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

Departamento de Informática
Universidade do Minho

Fevereiro de 2022

Grupo nr.	3
a90234	Filipa Rebelo
a87956	Joana Oliveira

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:

$$list2LTree :: [\,a\,] \rightarrow LTree\ a$$

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

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

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

   $$\begin{array}{l} list2LTree :: [\,a\,] \rightarrow LTree\ a \\ list2LTree = [\!(g\_list2LTree)\!] \end{array}$$

   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]

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

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

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

3. Defina g_mroot por forma a

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

   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:

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

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 $\mathit{worker}/\mathit{wrapper}$ 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:

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

O tipo Sheet (="página de jornal")

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

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

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

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

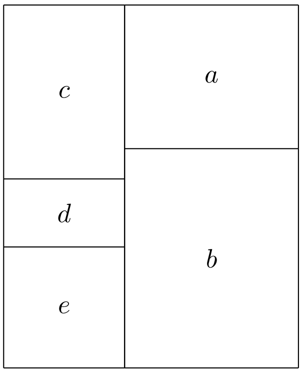
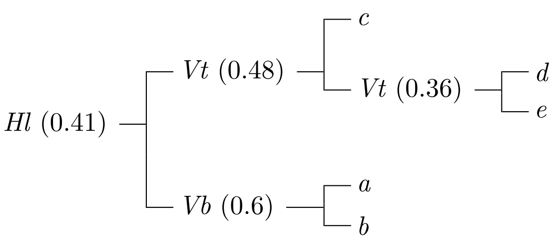


Figura 2: Layout de página de jornal.

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:

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

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 $scout :: Map \rightarrow Pos \rightarrow Pos \rightarrow Int \rightarrow [[Pos]]$, 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

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

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]

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

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

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

[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:

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

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

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]

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

Geração de HTML:

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

Funções auxiliares:

