MTree^2"] \\
\mathit{FTree}\ Z\ (Z \times A) & A + (\mathit{FTree}\ Z\ (Z \times A))^2 \arrow[l, "g_lTree2MTree"]
\end{tikzcd}$$

Para o calculo deste gene, utilizamos uma alternativa de funções onde, no caso desta apenas receber uma transação (etiqueta 1/esquerda), apenas efetuará o hashing desta, assim como a criação de um MerkleTree singular (Unit). Caso receba um par constituído por duas MerkleTrees, esta cria um par, onde o primeiro elemento será a concatenação das hashes destas, e o segundo elemento será constituído pelas suas sub-árvores.

Gene do catamorfismo:

$$g\_lTree2MTree :: Hashable\ c \Rightarrow c + (\mathit{FTree}\ \mathbb{Z}\ (\mathbb{Z}, c), \mathit{FTree}\ \mathbb{Z}\ (\mathbb{Z}, c)) \rightarrow \mathit{FTree}\ \mathbb{Z}\ (\mathbb{Z}, c)$$
$$g\_lTree2MTree = [g1\,, g2]$$
$$\quad \textbf{where}\ g1 = Unit \cdot \langle Main.hash, id \rangle$$
$$\qquad g2 = \widehat{Comp} \cdot \langle Main.concHash \cdot ((firsts \cdot \mathsf{out}) \times (firsts \cdot \mathsf{out})), id \rangle$$

Alínea 3 - Obter a raíz da MerkleTree.

Para esta alínea, é apenas preciso retirar a hash presente na raiz da MerkleTree, para isso, utilizamos a função pré-definida firsts, que retira a Hash presente na MerkleTree.

Gene de $mroot$ ("get Merkle root"):

$$g\_mroot = firsts$$

Alínea 4 - Valorização. (por fazer)

$$pairsList :: [a] \to [(a, a)]$$
$$pairsList = [\!(g\_pairsList)\!]$$
$$g\_pairsList = \bot$$
$$classicMerkleTree :: Hashable\ a \Rightarrow [a] \to FTree\ \mathbb{Z}\ \mathbb{Z}$$
$$classicMerkleTree = (hyloNEList\ conquer\ divide) \cdot (\mathsf{map}\ \ Main.hash)$$
$$divide = \bot$$
$$conquer = [head, joinMerkleTree]\ \mathbf{where}$$
$$\quad joinMerkleTree\ (l, m) = mergeMerkleTree\ m\ (evenMerkleTreeList\ l)$$
$$\quad mergeMerkleTree = (\![h_1, h_2]\!)$$
$$\quad h_1\ c\ l = \bot$$
$$\quad h_2\ (c, (f, g))\ l = \bot$$
$$\quad evenMerkleTreeList = \bot$$

Problema 2

Para a resolução deste problema, começamos por calcular as versões *pointfree* das funções fornecidas no enunciado. Tendo em conta que o problema parte de resolver pela aplicação da lei da recursividade mútua:

$\langle f, g\rangle = (\!|\langle h, k\rangle|\!)$

Começamos por aplicar a lei de Fokkinga (52):

