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

Departamento de Informática
Universidade do Minho

Junho de 2019

Grupo nr.	99 (preencher)
a77045	Ricardo Barros Pereira
a78912	Miguel Pereira
a78997	Bruno Sousa

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" [1], 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:

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

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

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

     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

   - $compile :: String \rightarrow Codigo$ - 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>
```

   - $show' :: Expr \rightarrow String$ - 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:*

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

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

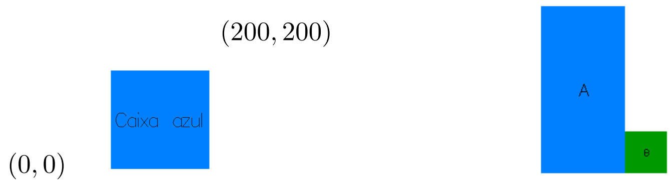

Figura 1: Caixa simples e caixa composta.

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

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,

$((200, 200), ($`"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

$ex2 = Comp\ Hb\ (Unid\ ((100, 200), ($`"A"`$, col\_blue)))$
$\qquad (Unid\ ((50, 50), ($`"B"`$, col\_green)))$

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.

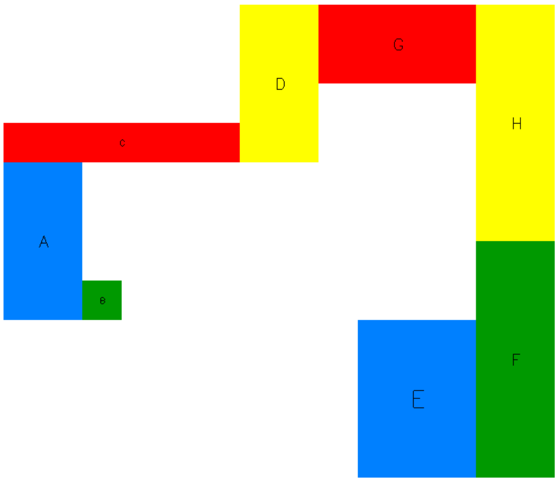

Figura 2: *Layout* feito de várias caixas coloridas.

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

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:

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

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

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

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

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

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

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

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

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

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

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

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:

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

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 [2], página 98.
[4] Podem obviamente usar-se outros símbolos, mas numa primeiraleitura dá jeito usarem-se tais nomes.
[5] Secção 3.17 de [2].

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:

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

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

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

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:

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

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:

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

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).
   $\mathit{check} :: \mathit{FS}\ a\ b \to \mathit{Bool}$

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

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

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

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

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

(c) Transformação de um arquivo com o conteúdo dos ficheiros indexado pelo *path* num sistema de ficheiros.
$untar :: [(Path\ a, b)] \rightarrow FS\ a\ b$
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.*

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

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

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

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

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

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

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

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.*

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

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

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

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

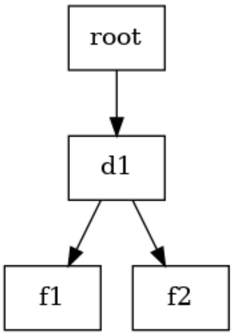

Figura 3: Exemplo de um sistema de ficheiros visualizado em Graphviz.

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

Sugestão: Construir um anamorfismo $nav :: (Path\ a, FS\ a\ b) \rightarrow FS\ a\ b$ 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.*

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

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

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

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

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

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:

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

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:

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

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 [2].
[7] Cf. [2], página 102.

C Código fornecido

Problema 1

Tipos:

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

Functor de base:

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

Instâncias:

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

Read para Exp's:

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

Combinadores:

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

```
prectos = parentesis '[' ']'
chavetas = parentesis '{' '}'

parentesis :: Char → Char → (ReadS a) → ReadS a
parentesis _ _ _ [] = []
parentesis ap pa r input
  = do
    ((_, (x, _)), c) ← ((readConst [ap]) `depois` (
      r `depois` (
      readConst [pa]))) input
    return (x, c)
```

Problema 2

Tipos:

```
type Fig = [(Origem, Caixa)]
type Origem = (Float, Float)
```

"Helpers":

```
col_blue = G.azure
col_green = darkgreen

darkgreen = G.dark (G.dark G.green)
```

Exemplos:

```
ex1Caixas = G.display (G.InWindow "Problema 4" (400, 400) (40, 40)) G.white $
  crCaixa (0, 0) 200 200 "Caixa azul" col_blue
ex2Caixas = G.display (G.InWindow "Problema 4" (400, 400) (40, 40)) G.white $
  caixasAndOrigin2Pict ((Comp Hb bbox gbox), (0.0, 0.0)) where
  bbox = Unid ((100, 200), ("A", col_blue))
  gbox = Unid ((50, 50), ("B", col_green))
ex3Caixas = G.display (G.InWindow "Problema 4" (400, 400) (40, 40)) G.white mtest where
  mtest = caixasAndOrigin2Pict $ (Comp Hb (Comp Ve bot top) (Comp Ve gbox2 ybox2), (0.0, 0.0))
  bbox1 = Unid ((100, 200), ("A", col_blue))
  bbox2 = Unid ((150, 200), ("E", col_blue))
  gbox1 = Unid ((50, 50), ("B", col_green))
  gbox2 = Unid ((100, 300), ("F", col_green))
  rbox1 = Unid ((300, 50), ("C", G.red))
  rbox2 = Unid ((200, 100), ("G", G.red))
  wbox1 = Unid ((450, 200), ("", G.white))
  ybox1 = Unid ((100, 200), ("D", G.yellow))
  ybox2 = Unid ((100, 300), ("H", G.yellow))
  bot = Comp Hb wbox1 bbox2
  top = (Comp Ve (Comp Hb bbox1 gbox1) (Comp Hb rbox1 (Comp H ybox1 rbox2)))
```

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