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

```
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:

$$
\begin{array}{l}
map_1 = [[Free, Blocked, Free], [Free, Blocked, Free], [Free, Free, Free]]\\
map_2 = [[Free, Blocked, Free], [Free, Free, Free], [Free, Blocked, Free]]\\
map_3 = [[Free, Free, Free], [Free, Blocked, Free], [Free, Blocked, Free]]
\end{array}
$$

Código para impressões de mapas e caminhos:

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

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

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

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

$$
\begin{array}{l}
randomRIOL :: (Random\ a) \Rightarrow (a, a) \rightarrow Int \rightarrow \mathsf{IO}\ [a]\\
randomRIOL\ x = (\![f_1, f_2]\!) \ \mathbf{where}\\
\quad f_1 = \underline{return\ [\,]}\\
\quad f_2\ l = \mathbf{do}\ r1 \leftarrow randomRIO\ x\\
\qquad r2 \leftarrow l\\
\qquad return\ \$\ r1 : r2\\
buildMat :: Int \rightarrow Int \rightarrow \mathsf{IO}\ [[Int]]\\
buildMat\ n = (\![f_1, f_2]\!) \ \mathbf{where}\\
\quad f_1 = \underline{return\ [\,]}\\
\quad f_2\ l = \mathbf{do}\ x \leftarrow randomRIOL\ (0 :: Int, 3 :: Int)\ n\\
\qquad y \leftarrow l\\
\qquad return\ \$\ x : y\\
testWithRndMap :: \mathsf{IO}\ ()\\
testWithRndMap = \mathbf{do}\\
\quad dim \leftarrow randomRIO\ (2, 10) :: \mathsf{IO}\ Int\\
\quad out \leftarrow buildMat\ dim\ dim\\
\quad \mathsf{map}\ \leftarrow return\ \$\ \mathsf{map}\ \ (\mathsf{map}\ \ table)\ out\\
\quad putStr\ \$\ showM\ \mathsf{map}\\
\quad putStrLn\ \$\ \texttt{"Map of dimension "} +\!\!+ (show\ dim) +\!\!+ \texttt{"x"} +\!\!+ (show\ dim) +\!\!+ \texttt{"."}
\end{array}
$$

$putStr$ "Please provide a target position (must be different from (0,0)): "
$t \leftarrow readLn :: \mathsf{IO}\ (Int, Int)$
$putStr$ "Please provide the number of steps to compute: "
$n \leftarrow readLn :: \mathsf{IO}\ Int$
let $paths = hasTarget\ t\ (scout\ \mathsf{map}\ (0,0)\ t\ n)$ **in**
if $length\ paths \equiv 0$
then $putStrLn$ "No paths found."
else $putStrLn\ \$$ "There are at least " $+\!\!+ (show\ \$\ length\ paths) +\!\!+$
" possible paths. Here is one case: \n" $+\!\!+ (showM\ \$\ markMap\ (head\ paths)\ \mathsf{map})$

$table\ 0 = Free$
$table\ 1 = Free$
$table\ 2 = Free$
$table\ 3 = Blocked$

$hasTarget\ y = filter\ (\lambda l \to elem\ y\ l)$

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:

$subst :: a \to Int \to [a] \to [a]$
$subst\ x = (\![f_1, f_2]\!)$ **where**
$f_1 = \underline{\lambda l \to x : tail\ l}$
$f_2\ f\ (h : t) = h : f\ t$

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

type $Pos = (Int, Int)$
$checkAround :: Map \to Pos \to [Pos]$
$checkAround\ m\ p = concat\ \$\ \mathsf{map}\ (\lambda f \to f\ m\ p)$
$[checkLeft, checkRight, checkUp, checkDown]$
$checkLeft :: Map \to Pos \to [Pos]$
$checkLeft\ m\ (x, y) =$ **if** $x \equiv 0 \lor (m\ !!\ y)\ !!\ (x - 1) \equiv Blocked$
then $[\,]$ **else** $[(x - 1, y)]$
$checkRight :: Map \to Pos \to [Pos]$
$checkRight\ m\ (x, y) =$ **if** $x \equiv (ncols\ m - 1) \lor (m\ !!\ y)\ !!\ (x + 1) \equiv Blocked$
then $[\,]$ **else** $[(x + 1, y)]$
$checkUp :: Map \to Pos \to [Pos]$
$checkUp\ m\ (x, y) =$ **if** $y \equiv (nlines\ m - 1) \lor (m\ !!\ (y + 1))\ !!\ x \equiv Blocked$
then $[\,]$ **else** $[(x, y + 1)]$
$checkDown :: Map \to Pos \to [Pos]$