$$
\begin{aligned}
& wc\_w \cdot \mathsf{in} = h \cdot \mathsf{F}\ \langle wc\_w, lookahead\_sep\rangle \\
\equiv\quad & \{\ \mathsf{in} := [nil, cons]\ ;\ h := [h_1, h_2]\ ;\ \mathsf{F}\ f = id + id \times f\ \} \\
& wc\_w \cdot [nil, cons] = [h_1, h_2] \cdot (id + id \times \langle wc\_w, lookahead\_sep\rangle) \\
\equiv\quad & \{\ \text{Fusão-+ (20) ; Absorção-+ (22)}\ \} \\
& [wc\_w \cdot nil, wc\_w \cdot cons] = [h_1 \cdot id, h_2 \cdot (id \times \langle wc\_w, lookahead\_sep\rangle)] \\
\equiv\quad & \{\ \text{Eq-+ (27); Natural-id (1)}\ \} \\
& \begin{cases} wc\_w \cdot nil = h_1 \\ wc\_w \cdot cons = h_2 \cdot (id \times \langle wc\_w, lookahead\_sep\rangle) \end{cases} \\
\equiv\quad & \{\ \text{Igualdade Extensional (71) ; Def-comp (72); Def-const (74), Def-nil}\ \} \\
& \begin{cases} wc\_w\ [\,] = h_1\ a \\ wc\_w\ (cons\ a) = h_2\ ((id \times \langle wc\_w, lookahead\_sep\rangle)\ a) \end{cases} \\
\equiv\quad & \{\ wc\_w\ [\,] = 0\ ;\ \text{a := (a,b) ; Def-x (77)}\ \} \\
& \begin{cases} 0 = h_1\ a \\ wc\_w\ (cons\ (a, b)) = h_2\ (id\ a, \langle wc\_w, lookahead\_sep\rangle\ b) \end{cases} \\
\equiv\quad & \{\ \text{Def-id (73); Def-split (76); Def-cons}\ \} \\
& \begin{cases} 0 = h_1\ a \\ wc\_w\ (a : b) = h_2\ (a, (wc\_w\ b, lookahead\_sep\ b)) \end{cases} \\
\equiv\quad & \{\ \text{Propriedade reflexiva da igualdade ; } wc\_w\ (a : b) = \mathbf{if}\ \neg\ (sp\ a) \wedge lookahead\_sep\ b\ \mathbf{then}\ wc\_w\ b + 1\ \mathbf{else}\ wc \\
& \begin{cases} h_1\ a = 0 \\ h_2\ (a, (wc\_w\ b, lookahead\_sep\ b)) = \mathbf{if}\ \neg\ (sp\ a) \wedge lookahead\_sep\ b\ \mathbf{then}\ wc\_w\ b + 1\ \mathbf{else}\ wc\_w\ b \end{cases} \\
\equiv\quad & \{\ \text{Def-cond (78)}\ \} \\
& \begin{cases} h_1\ a = 0 \\ h_2\ (a, (wc\_w\ b, lookahead\_sep\ b)) = cond\ (\neg\ (sp\ a) \wedge lookahead\_sep\ b)\ (wc\_w\ b + 1)\ (wc\_w\ b) \end{cases} \\
\equiv\quad & \{\ \text{Def succ; } andOperator\ ;\ andOperator := and\ ;\ lookahead\_sep := la\ ;\ wc\_c := wc\ ;\ \text{(Abreviaturas)}\ \} \\
& \begin{cases} h_1\ a = 0 \\ h_2\ (a, (wc\ b, la\ b)) = cond\ (and\ (\neg\ (sp\ a), la\ b))\ (\mathsf{succ}\ \ (wc\ b))\ (wc\ b) \end{cases} \\
\equiv\quad & \{\ \text{Def-comp (72)}\ \} \\
& \begin{cases} h_1\ a = 0 \\ h_2\ (a, (wc\ b, la\ b)) = cond\ (and\ ((\neg \cdot sp)\ a, la\ b))\ (\mathsf{succ}\ \ (wc\ a))\ (wc\ b) \end{cases} \\
\equiv\quad & \{\ \text{Def-proj (79)}\ \} \\
& \begin{cases} h_1\ a = 0 \\ h_2\ (a, (wc\ b, la\ b)) = cond\ (and\ ((\neg \cdot sp)\ x, \pi_2\ (wc\ y, la\ y)))\ (\mathsf{succ}\ \ (\pi_1\ (wc\ y, la\ y)))\ (\pi_1\ (wc\ y, la\ y)) \end{cases} \\
\equiv\quad & \{\ \text{Def-x (77), Def-comp (72)}\ \} \\
& \begin{cases} h_1\ a = 0 \\ h_2\ (a, (wc\ b, la\ b)) = cond\ ((and \cdot ((\neg \cdot sp) \times \pi_2))\ (a, (wc\ b, la\ b)))\ ((\mathsf{succ}\ \cdot \pi_1)\ (wc\ b, la\ b))\ (\pi_1\ (wc\ b, l \end{cases}
\end{aligned}
$$

$$\equiv \qquad \{ \text{ Def-x (79), Def-comp (72) } \}$$

$$\begin{cases} h_1 \ a = 0 \\ h_2 \ (a, (wc \ b, la \ b)) = cond \ ((and \cdot ((\neg \cdot sp) \times \pi_2)) \ (a, (wc \ b, la \ b))) \ ((\mathsf{succ} \ \cdot \pi_1 \cdot \pi_2) \ (a, (wc \ b, la \ b))) \ (\pi_1 \end{cases}$$

$$\equiv \qquad \{ \text{ Def-cond (78) } \}$$

$$\begin{cases} h_1 \ a = 0 \\ h_2 \ (a, (wc \ b, la \ b)) = (cond \ (and \cdot ((\neg \cdot sp) \times \pi_2)) \ (\mathsf{succ} \ \cdot \pi_1 \cdot \pi_2) \ (\pi_1 \cdot \pi_2)) \ (a, (wc \ b, la \ b)) \end{cases}$$