```
crCaixa :: Origem → Float → Float → String → G.Color → G.Picture
crCaixa (x, y) w h l c = G.Translate (x + (w / 2)) (y + (h / 2)) $ G.pictures [caixa, etiqueta] where
  caixa = G.color c (G.rectangleSolid w h)
  etiqueta = G.translate calc_trans_x calc_trans_y $
    G.Scale calc_scale calc_scale $ G.color G.black $ G.Text l
  calc_trans_x = (−((fromIntegral (length l)) ∗ calc_scale) / 2) ∗ base_shift_x
  calc_trans_y = (−calc_scale / 2) ∗ base_shift_y
  calc_scale = bscale ∗ (min h w)
  bscale = 1 / 700
```

$$
\begin{array}{l}
base\_shift\_y = 100 \\
base\_shift\_x = 64
\end{array}
$$

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:

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

Funções auxiliares:

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

Dados para testes:

- Sistema de ficheiros vazio:

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

- Nível 0

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

- Nível 1

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

- 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:

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

Outras funções auxiliares

Lógicas:

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

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

1. Começamos por definir o inExpr através do seu diagrama:

$$\begin{tikzcd} Int + (Op \times (Expr \times Expr)) \arrow[d, "{[f,g]}"'] \\ Expr \end{tikzcd}$$

Sendo assim, f define-se simplesmente como Num:

$$\begin{tikzcd} Int \arrow[d, "Num"'] \\ Expr \end{tikzcd}$$

Para obter g, sabemos que vamos precisar que g primeiro transforme o input para o tipo de entrada de Bop Expr Op Expr:

$$\begin{tikzcd} Op \times (Expr \times Expr) \arrow[d, "assocl"'] \\ (Op \times Expr) \times Expr \arrow[d, "swap \times id"'] \\ (Expr \times Op) \times Expr \end{tikzcd}$$

Fica assim, só a faltar ao invés de receber pares, receber um elemento de cada vez, que obtemos através da função uncurry.uncurry, logo:

$$\begin{tikzcd} Op \times (Expr \times Expr) \arrow[d, "((\widehat{\widehat{\cdot}})\ Bop) \cdot ((swap \times id) \cdot assocl)"'] \\ Expr \end{tikzcd}$$

E inExpr fica assim definido:

$$inExpr :: Int + (Op, (Expr, Expr)) \rightarrow Expr$$
$$inExpr = [Num, ((\widehat{\widehat{\cdot}})\ Bop) \cdot ((swap \times id) \cdot assocl)]$$

Para definir outExpr como se analisa a Expr por casos, basta fazer as injeções respetivas, tendo em atenção de no caso de Bop trocar a ordem dos elementos:

$$outExpr :: Expr \rightarrow Int + (Op, (Expr, Expr))$$
$$outExpr\ (Num\ a) = i_1\ a$$
$$outExpr\ (Bop\ e_1\ o\ e_2) = i_2\ (o, (e_1, e_2))$$

Para definir recExpr, cataExpr, anaExpr, hyloExpr, estes são imediatos tendo baseExpr definido:

$$recExpr\ f = baseExpr\ id\ f$$
$$cataExpr\ g = g \cdot (recExpr\ (cataExpr\ g)) \cdot outExpr$$
$$anaExpr\ g = inExpr \cdot (recExpr\ (anaExpr\ g)) \cdot g$$
$$hyloExpr\ h\ g = cataExpr\ h \cdot anaExpr\ g$$

2.

Começamos por definir calcula como um catamorfismo, visto que pretendemos perder informação, transformar uma expressão no seu resultado:

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

$$
\begin{aligned}
&calcula :: Expr \to Int\\
&calcula = cataExpr\ [id, calcop]\\
&\quad \mathbf{where}\ calcop\ (Op\ \texttt{"+"}, (e_1, e_2)) = e_1 + e_2\\
