Cálculo de Programas
Trabalho Prático
MiEI+LCC — 2018/19

Departamento de Informática
Universidade do Minho

Junho de 2019

Grupo nr.	22
a84003	Beatriz Rocha
a85308	Filipe Guimarães
a84073	Gonçalo Ferreira

1 Preâmbulo

A disciplina de 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 dois cursos que têm esta disciplina, restringe-se a aplicação deste método à programação funcional em Haskell. 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, validá-los, e a produzir textos técnico-científicos de qualidade.

2 Documentação

Para cumprir de forma integrada os objectivos enunciados acima vamos recorrer a uma técnica de programação dita "literária" [**?**], 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 `cp1819t.pdf` que está a ler é já um exemplo de programação literária: foi gerado a partir do texto fonte `cp1819t.lhs`[1] que encontrará no material pedagógico desta disciplina descompactando o ficheiro `cp1819t.zip` e executando

```
$ lhs2TeX cp1819t.lhs > cp1819t.tex
$ pdflatex cp1819t
```

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
```

Por outro lado, o mesmo ficheiro `cp1819t.lhs` é executável e contém o "kit" básico, escrito em Haskell, para realizar o trabalho. Basta executar

```
$ ghci cp1819t.lhs
```

[1] O suffixo 'lhs' quer dizer *literate Haskell*.

Abra o ficheiro `cp1819t.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}
```

vai ser seleccionado pelo GHCi para ser executado.

3 Como realizar o trabalho

Este trabalho teórico-prático deve ser realizado por grupos de três 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 de trabalho por forma a poderem responder às questões que serão colocadas 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 D 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 cp1819t.aux
$ makeindex cp1819t.idx
```

e recompilar o texto como acima se indicou. Dever-se-á ainda instalar o utilitário QuickCheck, que ajuda a validar programas em Haskell e a biblioteca Gloss para geração de gráficos 2D:

```
$ cabal install QuickCheck gloss
```

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

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

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

Problema 1

Um compilador é um programa que traduz uma linguagem dita de *alto nível* numa linguagem (dita de *baixo nível*) que seja executável por uma máquina. Por exemplo, o GCC compila C/C++ em código objecto que corre numa variedade de arquitecturas.

Compiladores são normalmente programas complexos. Constam essencialmente de duas partes: o *analisador sintático* que lê o texto de entrada (o programa *fonte* a compilar) e cria uma sua representação interna, estruturada em árvore; e o *gerador de código* que converte essa representação interna em código executável. Note-se que tal representação intermédia pode ser usada para outros fins, por exemplo, para gerar uma listagem de qualidade (*pretty print*) do programa fonte.

O projecto de compiladores é um assunto complexo que será assunto de outras disciplinas. Neste trabalho pretende-se apenas fazer uma introdução ao assunto, mostrando como tais programas se podem construir funcionalmente à custa de cata/ana/hilo-morfismos da linguagem em causa.

Para cumprirmos o nosso objectivo, a linguagem desta questão terá que ser, naturalmente, muito simples: escolheu-se a das expressões aritméticas com inteiros, *eg.* $1+2$, $3*(4+5)$ etc. Como representação interna adopta-se o seguinte tipo polinomial, igualmente simples:

$$\mathbf{data}\ Expr = Num\ Int \mid Bop\ Expr\ Op\ Expr$$
$$\mathbf{data}\ Op = Op\ String$$

1. Escreva as definições dos {cata, ana e hilo}-morfismos deste tipo de dados segundo o método ensinado nesta disciplina (recorde módulos como *eg.* BTree etc).

2. Como aplicação do módulo desenvolvido no ponto 1, defina como {cata, ana ou hilo}-morfismo a função seguinte:

   - $calcula :: Expr \to Int$ que calcula o valor de uma expressão;

     Propriedade QuickCheck 1 *O valor zero é um elemento neutro da adição.*

     $$
     \begin{array}{l}
     prop\_neutro1 :: Expr \to Bool \\
     prop\_neutro1 = calcula \cdot addZero \equiv calcula \ \mathbf{where} \\
     \quad addZero\ e = Bop\ (Num\ 0)\ (Op\ \texttt{"+"})\ e \\
     prop\_neutro2 :: Expr \to Bool \\
     prop\_neutro2 = calcula \cdot addZero \equiv calcula \ \mathbf{where} \\
     \quad addZero\ e = Bop\ e\ (Op\ \texttt{"+"})\ (Num\ 0)
     \end{array}
     $$

     Propriedade QuickCheck 2 *As operações de soma e multiplicação são comutativas.*

     $$
     \begin{array}{l}
     prop\_comuta = calcula \cdot mirror \equiv calcula \ \mathbf{where} \\
     \quad mirror = cataExpr\ [Num, g2] \\
     \quad g2 = \widehat{\widehat{Bop}} \cdot (swap \times id) \cdot assocl \cdot (id \times swap)
     \end{array}
     $$

3. Defina como {cata, ana ou hilo}-morfismos as funções

   - $compile :: String \to Codigo$ - trata-se do compilador propriamente dito. Deverá ser gerado código posfixo para uma máquina elementar de stack. O tipo *Codigo* pode ser definido à escolha. Dão-se a seguir exemplos de comportamentos aceitáveis para esta função:

     ```
     Tp4> compile "2+4"
     ["PUSH 2", "PUSH 4", "ADD"]
     Tp4> compile "3*(2+4)"
     ["PUSH 3", "PUSH 2", "PUSH 4", "ADD", "MUL"]
     Tp4> compile "(3*2)+4"
     ["PUSH 3", "PUSH 2", "MUL", "PUSH 4", "ADD"]
     Tp4>
     ```

   - $show' :: Expr \to String$ - gera a representação textual de uma *Expr* pode encarar-se como o *pretty printer* associado ao nosso compilador

     Propriedade QuickCheck 3 *Em anexo, é fornecido o código da função $readExp$, que é "inversa"da função $show'$, tal como a propriedade seguinte descreve:*

     $$
     \begin{array}{l}
     prop\_inv :: Expr \to Bool \\
     prop\_inv = \pi_1 \cdot head \cdot readExp \cdot show' \equiv id
     \end{array}
     $$

Valorização Em anexo é apresentado código Haskell que permite declarar *Expr* como instância da classe *Read*. Neste contexto, *read* pode ser vista como o analisador sintático do nosso minúsculo compilador de expressões aritméticas.

Analise o código apresentado, corra-o e escreva no seu relatório uma explicação **breve** do seu funcionamento, que deverá saber defender aquando da apresentação oral do relatório.

Exprima ainda o analisador sintático *readExp* como um anamorfismo.

Problema 2

Pretende-se neste problema definir uma linguagem gráfica "brinquedo" a duas dimensões (2D) capaz de especificar e desenhar agregações de caixas que contêm informação textual. Vamos designar essa linguagem por *L2D* e vamos defini-la como um tipo em Haskell:

$$\mathbf{type}\ L2D = X\ Caixa\ Tipo$$

onde X é a estrutura de dados

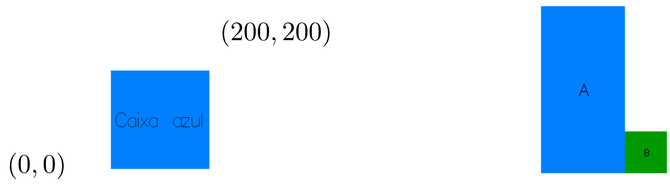

Figura 1: Caixa simples e caixa composta.

$$\mathbf{data}\ X\ a\ b = Unid\ a \mid Comp\ b\ (X\ a\ b)\ (X\ a\ b)\ \mathbf{deriving}\ Show$$

e onde:

$$\mathbf{type}\ Caixa = ((Int, Int), (Texto, G.Color))$$
$$\mathbf{type}\ Texto = String$$

Assim, cada caixa de texto é especificada pela sua largura, altura, o seu texto e a sua côr.[2] Por exemplo,

$((200, 200), (\texttt{"Caixa azul"}, col\_blue))$

designa a caixa da esquerda da figura 1.

O que a linguagem *L2D* faz é agregar tais caixas tipográficas umas com as outras segundo padrões especificados por vários "tipos", a saber,

$$\mathbf{data}\ Tipo = V \mid Vd \mid Ve \mid H \mid Ht \mid Hb$$

com o seguinte significado:

V - agregação vertical alinhada ao centro

Vd - agregação vertical justificada à direita

Ve - agregação vertical justificada à esquerda

H - agregação horizontal alinhada ao centro

Hb - agregação horizontal alinhada pela base

Ht - agregação horizontal alinhada pelo topo

Como *L2D* instancia o parâmetro b de X com *Tipo*, é fácil de ver que cada "frase" da linguagem *L2D* é representada por uma árvore binária em que cada nó indica qual o tipo de agregação a aplicar às suas duas sub-árvores. Por exemplo, a frase

$$\begin{aligned} ex2 = Comp\ Hb\ &(Unid\ ((100, 200), (\texttt{"A"}, col\_blue)))\\ &(Unid\ ((50, 50), (\texttt{"B"}, col\_green))) \end{aligned}$$

deverá corresponder à imagem da direita da figura 1. E poder-se-á ir tão longe quando a linguagem o permita. Por exemplo, pense na estrutura da frase que representa o *layout* da figura 2.

É importante notar que cada "caixa" não dispõe informação relativa ao seu posicionamento final na figura. De facto, é a posição relativa que deve ocupar face às restantes caixas que irá determinar a sua posição final. Este é um dos objectivos deste trabalho: *calcular o posicionamento absoluto de cada uma das caixas por forma a respeitar as restrições impostas pelas diversas agregações*. Para isso vamos considerar um tipo de dados que comporta a informação de todas as caixas devidamente posicionadas (i.e. com a informação adicional da origem onde a caixa deve ser colocada).

[2] Pode relacionar *Caixa* com as caixas de texto usadas nos jornais ou com *frames* da linguagem HTML usada na Internet.

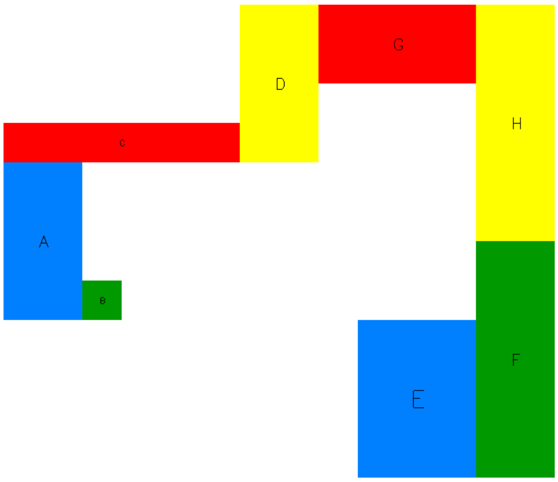

Figura 2: *Layout* feito de várias caixas coloridas.

$\textbf{type}\ Fig = [(Origem, Caixa)]$
$\textbf{type}\ Origem = (Float, Float)$

A informação mais relevante deste tipo é a referente à lista de "caixas posicionadas" (tipo $(Origem, Caixa)$). Regista-se aí a origem da caixa que, com a informação da sua altura e comprimento, permite definir todos os seus pontos (consideramos as caixas sempre paralelas aos eixos).

1. Forneça a definição da função $calc\_origems$, que calcula as coordenadas iniciais das caixas no plano:

   $$calc\_origems :: (L2D, Origem) \to X\ (Caixa, Origem)\ ()$$

2. Forneça agora a definição da função $agrup\_caixas$, que agrupa todas as caixas e respectivas origens numa só lista:

   $$agrup\_caixas :: X\ (Caixa, Origem)\ () \to Fig$$

Um segundo problema neste projecto é *descobrir como visualizar a informação gráfica calculada por* $desenho$. A nossa estratégia para superar o problema baseia-se na biblioteca Gloss, que permite a geração de gráficos 2D. Para tal disponibiliza-se a função

$$crCaixa :: Origem \to Float \to Float \to String \to G.Color \to G.Picture$$

que cria um rectângulo com base numa coordenada, um valor para a largura, um valor para a altura, um texto que irá servir de etiqueta, e a cor pretendida. Disponibiliza-se também a função

$$display :: G.Picture \to \mathsf{IO}\ ()$$

que dado um valor do tipo $G.picture$ abre uma janela com esse valor desenhado. O objectivo final deste exercício é implementar então uma função

$$mostra\_caixas :: (L2D, Origem) \to \mathsf{IO}\ ()$$

que dada uma frase da linguagem $L2D$ e coordenadas iniciais apresenta o respectivo desenho no ecrã. **Sugestão**: Use a função $G.pictures$ disponibilizada na biblioteca Gloss.

Problema 3

Nesta disciplina estudou-se como fazer programação dinâmica por cálculo, recorrendo à lei de recursividade mútua.[3]

Para o caso de funções sobre os números naturais ($\mathbb{N}_0$, com functor $\mathsf{F}\ X = 1 + X$) é fácil derivar-se da lei que foi estudada uma *regra de algibeira* que se pode ensinar a programadores que não tenham estudado Cálculo de Programas. Apresenta-se de seguida essa regra, tomando como exemplo o cálculo do ciclo-for que implementa a função de Fibonacci, recordar o sistema

$$\begin{array}{l} \mathit{fib}\ 0 = 1 \\ \mathit{fib}\ (n+1) = f\ n \end{array}$$
$$\begin{array}{l} f\ 0 = 1 \\ f\ (n+1) = \mathit{fib}\ n + f\ n \end{array}$$

Obter-se-á de imediato

$$\begin{array}{l} \mathit{fib}' = \pi_1 \cdot \mathsf{for}\ \mathit{loop}\ \mathit{init}\ \mathbf{where} \\ \quad \mathit{loop}\ (\mathit{fib}, f) = (f, \mathit{fib} + f) \\ \quad \mathit{init} = (1, 1) \end{array}$$

usando as regras seguintes:

- O corpo do ciclo loop terá tantos argumentos quanto o número de funções mutuamente recursivas.
- Para as variáveis escolhem-se os próprios nomes das funções, pela ordem que se achar conveniente.[4]
- Para os resultados vão-se buscar as expressões respectivas, retirando a variável n.
- Em init coleccionam-se os resultados dos casos de base das funções, pela mesma ordem.

Mais um exemplo, envolvendo polinómios no segundo grau a $x^2 + bx + c$ em $\mathbb{N}_0$. Seguindo o método estudado nas aulas[5], de $f\ x = ax^2 + bx + c$ derivam-se duas funções mutuamente recursivas:

$$\begin{array}{l} f\ 0 = c \\ f\ (n+1) = f\ n + k\ n \end{array}$$
$$\begin{array}{l} k\ 0 = a + b \\ k\ (n+1) = k\ n + 2\ a \end{array}$$

Seguindo a regra acima, calcula-se de imediato a seguinte implementação, em Haskell:

$$\begin{array}{l} f'\ a\ b\ c = \pi_1 \cdot \mathsf{for}\ \mathit{loop}\ \mathit{init}\ \mathbf{where} \\ \quad \mathit{loop}\ (f, k) = (f + k, k + 2 * a) \\ \quad \mathit{init} = (c, a + b) \end{array}$$

Qual é o assunto desta questão, então? Considerem fórmula que dá a série de Taylor da função coseno:

$$\mathit{cos}\ x = \sum_{i=0}^{\infty} \frac{(-1)^i}{(2i)!} x^{2i}$$

Pretende-se o ciclo-for que implementa a função $\mathit{cos}'\ x\ n$ que dá o valor dessa série tomando i até n inclusivé:

$$\mathit{cos}'\ x = \cdots \cdot \mathsf{for}\ \mathit{loop}\ \mathit{init}\ \mathbf{where}\ \cdots$$

Sugestão: Começar por estudar muito bem o processo de cálculo dado no anexo B para o problema (semelhante) da função exponencial.

Propriedade QuickCheck 4 *Testes de que $\mathit{cos}'\ x$ calcula bem o coseno de π e o coseno de $\pi\ /\ 2$:*

$$\begin{array}{l} \mathit{prop\_cos1}\ n = n \geqslant 10 \Rightarrow \mathit{abs}\ (\mathit{cos}\ \pi - \mathit{cos}'\ \pi\ n) < 0.001 \\ \mathit{prop\_cos2}\ n = n \geqslant 10 \Rightarrow \mathit{abs}\ (\mathit{cos}\ (\pi\ /\ 2) - \mathit{cos}'\ (\pi\ /\ 2)\ n) < 0.001 \end{array}$$

[3] Lei (3.94) em [?], página 98.
[4] Podem obviamente usar-se outros símbolos, mas numa primeiraleitura dá jeito usarem-se tais nomes.
[5] Secção 3.17 de [?].

Valorização Transliterar cos' para a linguagem C; compilar e testar o código. Conseguia, por intuição apenas, chegar a esta função?

Problema 4

Pretende-se nesta questão desenvolver uma biblioteca de funções para manipular *sistemas de ficheiros* genéricos. Um sistema de ficheiros será visto como uma associação de *nomes* a ficheiros ou *directorias*. Estas últimas serão vistas como sub-sistemas de ficheiros e assim recursivamente. Assumindo que a é o tipo dos identificadores dos ficheiros e directorias, e que b é o tipo do conteúdo dos ficheiros, podemos definir um tipo indutivo de dados para representar sistemas de ficheiros da seguinte forma:

$$\mathbf{data}\ FS\ a\ b = FS\ [(a, Node\ a\ b)]\ \mathbf{deriving}\ (Eq, Show)$$
$$\mathbf{data}\ Node\ a\ b = File\ b \mid Dir\ (FS\ a\ b)\ \mathbf{deriving}\ (Eq, Show)$$

Um caminho (*path*) neste sistema de ficheiros pode ser representado pelo seguinte tipo de dados:

$$\mathbf{type}\ Path\ a = [a]$$

Assumindo estes tipos de dados, o seguinte termo