Do último passo que determinou a versão *pointwise* deduzimos que: que h_1 é $zero$ e h_2 é a função $cond \ (andOperator \cdot ((\neg \cdot sp) \times \pi_2)) \ (\mathsf{succ} \ \cdot \pi_1 \cdot \pi_2) \ (\pi_1 \cdot \pi_2)$.

Vejamos agora como podemos calcular k, começando novamente pela lei de Fokkinga (52):

$$lookahead\_sep \cdot \mathsf{in} = k \cdot \mathsf{F} \ \langle wc\_w, lookahead\_sep \rangle \tag{18}$$

$$\equiv \qquad \{ \ \mathsf{in} := [nil, cons] \ ; \ k := [k_1, k_2] \ ; \ \mathsf{F} \ f = id + id \times f \ \}$$

$$lookahead\_sep \cdot [nil, cons] = [k_1, k_2] \cdot (id + id \times \langle wc\_w, lookahead\_sep \rangle) \tag{19}$$

$$\equiv \qquad \{ \text{ Fusão-+ (20) ; Absorção-+ (22) } \}$$

$$[lookahead\_sep \cdot nil, lookahead\_sep \cdot cons] = [k_1 \cdot id, k_2 \cdot (id \times \langle wc\_w, lookahead\_sep \rangle)] \tag{20}$$

$$\equiv \qquad \{ \text{ Eq-+ (27); Natural-id (1) } \}$$

$$\begin{cases} lookahead\_sep \cdot nil = k_1 \\ lookahead\_sep \cdot cons = k_2 \cdot (id \times \langle wc\_w, lookahead\_sep \rangle) \end{cases} \tag{21}$$

$$\equiv \qquad \{ \text{ Igualdade Extensional (71); Def-comp (72); Def-const (74), Def nil } \}$$

$$\begin{cases} lookahead\_sep \ [\,] = k_1 \ a \\ lookahead\_sep \ (cons \ a) = k_2 \ ((id \times \langle wc\_w, lookahead\_sep \rangle) \ a) \end{cases} \tag{22}$$

$$\equiv \qquad \{ \ lookahead\_sep \ [\,] = True \ ; \text{ a := (a,b) ; Def-x (77) } \}$$

$$\begin{cases} True = k_1 \ a \\ lookahead\_sep \ (cons \ (a, b)) = h_2 \ (id \ a, \langle wc\_w, lookahead\_sep \rangle \ b) \end{cases} \tag{23}$$

$$\equiv \qquad \{ \text{ Def-id (73); Def-split (76) ; Def cons } \}$$

$$\begin{cases} True = k_1 \ a \\ lookahead\_sep \ (a : b) = k_2 \ (a, (wc\_w \ b, lookahead\_sep \ b)) \end{cases} \tag{24}$$

$$\equiv \qquad \{ \ lookahead\_sep \ (aby) = sp \ a \ \}$$

$$\begin{cases} True = k_1 \ a \\ sp \ x = k_2 \ (a, (wc\_w \ b, lookahead\_sep \ b)) \end{cases} \tag{25}$$

$$\equiv \qquad \{ \text{ Propriedade reflexiva da igualdade } \}$$

$$\begin{cases} k_1 \ a = True \\ k_2 \ (a, (wc\_w \ b, lookahead\_sep \ b)) = sp \ a \end{cases} \tag{26}$$

Assim, chegamos à conclusão que k_1 é sempre $\underline{True}$ e k_2 é dado pela função $sp \cdot \pi_1$.