&\qquad calcop\ (Op\ \texttt{"*"}, (e_1, e_2)) = e_1 * e_2
\end{aligned}
$$

-- versao alternativa pointfree

$$
\begin{aligned}
&calcula' :: Expr \to Int\\
&calcula' = cataExpr\ [id, cond\ (((Op\ \texttt{"+"}) \equiv) \cdot \pi_1)\\
&\quad (fromIntegral \cdot add \cdot (toInteger \times toInteger) \cdot \pi_2)\\
&\quad (fromIntegral \cdot mul \cdot (toInteger \times toInteger) \cdot \pi_2)]
\end{aligned}
$$

3.

$$
\begin{aligned}
&compile :: String \to Codigo\\
&compile = hyloExpr\ conqCompile\ divCompile
\end{aligned}
$$

A função compile define-se como um hilomorfismo, pois queremos primeiramente transformar a String, através do anamorfismo divCompile, numa estrutura que representa uma árvore binária de expressões em que as operações se encontram na raiz e nas folhas os números, sendo que as operações vão aumentando de prioridade assim que se desce na árvore.

Para definir divCompile começamos por dividir em 2 casos: se a String só tiver um elemento estamos na presença de um número; caso contrário estamos perante uma expressão e para tal temos de partir a String em 3 substrings:

- operação menos prioritária;
- lado esquerdo da operação;
- lado direito da operação.

Para tal definimos auxDivComp que retorna um par contendo a posição da operação menos prioritária e qual o carater da operação, e de seguida através da função slice retiramos as substrings à esquerda e direita da operação, tirando os parêntesis.

$$
\begin{aligned}
&divCompile :: String \to Int + (Op, (String, String))\\
&divCompile\ [x] = i_1\ (digitToInt\ (x))\\
&divCompile\ l = i_2\ ((Op\ [c]), (slice\ 0\ p\ l, slice\ (p + 1)\ (\mathsf{length}\ \ (l))\ l))\\
&\quad \mathbf{where}\ (p, c) = auxDivComp\ l\ 0\ 0\\
&auxDivComp :: String \to Int \to Int \to (Int, Char)\\
&auxDivComp\ (h : t)\ c\ p\\
&\quad \mid (h \equiv \texttt{'('}) = auxDivComp\ t\ (c + 1)\ (p + 1)\\
&\quad \mid (h \equiv \texttt{')'}) = auxDivComp\ t\ (c - 1)\ (p + 1)\\
&\quad \mid (h \equiv \texttt{'+'} \lor h \equiv \texttt{'*'}) \land c \equiv 0 = (p, h)\\
&\quad \mid otherwise = auxDivComp\ t\ c\ (p + 1)\\
&slice :: Int \to Int \to String \to String\\
&slice\ start\ end\ string = \mathbf{if}\ (head\ (firstslice) \equiv \texttt{'('})\ \mathbf{then}\ (slice\ (start + 1)\ (end - 1)\ string)\ \mathbf{else}\ firstslice\\
&\quad \mathbf{where}\ firstslice = take\ (end - start)\ (drop\ start\ string)
\end{aligned}
$$

De seguida definimos o catamorfismo conqCompile que apenas tem de transformar o resultado do anamorfismo no Codigo correspondente, caso seja número ou operação:

$$
\begin{aligned}
&conqCompile :: Int + (Op, (Codigo, Codigo)) \to Codigo\\
&conqCompile = [auxNum, auxOp]\\
&auxNum :: Int \to Codigo\\
&auxNum\ a = [\texttt{"PUSH "} +\!\!+\ show\ (a)]\\
&auxOp\ (op, (c1, c2))\\
&\quad \mid op \equiv (Op\ \texttt{"+"}) = c1 +\!\!+\ c2 +\!\!+\ [\texttt{"ADD"}]\\
&\quad \mid op \equiv (Op\ \texttt{"*"}) = c1 +\!\!+\ c2 +\!\!+\ [\texttt{"MUL"}]
\end{aligned}
$$

Para definir o show′ começamos por desenhar o seu diagrama como um catamorfismo:

$$\begin{tikzcd}
Expr \arrow[d, "show'"'] & Int + (Op \times (Expr \times Expr)) \arrow[l, "inExpr"'] \arrow[d, "id+(id\times(show'\times show'))"] \\
String & Int + (Op \times (String \times String)) \arrow[l, "{[shownum, showop]}"]
\end{tikzcd}$$