```
FS [("f1", File "Ola"),
  ("d1", Dir (FS [("f2", File "Ole"),
    ("f3", File "Ole")
    ]))
  ]
```

representará um sistema de ficheiros em cuja raíz temos um ficheiro chamado *f1* com conteúdo `"Ola"` e uma directoria chamada `"d1"` constituída por dois ficheiros, um chamado `"f2"` e outro chamado `"f3"`, ambos com conteúdo `"Ole"`. Neste caso, tanto o tipo dos identificadores como o tipo do conteúdo dos ficheiros é $String$. No caso geral, o conteúdo de um ficheiro é arbitrário: pode ser um binário, um texto, uma colecção de dados, etc.

A definição das usuais funções $inFS$ e $recFS$ para este tipo é a seguinte:

$$inFS = FS \cdot \mathsf{map}\ (id \times inNode)$$
$$inNode = [File, Dir]$$
$$recFS\ f = baseFS\ id\ id\ f$$

Suponha que se pretende definir como um *catamorfismo* a função que conta o número de ficheiros existentes num sistema de ficheiros. Uma possível definição para esta função seria:

$$conta :: FS\ a\ b \to Int$$
$$conta = cataFS\ (sum \cdot \mathsf{map}\ ([\underline{1}, id] \cdot \pi_2))$$

O que é para fazer:

1. Definir as funções $outFS$, $baseFS$, $cataFS$, $anaFS$ e $hyloFS$.
2. Apresentar, no relatório, o diagrama de $cataFS$.
3. Definir as seguintes funções para manipulação de sistemas de ficheiros usando, obrigatoriamente, catamorfismos, anamorfismos ou hilomorfismos:

   (a) Verificação da integridade do sistema de ficheiros (i.e. verificar que não existem identificadores repetidos dentro da mesma directoria).
   $check :: FS\ a\ b \to Bool$

   Propriedade QuickCheck 5 *A integridade de um sistema de ficheiros não depende da ordem em que os últimos são listados na sua directoria:*

   $$prop\_check :: FS\ String\ String \to Bool$$
   $$prop\_check = check \cdot (cataFS\ (inFS \cdot reverse)) \equiv check$$

(b) Recolha do conteúdo de todos os ficheiros num arquivo indexado pelo *path*.
$tar :: FS\ a\ b \to [(Path\ a, b)]$

Propriedade QuickCheck 6 *O número de ficheiros no sistema deve ser igual ao número de ficheiros listados pela função* tar.

$$prop\_tar :: FS\ String\ String \to Bool$$
$$prop\_tar = \mathsf{length} \cdot tar \equiv conta$$

(c) Transformação de um arquivo com o conteúdo dos ficheiros indexado pelo *path* num sistema de ficheiros.
$untar :: [(Path\ a, b)] \to FS\ a\ b$
Sugestão: Use a função $joinDupDirs$ para juntar directorias que estejam na mesma pasta e que possuam o mesmo identificador.

Propriedade QuickCheck 7 *A composição* $tar \cdot untar$ *preserva o número de ficheiros no sistema.*

$$prop\_untar :: [(Path\ String, String)] \to Property$$
$$prop\_untar = validPaths \Rightarrow ((\mathsf{length} \cdot tar \cdot untar) \equiv \mathsf{length})$$
$$validPaths :: [(Path\ String, String)] \to Bool$$
$$validPaths = (\equiv 0) \cdot \mathsf{length} \cdot (filter\ (\lambda(a, \_) \to \mathsf{length}\ a \equiv 0))$$

(d) Localização de todos os *paths* onde existe um determinado ficheiro.
$find :: a \to FS\ a\ b \to [Path\ a]$

Propriedade QuickCheck 8 *A composição* $tar \cdot untar$ *preserva todos os ficheiros no sistema.*

$$prop\_find :: String \to FS\ String\ String \to Bool$$
$$prop\_find = curry\ \$$$
$$\mathsf{length} \cdot \widehat{find} \equiv \mathsf{length} \cdot \widehat{find} \cdot (id \times (untar \cdot tar))$$

(e) Criação de um novo ficheiro num determinado *path*.
$new :: Path\ a \to b \to FS\ a\ b \to FS\ a\ b$

Propriedade QuickCheck 9 *A adição de um ficheiro não existente no sistema não origina ficheiros duplicados.*

$$prop\_new :: ((Path\ String, String), FS\ String\ String) \to Property$$
$$prop\_new = ((validPath \wedge notDup) \wedge (check \cdot \pi_2)) \Rightarrow$$
$$(checkFiles \cdot \widehat{\widehat{new}})\ \mathbf{where}$$
$$validPath = (\not\equiv 0) \cdot \mathsf{length} \cdot \pi_1 \cdot \pi_1$$
$$notDup = \neg \cdot \widehat{elem} \cdot (\pi_1 \times ((\mathsf{fmap}\ \pi_1) \cdot tar))$$

Questão: *Supondo-se que no código acima se substitui a propriedade* $checkFiles$ *pela propriedade mais fraca* $check$*, será que a propriedade* $prop\_new$ *ainda é válida? Justifique a sua resposta.*

Propriedade QuickCheck 10 *A listagem de ficheiros logo após uma adição nunca poderá ser menor que a listagem de ficheiros antes dessa mesma adição.*

$$prop\_new2 :: ((Path\ String, String), FS\ String\ String) \to Property$$
$$prop\_new2 = validPath \Rightarrow ((\mathsf{length} \cdot tar \cdot \pi_2) \leqslant (\mathsf{length} \cdot tar \cdot \widehat{\widehat{new}}))\ \mathbf{where}$$
$$validPath = (\not\equiv 0) \cdot \mathsf{length} \cdot \pi_1 \cdot \pi_1$$

(f) Duplicação de um ficheiro.
$cp :: Path\ a \to Path\ a \to FS\ a\ b \to FS\ a\ b$

Propriedade QuickCheck 11 *A listagem de ficheiros com um dado nome não diminui após uma duplicação.*

$$prop\_cp :: ((Path\ String, Path\ String), FS\ String\ String) \to Bool$$
$$prop\_cp = \mathsf{length} \cdot tar \cdot \pi_2 \leqslant \mathsf{length} \cdot tar \cdot \widehat{\widehat{cp}}$$

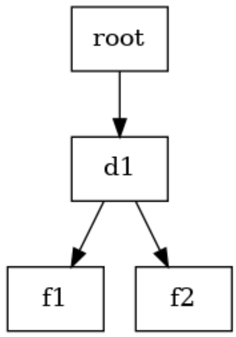

Figura 3: Exemplo de um sistema de ficheiros visualizado em Graphviz.

(g) Eliminação de um ficheiro.
$rm :: Path\ a \to FS\ a\ b \to FS\ a\ b$

Sugestão: Construir um anamorfismo $nav :: (Path\ a, FS\ a\ b) \to FS\ a\ b$ que navegue por um sistema de ficheiros tendo como base o $path$ dado como argumento.

Propriedade QuickCheck 12 *Remover duas vezes o mesmo ficheiro tem o mesmo efeito que o remover apenas uma vez.*

$$prop\_rm :: (Path\ String, FS\ String\ String) \to Bool$$
$$prop\_rm = \widehat{rm} \cdot \langle \pi_1, \widehat{rm} \rangle \equiv \widehat{rm}$$

Propriedade QuickCheck 13 *Adicionar um ficheiro e de seguida remover o mesmo não origina novos ficheiros no sistema.*

$$prop\_rm2 :: ((Path\ String, String), FS\ String\ String) \to Property$$
$$prop\_rm2 = validPath \Rightarrow ((\mathsf{length}\ \cdot tar \cdot \widehat{rm} \cdot \langle \pi_1 \cdot \pi_1, \widehat{new} \rangle)$$
$$\leqslant (\mathsf{length}\ \cdot tar \cdot \pi_2))\ \mathbf{where}$$
$$validPath = (\not\equiv 0) \cdot \mathsf{length}\ \cdot \pi_1 \cdot \pi_1$$

Valorização Definir uma função para visualizar em Graphviz a estrutura de um sistema de ficheiros. A Figura 3, por exemplo, apresenta a estrutura de um sistema com precisamente dois ficheiros dentro de uma directoria chamada `"d1"`.

Para realizar este exercício será necessário apenas escrever o anamorfismo

$$cFS2Exp :: (a, FS\ a\ b) \to (Exp\ ()\ a)$$

que converte a estrutura de um sistema de ficheiros numa árvore de expressões descrita em Exp.hs. A função $dotFS$ depois tratará de passar a estrutura do sistema de ficheiros para o visualizador.

Anexos

A Como exprimir cálculos e diagramas em LaTeX/lhs2tex

Estudar o texto fonte deste trabalho para obter o efeito:[6]

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

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

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

B Programação dinâmica por recursividade múltipla

Neste anexo dão-se os detalhes da resolução do Exercício 3.30 dos apontamentos da disciplina[7], onde se pretende implementar um ciclo que implemente o cálculo da aproximação até $i = n$ da função exponencial $exp\ x = e^x$ via série de Taylor:

$$
exp\ x = \sum_{i=0}^{\infty} \frac{x^i}{i!} \tag{1}
$$

Seja $e\ x\ n = \sum_{i=0}^{n} \frac{x^i}{i!}$ a função que dá essa aproximação. É fácil de ver que $e\ x\ 0 = 1$ e que $e\ x\ (n+1) = e\ x\ n + \frac{x^{n+1}}{(n+1)!}$. Se definirmos $h\ x\ n = \frac{x^{n+1}}{(n+1)!}$ teremos $e\ x$ e $h\ x$ em recursividade mútua. Se repetirmos o processo para $h\ x\ n$ etc obteremos no total três funções nessa mesma situação:

$$
\begin{array}{l}
e\ x\ 0 = 1 \\
e\ x\ (n+1) = h\ x\ n + e\ x\ n \\