$checkDown\ m\ (x, y) =$ **if** $y \equiv 0 \lor (m\ !!\ (y - 1))\ !!\ x \equiv Blocked$
then $[\,]$ **else** $[(x, y - 1)]$

QuickCheck

Lógicas:

infixr $0 \Rightarrow$
$(\Rightarrow) :: (Testable\ prop) \Rightarrow (a \to Bool) \to (a \to prop) \to a \to Property$
$p \Rightarrow f = \lambda a \to p\ a \Rightarrow f\ a$
infixr $0 \Leftrightarrow$
$(\Leftrightarrow) :: (a \to Bool) \to (a \to Bool) \to a \to Property$
$p \Leftrightarrow f = \lambda a \to (p\ a \Rightarrow property\ (f\ a))\ .\&\&.\ (f\ a \Rightarrow property\ (p\ a))$
infixr $4 \equiv$
$(\equiv) :: Eq\ b \Rightarrow (a \to b) \to (a \to b) \to (a \to Bool)$
$f \equiv g = \lambda a \to f\ a \equiv g\ a$

$$
\begin{array}{l}
\textbf{infixr } 4 \leqslant \\
(\leqslant) :: \mathit{Ord}\ b \Rightarrow (a \to b) \to (a \to b) \to (a \to \mathit{Bool}) \\
f \leqslant g = \lambda a \to f\ a \leqslant g\ a \\
\textbf{infixr } 4 \wedge \\
(\wedge) :: (a \to \mathit{Bool}) \to (a \to \mathit{Bool}) \to (a \to \mathit{Bool}) \\
f \wedge g = \lambda a \to ((f\ a) \wedge (g\ a)) \\
\textbf{instance } \mathit{Arbitrary}\ \mathit{Cell}\ \textbf{where} \\
\quad\quad \text{-- 1/4 chance of generating a cell 'Block'.} \\
\quad \mathit{arbitrary} = \textbf{do } x \leftarrow \mathit{chooseInt}\ (0,3) \\
\quad\quad \mathit{return}\ \$\ f\ x\ \textbf{where} \\
\quad\quad\quad f\ x = \textbf{if } x < 3\ \textbf{then } \mathit{Free}\ \textbf{else } \mathit{Blocked}
\end{array}
$$

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

Para chegar à definição do outNEList usamos a definição do inNEList, como é possível ver de seguida.

$$
\begin{array}{ll}
& \mathit{outNEList} \cdot \mathit{inNEList} = \mathit{id} \\
<=> & \quad \{ \text{ Definição de in } \} \\
& \mathit{outNEList} \cdot [\mathit{singl}, \mathit{cons}] = \mathit{id} \\
<=> & \quad \{ \text{ lei 20 do formulario } \} \\
& [\mathit{outNEList} \cdot \mathit{singl}, \mathit{outNEList} \cdot \mathit{cons}] = \mathit{id} \\
<=> & \quad \{ \text{ lei 17 do formulario } \} \\
& \left\{ \begin{array}{l} \mathit{id} \cdot i_1 = \mathit{outNEList} \cdot \mathit{singl} \\ \mathit{id} \cdot i_2 = \mathit{outNEList} \cdot \mathit{cons} \end{array} \right. \\
<=> & \quad \{ \text{ lei 1 do formulario } \} \\
& \left\{ \begin{array}{l} i_1 = \mathit{outNEList} \cdot \mathit{singl} \\ i_2 = \mathit{outNEList} \cdot \mathit{cons} \end{array} \right. \\
<=> & \quad \{ \text{ lei 71 e 72 do formulario } \} \\
& \left\{ \begin{array}{l} i_1\ a = \mathit{outNEList} \cdot (\mathit{singl}\ a) \\ i_2\ (h,t) = \mathit{outNEList} \cdot (\mathit{cons}\ (a,t)) \end{array} \right. \\
<=> & \quad \{ \text{ cons(a,b) = a:b ; singl a = [a] } \} \\
& \left\{ \begin{array}{l} i_1\ a = \mathit{outNEList} \cdot a \\ i_2\ (h,t) = \mathit{outNEList} \cdot (h : t) \end{array} \right.
\end{array}
$$

Listas vazias:

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

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

Gene do anamorfismo:

$$\begin{array}{l} g\_list2LTree = (id + aux) \cdot outNEList\ \mathbf{where} \\ \quad aux\ (h,t) = splitAt\ length\ (h:t) \div 2\ (h:t) \end{array}$$

Foi desenhado o diagrama do anamorfismo de modo a conseguirmos perceber melhor o nosso objetivo.

$$\begin{tikzcd} \mathit{LTree}\ A & & A + (\mathit{LTree}\ A)^2 \arrow[ll, "\mathsf{in}"'] \\ A^* \arrow[u, "\llbracket g \rrbracket"'] \arrow[rr, "g"] & & A + (A^*)^2 \arrow[u, "id + \llbracket g \rrbracket^2"'] \end{tikzcd}$$