e temos assim de definir showNum e showOp.

Para showNum apenas temos de fazer o show desse mesmo número, e caso seja negativo, colocar parênteses à volta do mesmo.

Em showOp temos de verificar de que tipo de operação se trata e apenas colocar as expressões já calculadas pela ordem correta.

$show' :: Expr \rightarrow String$
$show' = cataExpr\ [showNum, showOp]$
$showNum\ a = \mathbf{if}\ (a < 0)\ \mathbf{then}\ [\texttt{'('}] +\!\!+ (show\ a) +\!\!+ [\texttt{')'}]\ \mathbf{else}\ (show\ a)$
$showOp :: (Op, (String, String)) \rightarrow String$
$showOp\ (op, (s1, s2))$
$\quad |\ op \equiv (Op\ \texttt{"+"}) = [\texttt{'('}] +\!\!+ s1 +\!\!+ [\texttt{'+'}] +\!\!+ s2 +\!\!+ [\texttt{')'}]$
$\quad |\ otherwise = [\texttt{'('}] +\!\!+ s1 +\!\!+ [\texttt{'*'}] +\!\!+ s2 +\!\!+ [\texttt{')'}]$

Problema 2

Primeiro definimos os in e out e a partir destes a base, que de forma imediata nos deu o catamorfismo, anamorfismo e hilomorfismo:

$inL2D :: a + (b, (X\ a\ b, X\ a\ b)) \rightarrow X\ a\ b$
$inL2D = [Unid, \widehat{(\cdot\ Comp)}]$
$outL2D :: X\ a\ b \rightarrow a + (b, (X\ a\ b, X\ a\ b))$
$outL2D\ (Unid\ a) = i_1\ (a)$
$outL2D\ (Comp\ a\ b\ c) = i_2\ (a, (b, c))$
$baseL2D\ f\ g\ h = f + (g \times (h \times h))$
$recL2D\ f = baseL2D\ id\ id\ f$
$cataL2D\ g = g \cdot (recL2D\ (cataL2D\ g)) \cdot outL2D$
$anaL2D\ g = inL2D \cdot (recL2D\ (anaL2D\ g)) \cdot g$
$hyloL2D\ h\ g = cataL2D\ h \cdot anaL2D\ g$
$collectLeafs = cataL2D\ [singl, \mathsf{conc} \cdot \pi_2]$

$dimen :: X\ Caixa\ Tipo \rightarrow (Float, Float)$
$dimen = \bot$
$calcOrigins :: ((X\ Caixa\ Tipo), Origem) \rightarrow X\ (Caixa, Origem)\ ()$
$calcOrigins = \bot$
$calc :: Tipo \rightarrow Origem \rightarrow (Float, Float) \rightarrow Origem$
$calc = \bot$
$caixasAndOrigin2Pict = \bot$

Problema 3

Seguindo o Anexo B:

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

Seja:

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

Então:

$$cos\ x\ 0 = 1$$

$$cos\ x\ (n+1) = cos\ x\ n + \frac{(-1)^{(n+1)}}{(2(n+1))!} * x^{2(n+1)}$$

Se $h\ x\ n = \frac{(-1)^{(n+1)}}{(2(n+1))!} * x^{2(n+1)}$, obtemos então:

$$h\ x\ 0 = -\frac{x^2}{2}$$

$$h\ x\ (n+1) = \frac{(-1)^{(n+1+1)}}{(2(n+1+1))!} * x^{2(n+1+1)}$$

Desenvolvendo $h\ x\ (n+1)$:

$$h\ x\ (n+1) = \frac{(-1)^{(n+1)} * (-1)}{(2(n+1))! * (2n+3) * (2n+4)} * x^{2(n+1)} * x^2$$

$$h\ x\ (n+1) = h\ x\ n + \frac{(-x)^2}{(2n+3) * (2n+4)}$$

Definindo as funções auxiliares a e b, que correspondem a $2\ n+3$ e $2\ n+4$ respetivamente, obtemos no final:

$$\begin{array}{l} cos\ x\ 0 = 1 \\ cos\ x\ (n+1) = cos\ x\ n + h\ x\ n \\ h\ x\ 0 = ((-x) \uparrow 2)\ /\ 2 \\ h\ x\ (n+1) = h\ x\ n * ((-x) \uparrow 2)\ /\ ((a\ n) * (b\ n)) \\ a\ 0 = 3 \\ a\ (n+1) = 2 + a\ n \\ b\ 0 = 4 \\ b\ (n+1) = 2 + b\ n \end{array}$$

E obtemos assim a solução:

$$\begin{array}{l} coss\ x\ 0 = 1 \\ coss\ x\ (n+1) = (coss\ x\ n) + (h\ x\ n) \\ h\ x\ 0 = -(x \uparrow 2)\ /\ 2 \\ h\ x\ (n+1) = (h\ x\ n) * (-(x \uparrow 2))\ /\ ((a\ n) * (b\ n)) \\ a\ 0 = 3 \\ a\ (n+1) = 2 + a\ n \\ b\ 0 = 4 \\ b\ (n+1) = 2 + b\ n \\ cos'\ x = prj \cdot \mathsf{for}\ loop\ init\ \mathbf{where} \\ \quad loop\ (coss, h, a, b) = (coss + h, h * (-(x \uparrow 2))\ /\ (a * b), 2 + a, 2 + b) \\ \quad init = (1, -(x \uparrow 2)\ /\ 2, 3, 4) \\ \quad prj\ (coss, h, a, b) = coss \end{array}$$

Valorização:

```c
double coss(double x,int n){

  double res = 1;
  double h = -(x*x)/2;
  double a = 3;
  double b = 4;

  for (int i = 1; i <= n; i++){

    res = res + h;
    h = h *((-(x*x))/(a*b));
    a = a + 2;
    b = b + 2;
  }

  return res;
}

int main(int argc, char const *argv[]){
  if(argc != 3)
    return 0;

  double r = coss(atof(argv[1]),atoi(argv[2]));
  printf("%.16f\n",r);

  return 0;
}
```

Problema 4

1.

Para definir a função outFS recorremos a uma função auxiliar que apenas faz uma injeção à esquerda ou à direita, consoante o parâmetro for um Ficheiro(File) ou uma Diretoria(Dir), respetivamente. Definida esta função auxiliar, o outFS é um map com o produto da função identidade e da função outNode, definida por nós.

$$outFS\ (FS\ l) = (\mathsf{map}\ \ (id \times outNode))\ l$$
$$outNode\ (File\ f) = (i_1\ f)$$
$$outNode\ (Dir\ f) = (i_2\ f)$$

Como o baseFS vai ser aplicado a cada elemento que sai do outFS, temos um map do produto da função f com a soma das funções g e h.

$$baseFS\ f\ g\ h = \mathsf{map}\ \ (f \times (g + h))$$

Após definir o baseFS, obtemos de forma imediata o cataFS, o anaFS e o hyloFS.

$$cataFS :: ([(a, b + c)] \to c) \to FS\ \ a\ \ b \to c$$
$$cataFS\ \ g = g \cdot (recFS\ \ (cataFS\ \ g)) \cdot outFS$$
$$anaFS :: (c \to [(a, b + c)]) \to c \to FS\ \ a\ \ b$$
$$anaFS\ \ g = inFS \cdot (recFS\ \ (anaFS\ \ g)) \cdot g$$
$$hyloFS\ \ g\ \ h = cataFS\ \ g \cdot anaFS\ \ h$$

2.

$$\begin{tikzcd}
FS\ a\ b \arrow[d, "cataFS\ g"'] & (a \times (b + FS\ a\ b))* \arrow[l, "inFS"'] \arrow[d, "\mathsf{map}\ \ (id\times(id+(cataFS\ g))*"] \\
c & (a \times (b + c))* \arrow[l, "g"]
\end{tikzcd}$$

3.
Outras funções pedidas:
a)

$$
\begin{array}{l}
check :: (Eq\ a) \Rightarrow FS\ a\ b \rightarrow Bool \\
check = cataFS\ (parbool \cdot \langle repetidos \cdot \mathsf{map}\ \ (\pi_1), check' \cdot \mathsf{map}\ \ ([true, id] \cdot \pi_2)\rangle) \\
parbool\ (a, b) = \mathbf{if}\ (a \equiv False \lor b \equiv False)\ \mathbf{then}\ False\ \mathbf{else}\ True \\
check'\ [\,] = True \\
check'\ (x : xs) = \mathbf{if}\ (x \equiv False)\ \mathbf{then}\ False\ \mathbf{else}\ check'\ xs \\
repetidos\ [\,] = True \\
repetidos\ (h : t) = \mathbf{if}\ (x\ h\ t)\ \mathbf{then}\ False\ \mathbf{else}\ repetidos\ t \\
\quad \mathbf{where} \\
\qquad x\ h\ [\,] = False \\
\qquad x\ h\ (y : ys) = \mathbf{if}\ (h \equiv y)\ \mathbf{then}\ True\ \mathbf{else}\ x\ h\ ys
\end{array}
$$

Diagrama de check definido como um catamorfismo:

$$
\begin{tikzcd}
FS\ a\ b \arrow[d, "check"'] & (a \times (b + FS\ a\ b))* \arrow[l, "inFS"'] \arrow[d, "\mathsf{map}\ \ (id \times (id + check)*"] \\
Bool & (a \times (b + Bool))* \arrow[l, "f"]