h\ x\ 0 = x \\
h\ x\ (n+1) = x\ /\ (s\ n) * h\ x\ n \\
s\ 0 = 2 \\
s\ (n+1) = 1 + s\ n
\end{array}
$$

Segundo a *regra de algibeira* descrita na página 3 deste enunciado, ter-se-á, de imediato:

$$
\begin{array}{l}
e'\ x = prj \cdot \mathsf{for}\ loop\ init\ \mathbf{where} \\
\quad init = (1, x, 2) \\
\quad loop\ (e, h, s) = (h + e, x\ /\ s * h, 1 + s) \\
\quad prj\ (e, h, s) = e
\end{array}
$$

[6] Exemplos tirados de [?].
[7] Cf. [?], página 102.

C Código fornecido

Problema 1

Tipos:

$$
\begin{array}{l}
\textbf{data}\ \mathit{Expr} = \mathit{Num}\ \mathit{Int} \\
\quad \mid \mathit{Bop}\ \mathit{Expr}\ \mathit{Op}\ \mathit{Expr}\ \textbf{deriving}\ (\mathit{Eq}, \mathit{Show}) \\
\textbf{data}\ \mathit{Op} = \mathit{Op}\ \mathit{String}\ \textbf{deriving}\ (\mathit{Eq}, \mathit{Show}) \\
\textbf{type}\ \mathit{Codigo} = [\mathit{String}]
\end{array}
$$

Functor de base:

$$\mathit{baseExpr}\ f\ g = \mathit{id} + (f \times (g \times g))$$

Instâncias:

$$
\begin{array}{l}
\textbf{instance}\ \mathit{Read}\ \mathit{Expr}\ \textbf{where} \\
\quad \mathit{readsPrec}\ \_ = \mathit{readExp}
\end{array}
$$

Read para Exp's:

$$
\begin{array}{l}
\mathit{readOp} :: \mathit{String} \rightarrow [(\mathit{Op}, \mathit{String})] \\
\mathit{readOp}\ \mathit{input} = \textbf{do} \\
\quad (x, y) \leftarrow \mathit{lex}\ \mathit{input} \\
\quad \mathit{return}\ ((\mathit{Op}\ x), y) \\
\mathit{readNum} :: \mathit{ReadS}\ \mathit{Expr} \\
\mathit{readNum} = (\mathsf{map}\ \ (\lambda(x, y) \rightarrow ((\mathit{Num}\ x), y))) \cdot \mathit{reads} \\
\mathit{readBinOp} :: \mathit{ReadS}\ \mathit{Expr} \\
\mathit{readBinOp} = (\mathsf{map}\ \ (\lambda((x, (y, z)), t) \rightarrow ((\mathit{Bop}\ x\ y\ z), t))) \cdot \\
\quad ((\mathit{readNum}\ \text{`}\mathit{ou}\text{`}\ (\mathit{pcurvos}\ \mathit{readExp})) \\
\qquad \text{`}\mathit{depois}\text{`}\ (\mathit{readOp}\ \text{`}\mathit{depois}\text{`}\ \mathit{readExp})) \\
\mathit{readExp} :: \mathit{ReadS}\ \mathit{Expr} \\
\mathit{readExp} = \mathit{readBinOp}\ \text{`}\mathit{ou}\text{`}\ (\\
\quad \mathit{readNum}\ \text{`}\mathit{ou}\text{`}\ (\\
\quad \mathit{pcurvos}\ \mathit{readExp}))
\end{array}
$$

Combinadores:

$$
\begin{array}{l}
\mathit{depois} :: (\mathit{ReadS}\ a) \rightarrow (\mathit{ReadS}\ b) \rightarrow \mathit{ReadS}\ (a, b) \\
\mathit{depois}\ \_\ \_\ [\,] = [\,] \\
\mathit{depois}\ \mathit{r1}\ \mathit{r2}\ \mathit{input} = [((x, y), i_2) \mid (x, i_1) \leftarrow \mathit{r1}\ \mathit{input}, \\
\quad (y, i_2) \leftarrow \mathit{r2}\ i_1] \\
\mathit{readSeq} :: (\mathit{ReadS}\ a) \rightarrow \mathit{ReadS}\ [a] \\
\mathit{readSeq}\ r\ \mathit{input} \\
\quad = \textbf{case}\ (r\ \mathit{input})\ \textbf{of} \\
\qquad [\,] \rightarrow [([\,], \mathit{input})] \\
\qquad l \rightarrow \mathit{concat}\ (\mathsf{map}\ \ \mathit{continua}\ l) \\
\qquad\quad \textbf{where}\ \mathit{continua}\ (a, i) = \mathsf{map}\ \ (c\ a)\ (\mathit{readSeq}\ r\ i) \\
\qquad\qquad c\ x\ (\mathit{xs}, i) = ((x : \mathit{xs}), i) \\
\mathit{ou} :: (\mathit{ReadS}\ a) \rightarrow (\mathit{ReadS}\ a) \rightarrow \mathit{ReadS}\ a \\
\mathit{ou}\ \mathit{r1}\ \mathit{r2}\ \mathit{input} = (\mathit{r1}\ \mathit{input}) +\!\!+ (\mathit{r2}\ \mathit{input}) \\
\mathit{senao} :: (\mathit{ReadS}\ a) \rightarrow (\mathit{ReadS}\ a) \rightarrow \mathit{ReadS}\ a \\
\mathit{senao}\ \mathit{r1}\ \mathit{r2}\ \mathit{input} = \textbf{case}\ (\mathit{r1}\ \mathit{input})\ \textbf{of} \\
\quad [\,] \rightarrow \mathit{r2}\ \mathit{input} \\
\quad l \quad\ \rightarrow l \\
\mathit{readConst} :: \mathit{String} \rightarrow \mathit{ReadS}\ \mathit{String} \\
\mathit{readConst}\ c = (\mathit{filter}\ ((\equiv c) \cdot \pi_1)) \cdot \mathit{lex} \\
\mathit{pcurvos} = \mathit{parentesis}\ \texttt{'('}\ \texttt{')'}
\end{array}
$$

$$
\begin{array}{l}
\mathit{prectos} = \mathit{parentesis}\ \texttt{'['}\ \texttt{']'} \\
\mathit{chavetas} = \mathit{parentesis}\ \texttt{'\{'}\ \texttt{'\}'} \\
\mathit{parentesis} :: \mathit{Char} \to \mathit{Char} \to (\mathit{ReadS}\ a) \to \mathit{ReadS}\ a \\
\mathit{parentesis}\ \_\ \_\ \_\ [\,] = [\,] \\
\mathit{parentesis}\ \mathit{ap}\ \mathit{pa}\ r\ \mathit{input} \\
\quad = \mathbf{do} \\
\qquad ((\_, (x, \_)), c) \leftarrow ((\mathit{readConst}\ [\mathit{ap}])\ `\mathit{depois}`\ (\\
\qquad\quad r\ `\mathit{depois}`\ (\\
\qquad\quad \mathit{readConst}\ [\mathit{pa}]))) \ \mathit{input} \\
\qquad \mathit{return}\ (x, c)
\end{array}
$$

Problema 2

Tipos:

$$
\begin{array}{l}
\mathbf{type}\ \mathit{Fig} = [(\mathit{Origem}, \mathit{Caixa})] \\
\mathbf{type}\ \mathit{Origem} = (\mathit{Float}, \mathit{Float})
\end{array}
$$

"Helpers":

$$
\begin{array}{l}
\mathit{col\_blue} = G.\mathit{azure} \\
\mathit{col\_green} = \mathit{darkgreen} \\
\mathit{darkgreen} = G.\mathit{dark}\ (G.\mathit{dark}\ G.\mathit{green})
\end{array}
$$

Exemplos:

$$
\begin{array}{l}
\mathit{ex1Caixas} = G.\mathit{display}\ (G.\mathit{InWindow}\ \texttt{"Problema 4"}\ (400, 400)\ (40, 40))\ G.\mathit{white}\ \$ \\
\quad \mathit{crCaixa}\ (0, 0)\ 200\ 200\ \texttt{"Caixa azul"}\ \mathit{col\_blue} \\
\mathit{ex2Caixas} = G.\mathit{display}\ (G.\mathit{InWindow}\ \texttt{"Problema 4"}\ (400, 400)\ (40, 40))\ G.\mathit{white}\ \$ \\
\quad \mathit{caixasAndOrigin2Pict}\ ((\mathit{Comp}\ \mathit{Hb}\ \mathit{bbox}\ \mathit{gbox}), (0.0, 0.0))\ \mathbf{where} \\
\quad \mathit{bbox} = \mathit{Unid}\ ((100, 200), (\texttt{"A"}, \mathit{col\_blue})) \\
\quad \mathit{gbox} = \mathit{Unid}\ ((50, 50), (\texttt{"B"}, \mathit{col\_green})) \\
\mathit{ex3Caixas} = G.\mathit{display}\ (G.\mathit{InWindow}\ \texttt{"Problema 4"}\ (400, 400)\ (40, 40))\ G.\mathit{white}\ \mathit{mtest}\ \mathbf{where} \\
\quad \mathit{mtest} = \mathit{caixasAndOrigin2Pict}\ \$\ (\mathit{Comp}\ \mathit{Hb}\ (\mathit{Comp}\ \mathit{Ve}\ \mathit{bot}\ \mathit{top})\ (\mathit{Comp}\ \mathit{Ve}\ \mathit{gbox2}\ \mathit{ybox2}), (0.0, 0.0)) \\
\quad \mathit{bbox1} = \mathit{Unid}\ ((100, 200), (\texttt{"A"}, \mathit{col\_blue})) \\
\quad \mathit{bbox2} = \mathit{Unid}\ ((150, 200), (\texttt{"E"}, \mathit{col\_blue})) \\
\quad \mathit{gbox1} = \mathit{Unid}\ ((50, 50), (\texttt{"B"}, \mathit{col\_green})) \\
\quad \mathit{gbox2} = \mathit{Unid}\ ((100, 300), (\texttt{"F"}, \mathit{col\_green})) \\
\quad \mathit{rbox1} = \mathit{Unid}\ ((300, 50), (\texttt{"C"}, G.\mathit{red})) \\
\quad \mathit{rbox2} = \mathit{Unid}\ ((200, 100), (\texttt{"G"}, G.\mathit{red})) \\
\quad \mathit{wbox1} = \mathit{Unid}\ ((450, 200), (\texttt{""}, G.\mathit{white})) \\
\quad \mathit{ybox1} = \mathit{Unid}\ ((100, 200), (\texttt{"D"}, G.\mathit{yellow})) \\
\quad \mathit{ybox2} = \mathit{Unid}\ ((100, 300), (\texttt{"H"}, G.\mathit{yellow})) \\
\quad \mathit{bot} = \mathit{Comp}\ \mathit{Hb}\ \mathit{wbox1}\ \mathit{bbox2} \\
\quad \mathit{top} = (\mathit{Comp}\ \mathit{Ve}\ (\mathit{Comp}\ \mathit{Hb}\ \mathit{bbox1}\ \mathit{gbox1})\ (\mathit{Comp}\ \mathit{Hb}\ \mathit{rbox1}\ (\mathit{Comp}\ H\ \mathit{ybox1}\ \mathit{rbox2})))
\end{array}
$$

A seguinte função cria uma caixa a partir dos seguintes parâmetros: origem, largura, altura, etiqueta e côr de preenchimento.

$$
\begin{array}{l}
\mathit{crCaixa} :: \mathit{Origem} \to \mathit{Float} \to \mathit{Float} \to \mathit{String} \to G.\mathit{Color} \to G.\mathit{Picture} \\
\mathit{crCaixa}\ (x, y)\ w\ h\ l\ c = G.\mathit{Translate}\ (x + (w\ /\ 2))\ (y + (h\ /\ 2))\ \$\ G.\mathit{pictures}\ [\mathit{caixa}, \mathit{etiqueta}]\ \mathbf{where} \\
\quad \mathit{caixa} = G.\mathit{color}\ c\ (G.\mathit{rectangleSolid}\ w\ h) \\