Utilizando a Lei da Troca, podemos obter a definição final do gene da função worker.

$$
\begin{aligned}
& \llparenthesis \langle [h_1, h_2], [k_1, k_2] \rangle \rrparenthesis \\
\equiv \quad & \quad \{ \text{ Lei da troca (28) } \} \\
& \llparenthesis [\langle h_1, k_1 \rangle, \langle h_2, k_2 \rangle] \rrparenthesis
\end{aligned}
$$

Para o wrapper, apenas precisamos de retirar o primeiro elemento do par gerado pelo worker, pois contém o número de todas as palavras lidas.

$$
\begin{aligned}
& \mathit{wc\_w\_final} :: [\,\mathit{Char}\,] \to \mathit{Int} \\
& \mathit{wc\_w\_final} = \mathit{wrapper} \cdot \mathit{worker} \\
& \mathit{worker} = \llparenthesis [\mathit{g1}, \mathit{g2}] \rrparenthesis \\
& \mathit{wrapper} = \mathit{fromInteger} \cdot \pi_1
\end{aligned}
$$

Gene de worker:

$$
\begin{aligned}
& \mathit{g1} = \langle h_1, k_1 \rangle \\
& \mathit{g2} = \langle h_2, k_2 \rangle
\end{aligned}
$$

Genes $h = [h_1, h_2]$ e $k = [k_1, k_2]$ identificados no cálculo:

$$
\begin{aligned}
& h_1 = \mathit{zero} \\
& h_2 = \mathit{cond}\ (\mathit{andOperator} \cdot ((\neg \cdot \mathit{sp}) \times \pi_2))\ (\mathsf{succ}\ \cdot \pi_1 \cdot \pi_2)\ (\pi_1 \cdot \pi_2) \\
& k_1 = \underline{\mathit{True}} \\
& k_2 = \mathit{sp} \cdot \pi_1 \\
& \mathit{andOperator}\ (x, y) = x \wedge y
\end{aligned}
$$

Como podemos ver, o gene da função worker, ao receber uma lista vazia, apenas gere uma par (0,True), indicando que não leu nenhuma palavra e levanta a flag de separador, para indicar que próximos caracteres irão pertencer a uma nova palavra. No caso do gene receber um par composto por um caracter e o par (Integer,Bool) anterior, este irá avaliar o caracter a ser lido, assim como a flag presente, incrementando assim ou não o contador de palavras, gerando também um Bool dependendo do caracter lido.

Apresentamos aqui o diagrama do catamorfismo do worker da função:

$$
\begin{tikzcd}
\mathit{Char}^* \arrow[d, "\mathit{worker}"'] & 1 + (\mathit{Char} \times \mathit{Char}^*) \arrow[l, "\mathbf{in} = [\mathit{nil}, \mathit{cons}]"'] \arrow[d, "\mathit{recList}\ (\mathit{worker})"] \\
(\mathit{Integer}, \mathit{Bool}) & 1 + (\mathit{Char} \times (\mathit{Integer}, \mathit{Bool})) \arrow[l, "{[\mathit{g1}, \mathit{g2}]}"']
\end{tikzcd}
$$

Problema 3

$$
\begin{array}{l}
inX :: u + (i, (X\ \ u\ i, X\ \ u\ i)) \rightarrow X\ \ u\ i \\
inX = [XLeaf, \widehat{Node} \cdot assocl] \\
outX\ (XLeaf\ \ u) = i_1\ u \\
outX\ (Node\ \ i\ l\ r) = i_2\ (i, (l, r)) \\
baseX\ f\ h\ g = f + (h \times (g \times g)) \\
recX\ f = baseX\ \ id\ \ id\ f \\
cataX\ \ g = g \cdot (recX\ \ (cataX\ \ g)) \cdot outX
\end{array}
$$

Inserir a partir daqui o resto da resolução deste problema:
....

Problema 4

Alínea 1 - Função pairL

Para o cálculo desta função, começamos por desenvolver o diagrama do anamorfismo, utilizando os vários construtores de Listas.

$$\begin{tikzcd}
(A,A)^* & & 1+((A,A)\times(A,A)^*) \arrow[ll, "inList"'] \\
A^* \arrow[u, "pairL"] \arrow[r, "outList"] \arrow[rr, bend right, "g"'] & 1+(A\times A^*) \arrow[r, "id+pairHandler"] & 1+((A,A)\times A^*) \arrow[u, "recList\ (pairL)"']
\end{tikzcd}$$

Para o gene do anamorfismo, começamos por criar uma composição das funções $outList$ e a função que será responsável pela criação dos pares. Definimos esta função como uma alternativa de um id com uma subfunção chamada $pairHandler$. Esta função começa por avaliar se a lista fornecida era singular, pois assim gere apenas um par repetido ($singlePair$). Caso a lista fornecida contenha múltiplos elementos ($multiplePairs$), este vai gerando os pares com a cabeça da lista e a cabeça da cauda. Por fim, avalia se a cauda é uma lista singular, modificando-a para uma lista vazia para evitar a repetição do último elemento.