Assim sendo, foi possível chegar à conclusão de que o gene do anamorfismo devia criar criar um tuplo com duas listas, no caso em que recebe uma lista com mais do que um argumento. Posto isto, recorremos então à função *splitAt*, que nos permitiu dividir a lista original em duas e criar o tuplo correspondente. Esta lista é sempre dividida a meio de modo a permitir que a árvore criada seja equilibrada.

Gene do catamorfismo:

$$\begin{array}{l} g\_lTree2MTree :: Hashable\ c \Rightarrow c + (\mathit{FTree}\ \mathbb{Z}\ (\mathbb{Z}, c), \mathit{FTree}\ \mathbb{Z}\ (\mathbb{Z}, c)) \rightarrow \mathit{FTree}\ \mathbb{Z}\ (\mathbb{Z}, c) \\ g\_lTree2MTree = [g1, g2]\ \mathbf{where} \\ \quad g1\ x = Unit\ (toInteger\ (Data.Hashable.hash\ x), x) \\ \quad g2\ (Unit\ (h_1, x1), Unit\ (h_2, x2)) = Comp\ (concHash\ (h_1, h_2))\ (Unit\ (h_1, x1), Unit\ (h_2, x2)) \\ \quad g2\ (Unit\ (h_1, x1), Comp\ h_2\ ts2) = Comp\ (concHash\ (h_1, h_2))\ (Unit\ (h_1, x1), Comp\ h_2\ ts2) \\ \quad g2\ (Comp\ h_1\ ts1, Unit\ (h_2, x2)) = Comp\ (concHash\ (h_1, h_2))\ (Unit\ (h_2, x2), Comp\ h_1\ ts1) \\ \quad g2\ (Comp\ h_1\ ts1, Comp\ h_2\ ts2) = Comp\ (concHash\ (h_1, h_2))\ (Comp\ h_1\ ts1, Comp\ h_2\ ts2) \end{array}$$

Tal como anteriormente baseamos também a nossa resolução num diagrama.

$$\begin{tikzcd} \mathit{LTree}\ A \arrow[d, "(\!| g |\!)"'] & & A + (\mathit{LTree}\ A)^2 \arrow[ll, "\mathsf{in}"'] \arrow[d, "id + (\!| g |\!)^2"] \\ \mathit{FTree}\ \mathbb{Z}\ (\mathbb{Z}, A) & & A + (\mathit{FTree}\ \mathbb{Z}\ (\mathbb{Z}, A))^2 \arrow[ll, "g"] \end{tikzcd}$$

A partir deste foi possível perceber que podemos receber apenas um elemento do tipo A, que será então depois transformado em *FTree*, utilizando o tipo *Unit*. E além disso, podemos também receber pares de *FTree*, que se deverão juntar e dar então como resultado uma nova *FTree*.

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

$$g\_mroot = firsts$$

Mais uma vez apoiamos a nossa resolução num diagrama de maneira a descobrirmos o gene de *mroot*.

$$\begin{tikzcd} \mathit{FTree}\ \mathbb{Z}\ (\mathbb{Z}, A) \arrow[d, "(\!| g |\!)"'] & & (\mathbb{Z}, A) + \mathbb{Z} \times (\mathit{FTree}\ \mathbb{Z}\ (\mathbb{Z}, A))^2 \arrow[ll, "\mathsf{in}"'] \arrow[d, "id + id \times (\!| g |\!)^2"] \\ \mathbb{Z} & & (\mathbb{Z}, A) + \mathbb{Z} \times \mathbb{Z}^2 \arrow[ll, "g"] \end{tikzcd}$$

Daqui, conseguimos perceber então o que o catamorfismo irá receber como alternativas. Do lado esquerdo, temos a informação que está contida na raíz da árvore, sendo esta um tuplo com o hash e a transação associada. Do lado direito, temos o resultado recursivo de aplicar a função *mroot* às sub-árvores e também o valor que se encontra na sua raíz. Assim sendo, em ambos os casos, o que iremos querer retirar irá ser sempre o que se encontra no lado esquerdo do tuplo da alternativa esquerda. Para tal, decidimos então que faria sentido definir como gene a função *firsts*, uma vez que esta nos permite, numa alternativa, retirar o elemento do lado esquerdo e, de seguida retirar o elemento do lado esquerdo que lá se encontra, obtendo-se assim a *Merkle root* da árvore resultante.