\quad \mathit{etiqueta} = G.\mathit{translate}\ \mathit{calc\_trans\_x}\ \mathit{calc\_trans\_y}\ \$ \\
\qquad G.\mathit{Scale}\ \mathit{calc\_scale}\ \mathit{calc\_scale}\ \$\ G.\mathit{color}\ G.\mathit{black}\ \$\ G.\mathit{Text}\ l \\
\quad \mathit{calc\_trans\_x} = (-((\mathit{fromIntegral}\ (\mathsf{length}\ \ l)) * \mathit{calc\_scale})\ /\ 2) * \mathit{base\_shift\_x} \\
\quad \mathit{calc\_trans\_y} = (-\mathit{calc\_scale}\ /\ 2) * \mathit{base\_shift\_y} \\
\quad \mathit{calc\_scale} = \mathit{bscale} * (\mathit{min}\ h\ w) \\
\quad \mathit{bscale} = 1\ /\ 700
\end{array}
$$

$$base\_shift\_y = 100$$
$$base\_shift\_x = 64$$

Função para visualizar resultados gráficos:

$$display = G.display\ (G.InWindow\ \texttt{"Problema 4"}\ (400, 400)\ (40, 40))\ G.white$$

Problema 4

Funções para gestão de sistemas de ficheiros:

$$
\begin{aligned}
& concatFS = inFS \cdot \widehat{(+\!\!+)} \cdot (outFS \times outFS) \\
& mkdir\ (x, y) = FS\ [(x, Dir\ y)] \\
& mkfile\ (x, y) = FS\ [(x, File\ y)] \\
& joinDupDirs :: (Eq\ a) \Rightarrow (FS\ a\ b) \rightarrow (FS\ a\ b) \\
& joinDupDirs = anaFS\ (prepOut \cdot (id \times proc) \cdot prepIn)\ \textbf{where} \\
& \quad prepIn = (id \times (\mathsf{map}\ \ (id \times outFS))) \cdot sls \cdot (\mathsf{map}\ \ distr) \cdot outFS \\
& \quad prepOut = (\mathsf{map}\ \ undistr) \cdot \widehat{(+\!\!+)} \cdot ((\mathsf{map}\ \ i_1) \times (\mathsf{map}\ \ i_2)) \cdot (id \times (\mathsf{map}\ \ (id \times inFS))) \\
& \quad proc = concat \cdot (\mathsf{map}\ \ joinDup) \cdot groupByName \\
& \quad sls = \langle lefts, rights \rangle \\
& joinDup :: [(a, [b])] \rightarrow [(a, [b])] \\
& joinDup = cataList\ [nil, g]\ \textbf{where}\ g = return \cdot \langle \pi_1 \cdot \pi_1, concat \cdot (\mathsf{map}\ \ \pi_2) \cdot \widehat{(:)} \rangle \\
& createFSfromFile :: (Path\ a, b) \rightarrow (FS\ a\ b) \\
& createFSfromFile\ ([a], b) = mkfile\ (a, b) \\
& createFSfromFile\ (a : as, b) = mkdir\ (a, createFSfromFile\ (as, b))
\end{aligned}
$$

Funções auxiliares:

$$
\begin{aligned}
& checkFiles :: (Eq\ a) \Rightarrow FS\ a\ b \rightarrow Bool \\
& checkFiles = cataFS\ (\widehat{(\wedge)} \cdot \langle f, g \rangle)\ \textbf{where} \\
& \quad f = nr \cdot (\mathsf{fmap}\ \pi_1) \cdot lefts \cdot (\mathsf{fmap}\ distr) \\
& \quad g = and \cdot rights \cdot (\mathsf{fmap}\ \pi_2) \\
& groupByName :: (Eq\ a) \Rightarrow [(a, [b])] \rightarrow [[(a, [b])]] \\
& groupByName = (groupBy\ (curry\ p))\ \textbf{where} \\
& \quad p = \widehat{(\equiv)} \cdot (\pi_1 \times \pi_1) \\
& filterPath :: (Eq\ a) \Rightarrow Path\ a \rightarrow [(Path\ a, b)] \rightarrow [(Path\ a, b)] \\
& filterPath = filter \cdot (\lambda p \rightarrow \lambda(a, b) \rightarrow p \equiv a)
\end{aligned}
$$

Dados para testes:

- Sistema de ficheiros vazio:

$$efs = FS\ [\,]$$

- Nível 0

$$
\begin{aligned}
& f1 = FS\ [(\texttt{"f1"}, File\ \texttt{"hello world"})] \\
& f2 = FS\ [(\texttt{"f2"}, File\ \texttt{"more content"})] \\
& f00 = concatFS\ (f1, f2) \\
& f01 = concatFS\ (f1, mkdir\ (\texttt{"d1"}, efs)) \\
& f02 = mkdir\ (\texttt{"d1"}, efs)
\end{aligned}
$$

- Nível 1

$$
\begin{aligned}
& f10 = mkdir\ (\texttt{"d1"}, f00) \\
& f11 = concatFS\ (mkdir\ (\texttt{"d1"}, f00), mkdir\ (\texttt{"d2"}, f00)) \\
& f12 = concatFS\ (mkdir\ (\texttt{"d1"}, f00), mkdir\ (\texttt{"d2"}, f01)) \\
& f13 = concatFS\ (mkdir\ (\texttt{"d1"}, f00), mkdir\ (\texttt{"d2"}, efs))
\end{aligned}
$$

- Nível 2

$$
\begin{aligned}
&f20 = mkdir\ (\texttt{"d1"}, f10)\\
&f21 = mkdir\ (\texttt{"d1"}, f11)\\
&f22 = mkdir\ (\texttt{"d1"}, f12)\\
&f23 = mkdir\ (\texttt{"d1"}, f13)\\
&f24 = concatFS\ (mkdir\ (\texttt{"d1"}, f10), mkdir\ (\texttt{"d2"}, f12))
\end{aligned}
$$

- Sistemas de ficheiros inválidos:

$$
\begin{aligned}
&ifs0 = concatFS\ (f1, f1)\\
&ifs1 = concatFS\ (f1, mkdir\ (\texttt{"f1"}, efs))\\
&ifs2 = mkdir\ (\texttt{"d1"}, ifs0)\\
&ifs3 = mkdir\ (\texttt{"d1"}, ifs1)\\
&ifs4 = concatFS\ (mkdir\ (\texttt{"d1"}, ifs1), mkdir\ (\texttt{"d2"}, f12))\\
&ifs5 = concatFS\ (mkdir\ (\texttt{"d1"}, f1), mkdir\ (\texttt{"d1"}, f2))\\
&ifs6 = mkdir\ (\texttt{"d1"}, ifs5)\\
&ifs7 = concatFS\ (mkdir\ (\texttt{"d1"}, f02), mkdir\ (\texttt{"d1"}, f02))
\end{aligned}
$$

Visualização em Graphviz:

$$
\begin{aligned}
&dotFS :: FS\ String\ b \to \mathsf{IO}\ ExitCode\\
&dotFS = dotpict \cdot bmap\ \underline{\texttt{""}}\ id \cdot (cFS2Exp\ \texttt{"root"})
\end{aligned}
$$

Outras funções auxiliares

Lógicas:

$$
\begin{aligned}
&\textbf{infixr}\ 0 \Rightarrow\\
&(\Rightarrow) :: (Testable\ prop) \Rightarrow (a \to Bool) \to (a \to prop) \to a \to Property\\
&p \Rightarrow f = \lambda a \to p\ a \Rightarrow f\ a\\
&\textbf{infixr}\ 0 \Leftrightarrow\\
&(\Leftrightarrow) :: (a \to Bool) \to (a \to Bool) \to a \to Property\\
&p \Leftrightarrow f = \lambda a \to (p\ a \Rightarrow property\ (f\ a))\ .\&\&.\ (f\ a \Rightarrow property\ (p\ a))\\
&\textbf{infixr}\ 4 \equiv\\
&(\equiv) :: Eq\ b \Rightarrow (a \to b) \to (a \to b) \to (a \to Bool)\\
&f \equiv g = \lambda a \to f\ a \equiv g\ a\\
&\textbf{infixr}\ 4 \leqslant\\
&(\leqslant) :: Ord\ b \Rightarrow (a \to b) \to (a \to b) \to (a \to Bool)\\
&f \leqslant g = \lambda a \to f\ a \leqslant g\ a\\
&\textbf{infixr}\ 4 \land\\
&(\land) :: (a \to Bool) \to (a \to Bool) \to (a \to Bool)\\
&f \land g = \lambda a \to ((f\ a) \land (g\ a))
\end{aligned}
$$

Compilação e execução dentro do interpretador:[8]

$$run = \textbf{do}\ \{system\ \texttt{"ghc cp1819t"}; system\ \texttt{"./cp1819t"}\}$$

D Soluções dos alunos

Os alunos devem colocar neste anexo as suas soluções aos 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 e/ou outras funções auxiliares que sejam necessárias.

[8]Pode ser útil em testes envolvendo Gloss. Nesse caso, o teste em causa deve fazer parte de uma função *main*.

Problema 1

Definição do tipo de dados

$$
\begin{array}{l}
inExpr :: Int + (Op, (Expr, Expr)) \to Expr \\
inExpr = [Num, \widehat{Bop} \cdot (swap \times id) \cdot assocl] \\
outExpr :: Expr \to Int + (Op, (Expr, Expr)) \\
outExpr \ (Num \ a) = i_1 \ a \\
outExpr \ (Bop \ e_1 \ o \ e_2) = i_2 \ (o, (e_1, e_2))
\end{array}
$$

Padrões de recursividade

$$
\begin{array}{l}
recExpr \ f = baseExpr \ id \ f \\
cataExpr \ g = g \cdot (recExpr \ (cataExpr \ g)) \cdot outExpr \\
anaExpr \ f = inExpr \cdot (recExpr \ (anaExpr \ f)) \cdot f \\
hyloExpr \ a \ c = cataExpr \ a \cdot anaExpr \ c
\end{array}
$$

calcula

$$
\begin{tikzcd}
Expr \arrow[d, "calcula"'] & Int + (Op \times (Expr \times Expr)) \arrow[l, "\mathsf{in}"'] \arrow[d, "id + (id \times (calcula \times calcula))"] \\
Int & Int + (Op \times (Int \times Int)) \arrow[l, "{[id, aux]}"]
\end{tikzcd}
$$

$$
\begin{array}{l}
calcula :: Expr \to Int \\
calcula = cataExpr \ [id, aux] \\
\quad \mathbf{where} \ aux :: (Op, (Int, Int)) \to Int \\
\qquad aux \ (o, (int1, int2)) = calculaOP \ o \ int1 \ int2 \\
calculaOP :: Op \to Int \to Int \to Int \\
calculaOP \ (Op \ \texttt{"+"}) \ int1 \ int2 = (+) \ int1 \ int2 \\
calculaOP \ (Op \ \texttt{"-"}) \ int1 \ int2 = (-) \ int1 \ int2 \\
calculaOP \ (Op \ \texttt{"*"}) \ int1 \ int2 = (*) \ int1 \ int2 \\
calculaOP \ (Op \ \texttt{"/"}) \ int1 \ int2 = int1 \div int2
\end{array}
$$

compile

$$
\begin{tikzcd}
String \arrow[r, "divide"] \arrow[d, "{[\!(divide)\!]}"'] & Int + (Op \times (String \times String)) \arrow[d, "id + (id \times ([\!(divide)\!] \times [\!(divide)\!]))"] \\
Expr \arrow[d, "(\!| conquer |\!)"'] & Int + (Op \times (Expr \times Expr)) \arrow[l, "\mathsf{in}"'] \arrow[d, "id + (id \times ((\!| conquer |\!) \times (\!| conquer |\!)))"] \\
Codigo & Int + (Op \times (Codigo \times Codigo)) \arrow[l, "conquer"]
\end{tikzcd}
$$