Tivemos também a necessidade de definir duas funções auxiliares $emptyList$ e $singlList$. Estas apenas avaliam uma dada lista, indicando se esta está vazia ou se contém apenas um elemento, respetivamente.

$$pairL :: [\,a\,] \rightarrow [\,(a,a)\,]$$
$$pairL = [\!(g)\!] \ \textbf{where}$$
$$\quad g = (id + pairHandler) \cdot outList$$
$$\quad pairHandler = cond\ (emptyList \cdot \pi_2)\ singlePair\ multiplePairs$$
$$\quad singlePair = \langle\langle \pi_1, \pi_1\rangle, nil\rangle$$
$$\quad multiplePairs = \langle\langle \pi_1, head \cdot \pi_2\rangle, cond\ (singlList \cdot \pi_2)\ nil\ \pi_2\rangle$$
$$emptyList\ [\,] = True$$
$$emptyList\ \_ = False$$
$$singlList\ [\,x\,] = True$$
$$singlList\ \_ = False$$

Alínea 2 - Função markMap

Para esta alínea, apercebemo-nos que a função objetivo se tratava de um hilomorfismo. Como tal, conseguimos distinguir as duas etapas constituintes da mesma: $divide\ and\ conquer$. A técnica de $divide$, neste caso em específico, refere-se à construção (anamorfismo) de pares de posições a partir de uma lista de posições iniciais. Na segunda parte ($conquer$), é necessário consumir (catamorifismo) as mesmas posições, obtendo o mapa com todas as movimentações.

Como a primeira parte deste hilomorfismo já foi desenvolvida na alínea anterior ($pairL$), começamos por desenvolver a etapa de $conquer$, criando o diagrama do catamorfismo, no entanto, encontramos problemas com a tipagem devido à receção do argumento Map, ficando sem saber como este era introduzido no diagrama.

Decidimos por isso começar por desenvolver uma função que, ao receber um par de posições e um mapa, efetuava a modificação correta neste ($updateCell$). Para esta função foi necessário desenvolver as funções auxiliares $updateElemList$ e $getAtIndex$, sendo que a primeira atualiza um elemento numa dada lista, numa posição, e a segunda apenas retorna o elemento num dado index.

Com esta função, conseguimos efetuar modificações no mapa, no entanto, não tivemos sucesso ao implementa-la no catamorfismo visto que não conseguimos efetuar as modificações utilizando o mapa que é originado da aplicação do functor recursivo, devido à tipagem deste ser ($Map \rightarrow Map$) quando nós apenas esperavamos Map.

Na versão final apresentamos a seguinte definição do gene, esta está incompleta e apenas efetua a modificação do primeiro "passo"efetuado, dado que não conseguimos utilizar o mapa "modificado"até aquele ponto. Apresentamos também uma função $updateMap$ que faz todas modificações, obtendo o mapa pretendido, no entanto, não utiliza nenhum catamorfismo. Acreditamos que no futuro, com uma melhor análise da tipagem dos dados, se possa solucionar facilmente o problema em questão.

$$
\begin{tikzcd}
Pos^* \arrow[r, "g={[id,pairHandler]\cdot outList}"] \arrow[d, "pairL"] \arrow[dd, bend right=60, "markMap\ m"'] & 1+((Pos,Pos)\times Pos^*) \arrow[d, "recList\ (pairL)"] \\
(Pos\times Pos)^* \arrow[r, "outList"] \arrow[d, "(\![\underline{id},f_2]\!)\ m"] & 1+((Pos\times Pos)\times(Pos\times Pos)^*) \arrow[d, "recList\ (\![\underline{id},f_2]\!)\ m"] \\
Map & 1+((Pos\times Pos)\times Map) \arrow[l, "{[\underline{id},f_2]}"']
\end{tikzcd}
$$