Alínea 1.4: No algoritmo de construção de uma *Merkle Tree* cada nodo contém a soma das transações dos filhos. A *mroot* permite verificar rapidamente se uma transação específica ocorreu, em um determinado bloco, com a maior precisão possível. Se algum valor da transação for alterado, o valor da *mroot* também será alterado, pois esta contém a soma de todo o bloco de transações. Assim, basta alterar-se apenas um valor para que os valores dos seus pais sejam também alterados. Conclui-se assim que funcionam como uma blockchain em que, quando se altera a cadeia, os blocos a esta ligados serão alterados.

Valorização:

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

Problema 2

$$\begin{array}{l}
wc_w_final :: [\,Char\,] \to Int \\
wc_w_final = wrapper \cdot worker \\
worker = (\!|[g1, g2]|\!) \\
wrapper = \pi_2
\end{array}$$

Gene de $worker$:

$$\begin{array}{l}
g1 = \langle \underline{True}, \underline{0} \rangle \\
g2 = \langle sep \cdot \pi_1, cond\ (\widehat{(\wedge)} \cdot ((\neg \cdot sep) \times \pi_1))\ (\mathsf{succ}\ \cdot \pi_2 \cdot \pi_2)\ (\pi_2 \cdot \pi_2) \rangle \\
\quad \mathbf{where}\ sep\ c = (c \equiv \texttt{' '} \vee c \equiv \texttt{'\textbackslash n'} \vee c \equiv \texttt{'\textbackslash t'})
\end{array}$$

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

$$\begin{array}{l}
h_1 = \underline{True} \\
h_2 = (sep \cdot \pi_1) \\
k_1 = \underline{0} \\
k_2 = \bot
\end{array}$$

$$k_2 = (cond\ (\widehat{(\wedge)} \cdot ((\neg \cdot sep) \times \pi_1))\ (\mathsf{succ}\ \cdot \pi_2 \cdot \pi_2)\ (\pi_2 \cdot \pi_2))$$

Para resolver este problema começamos por customizar a lei de recursividade mútua, para facilitar a compreensão do raciocínio alteramos o nome da função *lookaheadsep* para *ls* e da *wc w* para *wc*

$$\begin{cases} f \cdot \mathbf{in} = h \cdot F\ \langle f, g\rangle \\ g \cdot \mathbf{in} = k \cdot F\ \langle f, g\rangle \end{cases}$$

<=> { $\mathbf{in} = [nil, cons]$, $F\ f$ = $id + id \times f$ }

$$\begin{cases} f \cdot [nil, cons] = h \cdot (id + id \times \langle f, g\rangle) \\ g \cdot [nil, cons] = k \cdot (id + id \times \langle f, g\rangle) \end{cases}$$

<=> { f = ls, g = wc, h = [h1,h2], k = [k1,k2] }

$$\begin{cases} ls \cdot [nil, cons] = [h_1, h_2] \cdot (id + id \times \langle ls, wc\rangle) \\ wc \cdot [nil, cons] = [k_1, k_2] \cdot (id + id \times \langle ls, wc\rangle) \end{cases}$$

Neste ponto é necessário customizar a função *ls* e *wc* de modo a conseguir aplicar a lei de *Fokkinga*. Pelo enunciado temos:

$$\begin{cases} ls \cdot [\,] = True \\ ls \cdot (c : l) = sep\ c \end{cases}$$

<=> { [] = nil, (c:l) = cons(c,l) }

Aplicando a lei (27) do formulário,

$$[ls \cdot nil, ls \cdot cons] = [True, sep]$$

<=> { Definição in, Lei (20) }

$$ls \cdot \mathbf{in} = [True, sep \cdot \pi_1]$$

<=> { Lei (1) }

$$ls \cdot \mathbf{in} = [True.id, sep \cdot id \cdot \pi_1]$$

<=> { Lei (12) }

$$ls \cdot \mathbf{in} = [True.id, (sep \cdot \pi_1) \cdot (id \times \langle ls, wc\rangle)]$$

<=> { Lei (22) }

$$ls \cdot \mathbf{in} = [True, sep \cdot \pi_1] \cdot (id + id \times \langle ls, wc\rangle))$$

Neste ponto falta apenas customizar a função *wc* para aplicar a lei de *Fokkinga*.
Pelo enunciado temos:

$$\begin{cases} wc \cdot nil = \underline{0} \\ wc \cdot cons = \widehat{(\wedge)} \cdot \langle (\neg \cdot sep) \cdot \pi_1, ls \cdot \pi_2\rangle \rightarrow (\mathsf{succ}\ \cdot wc \cdot \pi_2), (wc \cdot \pi_2) \end{cases}$$

<=> { Lei (11) }