```
compile :: String → Codigo
compile = hyloExpr conquer divide

divide :: String → Int + (Op, (String, String))
divide s = if (length s) ≡ 1 then i₁ (read s :: Int)
    else i₂ (testaExpr s)

testaExpr :: String → (Op, (String, String))
testaExpr x = if isNothing res then (id × (filterParenteses × filterParenteses))
  $ aux $ span (λx → x ≢ '+' ∧ x ≢ '-' ∧ x ≢ '*' ∧ x ≢ '/') x else fromJust res
    where res = testaPadrao (x, [])
      aux (a, b) = (Op [head b], (a, tail b))
      filterParenteses = filter (λx → x ≢ '(' ∧ x ≢ ')')

testaPadrao :: (String, String) → Maybe (Op, (String, String))
testaPadrao ([], _) = Nothing
testaPadrao ([x], _) = Nothing
testaPadrao ((c1 : c2 : s), a1) | c1 ≡ ')' ∧ c2 ≡ '*' = Just (Op "*", ((a1 ++ [c1]), s))
  | c1 ≡ ')' ∧ c2 ≡ '+' = Just (Op "+", ((a1 ++ [c1]), s))
  | c1 ≡ ')' ∧ c2 ≡ '-' = Just (Op "-", ((a1 ++ [c1]), s))
  | c1 ≡ ')' ∧ c2 ≡ '/' = Just (Op "/", ((a1 ++ [c1]), s))
  | c1 ≡ '*' ∧ c2 ≡ '(' = Just (Op "*", (a1, [c2] ++ s))
  | c1 ≡ '+' ∧ c2 ≡ '(' = Just (Op "+", (a1, [c2] ++ s))
  | c1 ≡ '/' ∧ c2 ≡ '(' = Just (Op "/", (a1, [c2] ++ s))
  | c1 ≡ '-' ∧ c2 ≡ '(' = Just (Op "-", (a1, [c2] ++ s))
  | otherwise = testaPadrao ((c2 : s), (a1 ++ [c1]))

conquer :: Int + (Op, (Codigo, Codigo)) → Codigo
conquer = [auxConq, auxConquer]

auxConq :: Int → Codigo
auxConq i = ["PUSH " ++ (show i)]

auxConquer :: (Op, (Codigo, Codigo)) → Codigo
auxConquer ((Op "+"), (c1, c2)) = c1 ++ c2 ++ ["ADD"]
auxConquer ((Op "-"), (c1, c2)) = c1 ++ c2 ++ ["SUB"]
auxConquer ((Op "*"), (c1, c2)) = c1 ++ c2 ++ ["MUL"]
auxConquer ((Op "/"), (c1, c2)) = c1 ++ c2 ++ ["DIV"]
```

show′

$$\begin{tikzcd} Expr \arrow[d, "show'"'] & Int + (Op \times (Expr \times Expr)) \arrow[l, "\mathsf{in}"'] \arrow[d, "id+(id\times(show'\times show'))"] \\ String & Int + (Op \times (String \times String)) \arrow[l, "{[showAux, auxShow]}"] \end{tikzcd}$$

```
show′ :: Expr → String
show′ = cataExpr [showAux, auxShow]
  where auxShow :: (Op, (String, String)) → String
    auxShow (o, (s1, s2)) = calculaString o s1 s2