$$
\begin{aligned}
&\quad markMap :: [Pos] \to Map \to Map\\
&markMap\ l = (\![\underline{id}, f_2]\!)\ (pairL\ l)\ \textbf{where}\\
&\quad f_2 = updateCell \cdot \pi_1
\end{aligned}
$$

$$
\begin{aligned}
&updateCell :: (Pos, Pos) \to Map \to Map\\
&updateCell\ (po1@(x1, y1), po2)\ m = updateElemList\ y1\ (updateElemList\ x1\ (toCell\ po1\ po2)\ (getAtIndex\ y1\ m
\end{aligned}
$$

$$
\begin{aligned}
&updateElemList :: Int \to a \to [a] \to [a]\\
&updateElemList\ n\ x\ l = \widehat{(+\!\!+)}\ (((id \times \overline{cons}\ x \cdot drop\ 1) \cdot (splitAt\ n))\ l)
\end{aligned}
$$

$$
\begin{aligned}
&getAtIndex :: Int \to [a] \to a\\
&getAtIndex\ 0\ (h : t) = h\\
&getAtIndex\ n\ (h : t) = getAtIndex\ (n - 1)\ t
\end{aligned}
$$

Função para atualização do mapa (Sem utilização do catamorfismo):

$$
\begin{aligned}
&updateMap :: [(Pos, Pos)] \to Map \to Map\\
&updateMap\ [\,]\ m = m\\
&updateMap\ (h : t)\ m = updateMap\ t\ (updateCell\ h\ m)
\end{aligned}
$$

Alínea 3 - Função scout

$$
\begin{aligned}
&scout :: Map \to Pos \to Pos \to Int \to [[Pos]]\\
&scout\ m\ s\ t = (\![f_1, (\gg\!\!= f_2\ m\ s)]\!)\ \textbf{where}\\
&\quad f_1 = \bot\\
&\quad f_2 = \bot
\end{aligned}
$$

Valorização (opcional) Completar as seguintes funções de teste no QuickCheck para verificação de propriedades das funções pedidas, a saber:

Propriedade [QuickCheck] 1 *A lista correspondente ao lado esquerdo dos pares em ($pairL\ l$) é a lista original l a menos do último elemento. Analogamente, a lista correspondente ao lado direito dos pares em ($pairL\ l$) é a lista original l a menos do primeiro elemento:*

$$prop\_reconst\ l = \bot$$

Propriedade [QuickCheck] 2 *Assuma que uma linha (de um mapa) é prefixa de uma outra linha. Então a representação da primeira linha também prefixa a representação da segunda linha:*

$$prop\_prefix2\ l\ l' = \bot$$

Propriedade [QuickCheck] 3 *Para qualquer linha (de um mapa), a sua representação deve conter um número de símbolos correspondentes a um tipo célula igual ao número de vezes que esse tipo de célula aparece na linha em questão.*

$prop\_nmbrs\ l\ c = \bot$
$count :: (Eq\ a) \Rightarrow a \rightarrow [a] \rightarrow Int$
$count = \bot$

Propriedade [QuickCheck] 4 *Para qualquer lista l a função $markMap\ l$ é idempotente.*

$inBounds\ m\ (x, y) = \bot$
$prop\_idemp2\ l\ m = \bot$

Propriedade [QuickCheck] 5 *Todas as posições presentes na lista dada como argumento irão fazer com que as células correspondentes no mapa deixem de ser $Free$.*

$prop\_extr2\ l\ m = \bot$

Propriedade [QuickCheck] 6 *Quanto maior for o tamanho máximo dos caminhos mais caminhos que alcançam a posição alvo iremos encontrar:*

$prop\_reach\ m\ t\ n\ n' = \bot$

Índice

Referências

[1] B.W. Kernighan and D.M. Richtie. *The C Programming Language*. Prentice Hall, Englewood Cliffs, N.J., 1978.

[2] D.E. Knuth. *Literate Programming*. CSLI Lecture Notes Number 27. Stanford University Center for the Study of Language and Information, Stanford, CA, USA, 1992.

[3] J.N. Oliveira. *Program Design by Calculation*, 2018. Draft of textbook in preparation. viii+297 pages. Informatics Department, University of Minho.

[4] SelfKey. What is a Merkle tree and how does it affect blockchain technology?, 2015. Blog: https://selfkey.org/what-is-a-merkle-tree-and-how-does-it-affect-blockchain-techno
Last read: 6 de Fevereiro de 2022.