$$\begin{cases} wc \cdot nil = \underline{0} \\ wc \cdot cons = \widehat{(\wedge)} \cdot ((\neg \cdot sep) \times ls) \cdot \langle \pi_1, \pi_2\rangle \rightarrow (\mathsf{succ}\ \cdot wc \cdot \pi_2), (wc \cdot \pi_2) \end{cases}$$

<=> { Lei (8) }

$$\begin{cases} wc \cdot nil = \underline{0} \\ wc \cdot cons = \widehat{(\wedge)} \cdot ((\neg \cdot sep) \times ls) \rightarrow (\mathsf{succ}\ \cdot wc \cdot \pi_2), (wc \cdot \pi_2) \end{cases}$$

<=> { Lei (27) }

$$[wc \cdot nil, wc \cdot cons] = [\underline{0}, \widehat{(\wedge)} \cdot ((\neg \cdot sep) \times ls) \rightarrow (\mathsf{succ}\ \cdot wc \cdot \pi_2), (wc \cdot \pi_2)]$$

<=> { Lei (20) }

$$wc \cdot [nil, cons] = [\underline{0}, \widehat{(\wedge)} \cdot ((\neg \cdot sep) \times ls) \rightarrow (\mathsf{succ}\ \cdot wc \cdot \pi_2), (wc \cdot \pi_2)]$$

<=> { Definição de in }

$$wc \cdot \mathbf{in} = [\underline{0}, \widehat{(\wedge)} \cdot ((\neg \cdot sep) \times ls) \rightarrow (\mathsf{succ}\ \cdot wc \cdot \pi_2), (wc \cdot \pi_2)]$$

$<=>$ { Lei (7), wc = $\pi_2 \cdot \langle ls, wc \rangle$, ls = $\pi_1 \cdot \langle ls, wc \rangle$ }

$$wc \cdot \mathbf{in} = [\underline{0}, (\widehat{(\wedge)} \cdot (\neg \cdot sep \times \pi_1 \cdot \langle ls, wc \rangle)) \rightarrow (\mathsf{succ}\ \cdot \pi_2 \cdot (\langle ls, wc \rangle \cdot \pi_2)), (\pi_2 \cdot (\langle ls, wc \rangle \cdot \pi_2))]$$

$<=>$ { Lei (14), Lei(13) }

$$wc \cdot \mathbf{in} = [\underline{0}, (\widehat{(\wedge)} \cdot (\neg \cdot sep \times \pi_1) \cdot (id \times \langle ls, wc \rangle)) \rightarrow (\mathsf{succ}\ \cdot \pi_2 \cdot \pi_2 \cdot (id \times \langle ls, wc \rangle)), (\pi_2 \cdot \pi_2 \cdot (id \times \langle ls, w$$

$<=>$ { Lei (32) }

$$wc \cdot \mathbf{in} = [\underline{0}, ((\widehat{(\wedge)} \cdot (\neg \cdot sep \times \pi_1)) \rightarrow (\mathsf{succ}\ \cdot \pi_2 \cdot \pi_2), (\pi_2 \cdot \pi_2)) \cdot (id \times \langle ls, wc \rangle)]$$

$<=>$ { Lei (22) }

$$wc \cdot \mathbf{in} = [\underline{0}, (\widehat{(\wedge)} \cdot (\neg \cdot sep \times \pi_1)) \rightarrow (\mathsf{succ}\ \cdot \pi_2 \cdot \pi_2), (\pi_2 \cdot \pi_2)] \cdot (id + (id \times \langle ls, wc \rangle))$$

Neste ponto reunimos todas as condições para aplicar a lei de *Fokkinga*.

$$\begin{cases} ls \cdot nil = True \\ ls \cdot cons = (sep \cdot \pi_1) \cdot (id + (id \times \langle ls, wc \rangle)) \end{cases}$$

$=$ { }

$$\begin{cases} wc \cdot nil = \underline{0} \\ wc \cdot cons = (\widehat{(\wedge)} \cdot (\neg \cdot sep \times \pi_1)) \end{cases} \rightarrow (\mathsf{succ}\ \cdot \pi_2 \cdot \pi_2, \pi_2 \cdot \pi_2) \cdot (id + (id \times \langle ls, wc \rangle)))$$

$=$ { Lei (50) }

$$\langle ls, wc \rangle = (\!|\langle [True, sep \cdot \pi_1], [\underline{0}, \widehat{(\wedge)} \cdot (\neg \cdot sep \times \pi_1) \rightarrow \mathsf{succ}\ \cdot \pi_2 \cdot \pi_2, \pi_2 \cdot \pi_2] \rangle|\!)$$

$=$ { Lei (28) }

$$\langle ls, wc \rangle = (\!|[\langle True, \underline{0} \rangle, \langle sep \cdot \pi_1, \widehat{(\wedge)} \cdot (\neg \cdot sep \times \pi_1) \rightarrow \mathsf{succ}\ \cdot \pi_2 \cdot \pi_2, \pi_2 \cdot \pi_2 \rangle]|\!)$$

$prop\_wc\_w :: String \rightarrow Bool$
$prop\_wc\_w\ s = wc\_w\ s \equiv wc\_w\_final\ s$

Problema 3

$inX :: u + (i, (X\ u\ i, X\ u\ i)) \rightarrow X\ u\ i$
$inX = \bot$

$outX\ (XLeaf\ u) = \bot$
$outX\ (Node\ i\ l\ r) = \bot$

$baseX\ f\ h\ g = \bot$