showAux :: Int → String
showAux a = show a

calculaString :: Op → String → String → String
calculaString (Op "+") s1 s2 = "(" ++ s1 ++ " + " ++ s2 ++ ")"
calculaString (Op "-") s1 s2 = "(" ++ s1 ++ " - " ++ s2 ++ ")"
calculaString (Op "*") s1 s2 = "(" ++ s1 ++ " * " ++ s2 ++ ")"
calculaString (Op "/") s1 s2 = "(" ++ s1 ++ " / " ++ s2 ++ ")"
```

Problema 2

Definição do tipo de dados

$$
\begin{array}{l}
inL2D :: a + (b, (X\ a\ b, X\ a\ b)) \to X\ a\ b \\
inL2D = [Unid, \widehat{Comp} \cdot assocl] \\
outL2D :: X\ a\ b \to a + (b, (X\ a\ b, X\ a\ b)) \\
outL2D\ (Unid\ a) = i_1\ a \\
outL2D\ (Comp\ b\ a1\ a2) = i_2\ (b, (a1, a2))
\end{array}
$$

Padrões de recursividade

$$
\begin{array}{l}
recL2D\ h = baseL2D\ id\ id\ h \\
cataL2D\ g = g \cdot (recL2D\ (cataL2D\ g)) \cdot outL2D \\
anaL2D\ g = inL2D \cdot (recL2D\ (anaL2D\ g)) \cdot g \\
baseL2D\ f\ g\ h = f + g \times (h \times h)
\end{array}
$$

collectLeafs

$$
\begin{tikzcd}
X\ A\ B \arrow[d, "collectLeafs"'] & A + (B \times (X\ A\ B \times X\ A\ B)) \arrow[l, "\mathsf{in}"'] \arrow[d, "id + (id \times (collectLeafs \times collectLeafs))"] \\
A^* & A + (B \times (A^* \times A^*)) \arrow[l, "{[singl, juntaListas \cdot \pi_2]}"]
\end{tikzcd}
$$

$$
\begin{array}{l}
collectLeafs :: X\ a\ b \to [a] \\
collectLeafs = cataL2D\ collectLeafsAux \\
collectLeafsAux :: a + (b, ([a], [a])) \to [a] \\
collectLeafsAux = [singl, juntaListas \cdot \pi_2] \\
juntaListas :: ([a], [a]) \to [a] \\
juntaListas\ (l1, l2) = l1 + l2
\end{array}
$$

dimen

$$
\begin{tikzcd}
X\ Caixa\ Tipo \arrow[d, "dimen"'] & Caixa + (Tipo \times (X\ Caixa\ Tipo \times X\ Caixa\ Tipo)) \arrow[l, "\mathsf{in}"'] \arrow[d, "id + (id \times (dimen \times dimen))"] \\
(Float, Float) & Caixa + (Tipo \times ((Float, Float) \times (Float, Float))) \arrow[l, "{[dimenCaixa, auxCata]}"]
\end{tikzcd}
$$

$$
\begin{array}{l}
dimen :: X\ Caixa\ Tipo \to (Float, Float) \\
dimen = cataL2D\ dimenAux \\
dimenAux :: Caixa + (Tipo, ((Float, Float), (Float, Float))) \to (Float, Float) \\
dimenAux = [dimenCaixa, auxCata] \\
dimenCaixa :: Caixa \to (Float, Float) \\
dimenCaixa\ ((int1, int2), (t, c)) = (fromIntegral\ int1, fromIntegral\ int2) \\
auxCata :: (Tipo, ((Float, Float), (Float, Float))) \to (Float, Float) \\
auxCata\ (V, ((f1, f2), (f3, f4))) = \mathbf{if}\ (f3 > f1)\ \mathbf{then}\ (f3, f2 + f4)\ \mathbf{else}\ (f1, f2 + f4) \\
auxCata\ (Vd, ((f1, f2), (f3, f4))) = \mathbf{if}\ (f3 > f1)\ \mathbf{then}\ (f3, f2 + f4)\ \mathbf{else}\ (f1, f2 + f4) \\
auxCata\ (Ve, ((f1, f2), (f3, f4))) = \mathbf{if}\ (f3 > f1)\ \mathbf{then}\ (f3, f2 + f4)\ \mathbf{else}\ (f1, f2 + f4) \\
auxCata\ (H, ((f1, f2), (f3, f4))) = \mathbf{if}\ (f4 > f2)\ \mathbf{then}\ (f1 + f3, f4)\ \mathbf{else}\ (f1 + f3, f2) \\
auxCata\ (Hb, ((f1, f2), (f3, f4))) = \mathbf{if}\ (f4 > f2)\ \mathbf{then}\ (f1 + f3, f4)\ \mathbf{else}\ (f1 + f3, f2) \\
auxCata\ (Ht, ((f1, f2), (f3, f4))) = \mathbf{if}\ (f4 > f2)\ \mathbf{then}\ (f1 + f3, f4)\ \mathbf{else}\ (f1 + f3, f2)
\end{array}
$$

calcOrigins

$$\begin{tikzcd}
X\ (Caixa \times Origem)\ 1 & (Caixa \times Origem) + (1 \times (X\ (Caixa \times Origem)\ 1 \times X\ (Caixa \times Origem)\ 1)) \arrow[l, "\mathsf{in}"'] \\
X\ Caixa\ Tipo \times Origem \arrow[u, "calcOrigins"] \arrow[r] & (Caixa \times Origem) + (1 \times ((X\ Caixa\ Tipo \times Origem) \times (X\ Caixa\ Tipo \times Origem))) \arrow[u, "id+(id\times(calcOrigins\times calcOrigins))"']
\end{tikzcd}$$

$calcOrigins :: ((X\ Caixa\ Tipo), Origem) \rightarrow X\ (Caixa, Origem)\ ()$
$calcOrigins = anaL2D\ calcOriginsAux$

$calcOriginsAux :: (X\ Caixa\ Tipo, Origem) \rightarrow (Caixa, Origem) + \cdot$
$\quad ((), (((X\ Caixa\ Tipo), Origem), ((X\ Caixa\ Tipo), Origem)))$
$calcOriginsAux\ ((Unid\ a), o) = i_1\ (a, o)$
$calcOriginsAux\ (Comp\ t\ x1\ x2, o) = \mathbf{case}\ t\ \mathbf{of}$
$\quad V \rightarrow i_2\ ((!)\ o, ((x1, o), (x2, (\pi_1\ o + (\pi_1\ (dimen\ x1))\ /\ 2, \pi_2\ (dimen\ x1) + \pi_2\ o))))$
$\quad Vd \rightarrow i_2\ ((!)\ o, ((x1, o), (x2, (\pi_1\ o + \pi_1\ (dimen\ x1) - \pi_1\ (dimen\ x2), \pi_2\ (dimen\ x1) + \pi_2\ o))))$
$\quad Ve \rightarrow i_2\ ((!)\ o, ((x1, o), (x2, (\pi_1\ o, \pi_2\ (dimen\ x1) + \pi_2\ o))))$
$\quad H \rightarrow i_2\ ((!)\ o, ((x1, o), (x2, (\pi_1\ (dimen\ x1) + \pi_1\ o, \pi_2\ o + (\pi_2\ (dimen\ x1))\ /\ 2))))$
$\quad Hb \rightarrow i_2\ ((!)\ o, ((x1, o), (x2, (\pi_1\ (dimen\ x1) + \pi_1\ o, \pi_2\ o))))$
$\quad Ht \rightarrow i_2\ ((!)\ o, ((x1, o), (x2, (\pi_1\ (dimen\ x1) + \pi_1\ o, \pi_2\ (dimen\ x1) - \pi_2\ (dimen\ x2) + \pi_2\ o))))$

agrupcaixas

$$\begin{tikzcd}
X\ (Caixa \times Origem)\ 1 \arrow[d, "agrupcaixas"'] & (Caixa \times Origem) + (1 \times (X\ (Caixa \times Origem)\ 1 \times X\ (Caixa \times Origem)\ 1)) \arrow[l, "\mathsf{in}"'] \arrow[d, "id+(id\times(agrupcaixas\times agrupcaixas))"] \\
Fig & (Caixa \times Origem) + (1 \times (Fig \times Fig)) \arrow[l, "agrupcaixasAux"]
\end{tikzcd}$$

$agrupcaixas :: X\ (Caixa, Origem)\ () \rightarrow Fig$
$agrupcaixas = cataL2D\ agrupcaixasAux$

$agrupcaixasAux :: (Caixa, Origem) + ((), (Fig, Fig)) \rightarrow Fig$
$agrupcaixasAux\ (i_1\ (c, o)) = [(o, c)]$
$agrupcaixasAux\ (i_2\ ((), (f1, f2))) = f1 +\!\!+ f2$

$constroiFig :: Fig \rightarrow [G.Picture]$
$constroiFig\ [\,] = [\,]$
$constroiFig\ ((o, ((int1, int2), (t, c))) : xs) = (crCaixa\ o\ (fromIntegral\ int1)\ (fromIntegral\ int2)\ t\ c)$
$\quad : constroiFig\ xs$

$mostracaixas :: (L2D, Origem) \rightarrow \mathsf{IO}\ ()$
$mostracaixas\ (l, o) = display\ (caixasAndOrigin2Pict\ (l, o))$

$caixasAndOrigin2Pict :: (X\ Caixa\ Tipo, Origem) \rightarrow G.Picture$
$caixasAndOrigin2Pict\ (x, o) = G.pictures\ (constroiFig\ (agrupcaixas\ (calcOrigins\ (x, o))))$

$calc :: Tipo \rightarrow Origem \rightarrow (Float, Float) \rightarrow Origem$
$calc\ t\ o\ (f1, f2) = \mathbf{case}\ t\ \mathbf{of}$
$\quad V \rightarrow (\pi_1\ o + f1\ /\ 2, \pi_2\ o + f2)$
$\quad Ve \rightarrow (\pi_1\ o, \pi_2\ o + f2)$
$\quad Vd \rightarrow (\pi_1\ o + f1, \pi_2\ o + f2)$
$\quad H \rightarrow (\pi_1\ o + f1, \pi_2\ o + f2\ /\ 2)$
$\quad Ht \rightarrow (\pi_1\ o + f1, \pi_2\ o + f2)$
$\quad Hb \rightarrow (\pi_1\ o + f1, \pi_2\ o)$

Problema 3

Solução:

Neste problema pretende-se implementar um ciclo que implemente o cálculo da aproximação até $i = n$ da função cosseno $cos\ x$ via série de Taylor:

$$cos\ x = \sum_{i=0}^{\infty} \frac{(-1)^i}{(2i)!} x^{2i}$$

Seja $c\ x\ n = \sum_{i=0}^{n} \frac{(-1)^i}{(2i)!} x^{2i}$ a função que dá essa aproximação. É fácil de ver que $c\ x\ 0 = 1$ e que $c\ x\ (n+1) = c\ x\ n + \frac{(-1)^{n+1}}{(2(n+1))!} x^{2(n+1)}$. Se definirmos $h\ x\ n = \frac{(-1)^{n+1}}{(2(n+1))!} x^{2(n+1)}$ teremos $c\ x$ e $h\ x$ em recursividade mútua. Repetindo o processo para $h\ x\ n$ é fácil de ver que $h\ x\ 0 = \frac{-x^2}{2}$ e que $h\ x\ (n+1) = h\ x\ n * \frac{-x^2}{4n^2+14n+12}$ e, portanto, podemos definir $s\ n = 4n^2 + 14n + 12$. Por último, e repetindo, mais uma vez, o processo para $s\ n$ é fácil de ver que $s\ 0 = 12$ e que $s\ (n+1) = s\ n + 8n + 18$ e, portanto, podemos definir $k\ n = 8n + 18$, verificando-se $k\ 0 = 18$ e $k\ (n+1) = k\ n + 8$. Daqui, obtemos no total quatro funções em recursividade mútua:

$$\begin{array}{l} c\ x\ 0 = 1 \\ c\ x\ (n+1) = c\ x\ n + h\ x\ n \end{array}$$
$$\begin{array}{l} h\ x\ 0 = -x^2\ /\ 2 \\ h\ x\ (n+1) = -x^2\ /\ (s\ n) * h\ x\ n \end{array}$$
$$\begin{array}{l} s\ 0 = 12 \\ s\ (n+1) = k\ n + s\ n \end{array}$$
$$\begin{array}{l} k\ 0 = 18 \\ k\ (n+1) = k\ n + 8 \end{array}$$

Segundo a *regra de algibeira* descrita na página 3 deste enunciado, ter-se-á, de imediato:

$$\begin{array}{l} cos'\ x = prj \cdot \mathsf{for}\ loop\ init\ \mathbf{where} \\ \quad loop\ (c,h,s,k) = (c+h, -x \uparrow 2\ /\ s * h, s+k, k+8) \\ \quad init = (1, -x \uparrow 2\ /\ 2, 12, 18) \\ \quad prj\ (c,h,s,k) = c \end{array}$$

Problema 4

Definição do tipo de dados

$$outFS\ (FS\ l) = \mathsf{map}\ \ (id \times outNode)\ l$$
$$\begin{array}{l} outNode :: Node\ a\ b \rightarrow b + (FS\ a\ b) \\ outNode\ (File\ b) = i_1\ b \\ outNode\ (Dir\ (FS\ a)) = i_2\ (FS\ a) \end{array}$$

Padrões de recursividade

$$baseFS\ f\ g\ h = \mathsf{map}\ \ (f \times (g+h))$$
$$\begin{array}{l} cataFS :: ([(a, b+c)] \rightarrow c) \rightarrow FS\ a\ b \rightarrow c \\ cataFS\ g = g \cdot (recFS\ (cataFS\ g)) \cdot outFS \end{array}$$
$$\begin{array}{l} anaFS :: (c \rightarrow [(a, b+c)]) \rightarrow c \rightarrow FS\ a\ b \\ anaFS\ g = inFS \cdot (recFS\ (anaFS\ g)) \cdot g \end{array}$$
$$hyloFS\ g\ h = (cataFS\ g) \cdot (anaFS\ h)$$

cataFS

$$\begin{tikzcd}
FS\ A\ B \arrow[d, "(\!|g|\!)"'] & (A \times (B + FS\ A\ B))^* \arrow[l, "\mathsf{in}"'] \arrow[d, "\mathsf{map}\ (id \times (id + (\!|g|\!)))"] \\
C & (A \times (B + C))^* \arrow[l, "g"]
\end{tikzcd}$$

Outras funções pedidas:

check

Em primeiro lugar aplicamos a função idsToList e de seguida a função isUnique que passamos a explicar de seguida. Primeiramente, vamos definir um *anamorfismo* de listas que gera uma lista de pares cujo primeiro elemento é um *booleano* que indica se o elemento é único e o segundo é o elemento em si. De seguida definimos um *catamorfismo* que consome os booleanos da lista.

$$\begin{tikzcd}
A^* \arrow[r, "listBool"] \arrow[d, "[\!(listBool)\!]"'] & 1 + Bool \times A\ list \arrow[d, "id + id \times [\!(listBool)\!]"] \\
(Bool \times A)^* \arrow[d, "(\!|consumeBool|\!)"'] & 1 + (Bool \times A) \times (Bool \times A)^* \arrow[l, "\mathsf{in}"'] \arrow[d, "id + id \times (\!|consumBool|\!)"] \\
Bool & 1 + (Bool \times A) \times Bool^* \arrow[l, "consumeBool"]
\end{tikzcd}$$

$check :: (Eq\ a) \Rightarrow FS\ a\ b \rightarrow Bool$
$check = isUnique \cdot idsToList$

$idsToList :: FS\ a\ b \rightarrow [a]$

$idsToList = (cataList\ [nil, \widehat{(+\!\!+)} \cdot ((inList \cdot i_2 \cdot (id \times [nil, idsToList]) \times id))]) \cdot outFS$

$isUnique :: (Eq\ a) \Rightarrow [a] \rightarrow Bool$
$isUnique = consumeBool \cdot listBool$ **where**
  $listBool = anaList\ ((id + \langle\langle \widehat{notElem}, \pi_1\rangle, \pi_2\rangle) \cdot outList)$
  $consumeBool = cataList\ [\underline{True}, \widehat{(\wedge)} \cdot (\pi_1 \times id)]$

tar

Definimos a função tar como um catamorfismo cujo gene é (concat . map (auxTar.(id x [auxTar',id]))), tal como pode ser visto no diagrama abaixo.

$$\begin{tikzcd}
FS\ A\ B \arrow[d, "tar"'] & (A \times (B + FS\ A\ B))^* \arrow[l, "\mathsf{in}"'] \arrow[d, "\mathsf{map}\ (id \times (id + tar))"] \\
(Path\ A \times B)^* & (A \times (B + (Path\ A \times B)^*))^* \arrow[l, "(concat \cdot \mathsf{map}\ (auxTar \cdot (id \times [auxTar', id])))"]
\end{tikzcd}$$

$tar :: FS\ a\ b \rightarrow [(Path\ a, b)]$
$tar = cataFS\ (concat \cdot \mathsf{map}\ (auxTar \cdot (id \times [auxTar', id])))$
$auxTar :: (a, [([a], b)]) \rightarrow [([a], b)]$

$auxTar = ((\lambda(x, y) \rightarrow (cataList\ [nil, \lambda((w, v), l) \rightarrow ([x] +\!\!+\ w, v) : l]\ y)))$
$auxTar' :: b \rightarrow [([a], b)]$
$auxTar' = singl \cdot (\lambda b \rightarrow ([\,], b))$

untar

Definimos a função untar como um anamorfismo cujo gene é a função untarAux. Esta última função irá colocar a cabeça do Path a fornecido do lado esquerdo de [(a,Either b [([a],b)])]. Quanto ao lado direito, se o Path a fornecido tiver um único elemento, então apenas injetamos do lado esquerdo o b fornecido. Caso contrário, injetamos do lado direito a lista de pares formados pela cauda do Path A fornecido e pelo b fornecido. É importante mencionar que no fim aplicamos a função joinDupDirs para juntar directorias que estejam na mesma pasta e que possuam o mesmo identificador.

$$
\begin{tikzcd}
FS\ A\ B & & (A \times (B + FS\ A\ B))^{*} \arrow[ll, "\mathsf{in}"'] \\
(Path\ A \times B)^{*} \arrow[u, "untar"] \arrow[rr, "untarAux"'] & & (A \times (B + (Path\ A \times B)^{*}))^{*} \arrow[u, "\mathsf{map}\ (id \times (id + untar))"']
\end{tikzcd}
$$

$untar :: (Eq\ a) \Rightarrow [(Path\ a, b)] \rightarrow FS\ a\ b$
$untar = joinDupDirs \cdot (anaFS\ untarAux)$
$untarAux :: (Eq\ a) \Rightarrow [([a], b)] \rightarrow [(a, b + [([a], b)])]$
$untarAux = \mathsf{map}\ \langle head \cdot \pi_1, g\rangle$
$\quad \mathbf{where}\ g = (\lambda((x : xs), f) \rightarrow \mathbf{if}\ (null\ xs)\ \mathbf{then}\ i_1\ f$
$\quad\quad \mathbf{else}\ i_2\ [(xs, f)])$

find

Definimos a função find como um catamorfismo cujo gene é a função g como pode ser visto no diagrama abaixo.

$$
\begin{tikzcd}
FS\ A\ B \arrow[d, "find\ a"'] & & (A \times (B + FS\ A\ B))^{*} \arrow[ll, "\mathsf{in}"'] \arrow[d, "\mathsf{map}\ (id \times (id + find\ a))"] \\
(Path\ A)^{*} & & (A \times (B + (Path\ A)^{*}))^{*} \arrow[ll, "g"]
\end{tikzcd}
$$

$find :: (Eq\ a) \Rightarrow a \rightarrow FS\ a\ b \rightarrow [Path\ a]$
$find = cataFS \cdot concatMap \cdot g$
$\quad \mathbf{where}\ f :: a \rightarrow [Path\ a] + [Path\ a] \rightarrow [Path\ a]$