$recX\ f = \bot$

$cataX\ g = \bot$

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

Problema 4

$pairL :: [a] \rightarrow [(a, a)]$
$pairL = [\!(g)\!]$ **where**
  $g = (id + \langle pairFirstTwo \cdot cons, f_2 \rangle) \cdot outList$ **where**
    $pairFirstTwo\ [x] = (x, x)$

$pairFirstTwo\ (a : b : as) = (a, b)$
$f_2\ (h, t) = \mathbf{if}\ length\ t \equiv 1\ \mathbf{then}\ [\,]\ \mathbf{else}\ t$

Para descobrir o gene do anamorfismo, começamos por desenhar o diagrama correspondente.

$$
\begin{tikzcd}
[(a,a)] & 1 + (a,a) \times [(a,a)] \arrow[l, "inList"'] \\
[a] \arrow[u, "[\!(g)\!]"] \arrow[r, "g"] & 1 + (a,a) \times [a] \arrow[u, "1+id\times[\!(g)\!]"']
\end{tikzcd}
$$

Daqui conseguimos concluir que, para o caso em que a lista não fosse vazia, teriámos que criar uma função auxiliar, que nos permitisse agrupar apenas os dois primeiros elementos da lista, e de seguida, aplicá-la à cauda da lista. Assim sendo, criamos então a função *pairFirstTwo*, que será aplicada ao resultado de cons, que é, neste caso, a lista recebida como argumento. Para evitar que o último elemento seja repetido, criamos então a função *f2* que vê se o tamanho da cauda da lista é igual a 1 e caso seja, descarta-a. Isto permite que, quando chegarmos ao caso em que a lista tenha apenas dois elementos, estes se juntem num par e o último elemento não fique repetido.

$markMap :: [Pos] \to Map \to Map$
$markMap\ l = (\![\underline{id}, f_2]\!)\ (pairL\ l)\ \mathbf{where}$
$\quad f_2 :: ((Pos, Pos), Map \to Map) \to (Map \to Map)$
$\quad f_2\ (((x1, y1), (x2, y2)), f)\ m = \mathbf{if}\ (((!!)\ ((!!)\ m\ y1)\ x1) \equiv Blocked)$
$\qquad \mathbf{then}\ f\ m$
$\qquad \mathbf{else}\ subst\ (subst\ (toCell\ (x1, y1)\ (x2, y2))\ x1\ ((!!)\ (f\ m)\ y1))\ y1\ (f\ m)$

Começamos por desenhar o diagrama do catamorfismo correspondente à função auxiliar.

$$
\begin{tikzcd}
[(Pos,Pos)] \arrow[d, "(\!|g|\!)"'] & 1 + (Pos,Pos) \times [(Pos,Pos)] \arrow[l, "inList"'] \arrow[d, "id+id\times(\!|g|\!)"] \\
f & 1 + ((Pos,Pos), f) \arrow[l, "g=[\underline{id}, f_2]"]
\end{tikzcd}
$$

Olhando para o diagrama acima é então possível perceber que *f2* receberá como argumento um par. Do lado esquerdo está presente um par de posições, correspondente ao primeiro elemento da lista original e, do lado direito uma função que permitirá alterar o mapa com as posições presentes na cauda da lista.

Tendo a função auxiliar o tipo $[(Pos, Pos)] \to Map \to Map$, é então possível concluir que *f* terá o tipo $Map \to Map$ e, consequentemente, *f2* o tipo $((Pos, Pos), Map \to Map) \to (Map \to Map)$.

Para a implementação de *f2* utilizamos então a função auxiliar *subst*, que nos permite susbtituir no mapa alterado pela função *f*, o resultado da função *toCell* aplicada às posições que se encontram no início da lista. É tida em causa a situação em que nos encontramos numa célula que está bloqueada, sendo que neste caso não fazemos qualquer tipo de alteração ao mapa.

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

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:*

$prop\_reconst\ l = (l \not\equiv [\,] \wedge (length\ l \not\equiv 1) \wedge init\ l \equiv (\mathsf{map}\ \pi_1\ (pairL\ l)) \wedge tail\ l \equiv (\mathsf{map}\ \pi_2\ (pairL\ l))) \vee (length\ l$

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: 7 de Fevereiro de 2022.