$\quad\quad f\ a = [[a]:, \mathsf{map}\ (a:)]$
$\quad\quad g :: Eq\ a \Rightarrow a \rightarrow (a, b + [Path\ a]) \rightarrow [Path\ a]$
$\quad\quad g\ a = \widehat{f} \cdot (id \times (nil + filter\ ((\equiv a) \cdot last)))$

new

Em primeiro lugar, a função new aplica a função tar, devolvendo assim um [(Path a, b)]. De seguida, é aplicada a função g que junta o Path a e o b fornecidos com o resultado da função tar num par, ou seja, ((Path a, b),[(Path a, b)]) e, posteriormente, aplica cons de forma a que o que está à esquerda do par fique à cabeça da lista à direita. Por último, é aplicada a função untar para devolver o FS a b. Relativamente à questão colocada nesta alínea a resposta é depende, visto que se a função check aceder às subdiretorias então sim, a propriedade prop_new ainda é válida, caso contrário é inválida.

$$
\begin{array}{l}
new :: (Eq\ a) \Rightarrow Path\ a \rightarrow b \rightarrow FS\ a\ b \rightarrow FS\ a\ b \\
new\ a\ b = untar \cdot g\ (a, b) \cdot tar \\
\quad \mathbf{where} \\
\qquad g = (cons \cdot) \cdot (,)
\end{array}
$$

cp

Para resolver esta alínea iremos aplicar um anamorfismo após outro anamorfismo (neste caso, auxAna2 após auxAna1). Em primeiro lugar, achamos relevante explicar que o anamorfismo ana1 será aplicado ao par formado pelo primeiro Path a fornecido e pelo FS a b fornecido, devolvendo então um FS a b. De seguida, o anamorfismo auxAna2 será aplicado ao FS a b resultante e ao par formado pelo segundo Path a fornecido e pelo FS a b fornecido, devolvendo assim o FS a b desejado.

$$
\begin{tikzcd}
FS\ A\ B & (A \times (B + FS\ A\ B))^{*} \arrow[l, "\mathsf{in}"'] \\
(Path\ A \times FS\ A\ B) \arrow[u, "ana1"] \arrow[r, "auxaux"'] & (A \times (B + (Path\ A \times FS\ A\ B)))^{*} \arrow[u, "\mathsf{map}\ (id \times (id + ana1))"']
\end{tikzcd}
$$

$$
\begin{array}{l}
cp :: (Eq\ a) \Rightarrow Path\ a \rightarrow Path\ a \rightarrow FS\ a\ b \rightarrow FS\ a\ b \\
cp\ from\ to\ f = \mathbf{let}\ nFS = ana1\ (from, f) \\
\quad \mathbf{in}\ auxAna2\ (nFS, (to, f)) \\
auxAna2 :: Eq\ a \Rightarrow (FS\ a\ b, (Path\ a, FS\ a\ b)) \rightarrow FS\ a\ b \\
auxAna2 = cond\ (null \cdot outFS \cdot \pi_1)\ (\pi_2 \cdot \pi_2)\ g\ \mathbf{where} \\
\quad g = cond\ (null \cdot \pi_1 \cdot \pi_2)\ (inFS \cdot \mathsf{conc} \cdot (outFS \times outFS \cdot \pi_2))\ f\ \mathbf{where} \\
\qquad f\ (ficheiro, ((x : xs), fs)) = inFS \cdot \mathsf{map}\ \ (\lambda(i, v) \rightarrow \mathbf{if}\ i \equiv x\ \mathbf{then}\ (i, inter\ ficheiro\ xs\ v)\ \mathbf{else}\ (i, v)) \\
\qquad\qquad \cdot\ outFS\ \$\ fs \\
\qquad inter :: Eq\ a \Rightarrow FS\ a\ b \rightarrow Path\ a \rightarrow b + (FS\ a\ b) \rightarrow b + (FS\ a\ b) \\
\qquad inter\ ficheiro\ l\ v = \mathbf{case}\ v\ \mathbf{of} \\
\qquad\quad (i_1\ b) \rightarrow i_1\ b \\
\qquad\quad (i_2\ a) \rightarrow i_2\ \$\ auxAna2\ (ficheiro, (l, a)) \\
ana1 :: Eq\ a \Rightarrow (Path\ a, FS\ a\ b) \rightarrow FS\ a\ b \\
ana1 = cond\ (null \cdot \pi_1)\ \underline{(FS\ [\,])}\ (anaFS\ auxaux) \\
auxaux :: Eq\ a \Rightarrow (Path\ a, FS\ a\ b) \rightarrow [(a, b + (Path\ a, FS\ a\ b))] \\
auxaux = cond\ (null \cdot \pi_1)\ ((\mathsf{map}\ \ (id \times (id + \langle nil, id \rangle))) \cdot outFS \cdot \pi_2)\ auxAna1 \\
auxAna1 :: Eq\ a \Rightarrow (Path\ a, FS\ a\ b) \rightarrow [(a, b + (Path\ a, FS\ a\ b))] \\
auxAna1 = cond\ ((\lambda(x : xs) \rightarrow null\ xs) \cdot \pi_1)\ f\ g \\
\quad \mathbf{where} \\
\qquad f\ (caminho, fs) = \mathsf{map}\ \ (id \times (id + \langle nil, id \rangle)) \cdot filter\ ((\equiv actual) \cdot \pi_1) \cdot outFS\ \$\ fs\ \mathbf{where} \\
\qquad\qquad actual = head\ caminho \\
\qquad g\ (caminho, fs) = auxaux \cdot (cond\ null\ \underline{([\,], FS\ [\,])}\ \langle \underline{resto}, head \rangle) \cdot \\
\qquad\quad \mathsf{map}\ \ ([\underline{(FS\ [\,])}, id] \cdot \pi_2) \cdot filter\ ((\equiv actual) \cdot \pi_1) \cdot outFS\ \$\ fs\ \mathbf{where} \\
\qquad\qquad actual = head\ caminho \\
\qquad\qquad resto = tail\ caminho
\end{array}
$$

rm

Em primeiro lugar, a função rm irá aplicar a função tar ao FS a b fornecido e, assim, irá devolver um [(Path a, b)]. De seguida, será aplicada a função rmAux ao Path a fornecido e ao [(Path a, b)] resultante. Esta função consiste em remover o primeiro par dessa lista cujo Path a seja igual ao Path a fornecido. Por último, é aplicada a função untar que devolve o FS a b.

$$
\begin{aligned}
&rm :: (Eq\ a) \Rightarrow (Path\ a) \rightarrow (FS\ a\ b) \rightarrow FS\ a\ b\\
&rm\ a\ b = untar\ (rmAux\ a\ (tar\ b))\\
&rmAux :: Eq\ a \Rightarrow Path\ a \rightarrow [(Path\ a, b)] \rightarrow [(Path\ a, b)]\\
&rmAux\ \_\ [\,] = [\,]\\
&rmAux\ a1\ ((a2, b) : xs) = \mathbf{if}\ (a1 \equiv a2)\ \mathbf{then}\ xs\ \mathbf{else}\ rmAux\ a1\ xs\\
&auxJoin :: ([(a, b + c)], d) \rightarrow [(a, b + (d, c))]\\
&auxJoin = (flip\ \widehat{(\mathsf{map}}\ \cdot g))\\
&\quad \mathbf{where}\\
&\qquad g\ d = id \times (id + (,)\ d)
\end{aligned}
$$

cFS2Exp

Definimos a função cFS2Exp como um anamorfismo cujo gene é cFS2ExpAux. Esta última transforma um par formado por a (sendo este a root) e um FS a b num par formado pelo mesmo a e o resultado de aplicar outFS (onde, neste caso, a irá representar as subdiretorias). De seguida, o b resultante de aplicar outFS será transformado num FS a b e, assim, poderá juntar-se ao outro FS a b. Por último, apenas será necessário aplicar a injeção i2.

$$
\begin{tikzcd}
Exp\ 1\ A & & 1 + (A \times (Exp\ 1\ A)^{*}) \arrow[ll, "\mathsf{in}"'] \\
A \times FS\ A\ B \arrow[u, "cFS2Exp"] \arrow[rr, "i_2 \cdot g \cdot h \cdot f"'] & & 1 + (A \times (A \times FS\ A\ B)^{*}) \arrow[u, "id + (id \times \mathsf{map}\ \ cFS2Exp)"]
\end{tikzcd}
$$

$$
\begin{aligned}
&cFS2Exp :: a \rightarrow FS\ a\ b \rightarrow (Exp\ ()\ a)\\
&cFS2Exp = curry\ (anaExp\ cFS2ExpAux)\\
&cFS2ExpAux :: (a, FS\ a\ b) \rightarrow () + (a, [(a, FS\ a\ b)])\\
&cFS2ExpAux = i_2 \cdot g \cdot h \cdot f\\
&\quad \mathbf{where}\\
&\qquad g :: (a, [(a, (FS\ a\ b) + (FS\ a\ b))]) \rightarrow (a, [(a, FS\ a\ b)])\\
&\qquad g = (id \times \mathsf{map}\ \ (id \times [id, id]))\\
&\qquad f :: (a, FS\ a\ b) \rightarrow (a, [(a, b + (FS\ a\ b))])\\
&\qquad f = (id \times outFS)\\
&\qquad h :: (a, [(a, b + (FS\ a\ b))]) \rightarrow (a, [(a, (FS\ a\ b) + (FS\ a\ b))])\\
&\qquad h = (id \times \mathsf{map}\ \ (id \times (\underline{(FS\ [\,])} + id)))
\end{aligned}
$$