\end{tikzcd}
$$

Diagrama de f:

$$
\begin{tikzcd}
(a \times (b + Bool))* \arrow[d, "<repetidos \cdot \mathsf{map}\ \ (\pi_1), check' \cdot (\mathsf{map}\ \ ([true, id] \cdot \pi_2))>"'] \\
Bool \times Bool \arrow[d, "parbool"'] \\
Bool
\end{tikzcd}
$$

b)

$$
\begin{array}{l}
tar :: FS\ a\ b \rightarrow [(Path\ a, b)] \\
tar = cataFS\ (disc \cdot \mathsf{map}\ \ (auxtar)) \\
auxtar\ (a, (i_1\ b)) = [([a], b)] \\
auxtar\ (a, (i_2\ [\,])) = [\,] \\
auxtar\ (a, (i_2\ ((x, y) : xs))) = ([a] +\!\!+\ x, y) : auxtar\ (a, i_2\ xs) \\
disc\ [\,] = [\,] \\
disc\ (x : xs) = x +\!\!+\ disc\ xs
\end{array}
$$

Diagrama de tar:

$$
\begin{tikzcd}
FS\ a\ b \arrow[d, "tar"'] & (a \times (b + FS\ a\ b))* \arrow[l, "inFS"'] \arrow[d, "\mathsf{map}\ \ (id \times (id + tar)*"] \\
(Path\ a \times b)* & (a \times (b + (Path\ a \times b)*))* \arrow[l, "disc \cdot \mathsf{map}\ \ (auxtar)"]
\end{tikzcd}
$$

c)

$$
\begin{array}{l}
untar :: (Eq\ a) \Rightarrow [(Path\ a, b)] \rightarrow FS\ a\ b \\
untar = joinDupDirs \cdot anaFS\ (\mathsf{map}\ \ (intar)) \\
intar\ ([a], b) = (a, i_1\ b) \\
intar\ ((x : xs), b) = (x, i_2\ [(xs, b)])
\end{array}
$$

Diagrama do anamorfismo presente em untar:

$$\begin{tikzcd}
(a \times (b + (Path\ a \times b)*))* \arrow[d, "\mathsf{map}\ (id \times (id + untar)*"'] & (Path\ a \times b)* \arrow[l, "\mathsf{map}\ (intar)"'] \arrow[d, "untar"] \\
(a \times (b + FS\ a\ b))* & FS\ a\ b \arrow[l, "outFS"]
\end{tikzcd}$$

d) Para definir o find, fizemos uso da função tar para dar todos os paths existentes no sistema de ficheiros, que depois percorremos à procura do ficheiro prentendido através da função auxfind.

$find :: (Eq\ a) \Rightarrow a \rightarrow FS\ a\ b \rightarrow [Path\ a]$
$find = curry\ (auxfind \cdot (id \times tar))$

$auxfind :: (Eq\ a) \Rightarrow (a, [(Path\ a, b)]) \rightarrow [Path\ a]$
$auxfind\ (a, [\,]) = [\,]$
$auxfind\ (a, (l, \_) : xs) = \mathbf{if}\ (last\ (l) \equiv a)\ \mathbf{then}\ l : auxfind\ (a, xs)\ \mathbf{else}\ auxfind\ (a, xs)$

e)
Para acrescentar um novo ficheiro utilizamos a função tar para transformar o sistema de ficheiros na sua lista de paths e conteúdos respetivos e adicionamos o ficheiro novo a essa lista e fazemos o untar para voltarmos ao sistema de ficheiros:

$new :: (Eq\ a) \Rightarrow Path\ a \rightarrow b \rightarrow FS\ a\ b \rightarrow FS\ a\ b$
$new = uncurriedNew$

$uncurriedNew\ p\ b\ fs = untar\ ([(p, b)] +\!\!+\ tar\ (fs))$

f)
Para definir a função de cópia, utilizamos mais uma vez tar para através da função getconteudo obtermos o conteudo associado ao path de origem. De seguida acrescentamos o par (path destino,conteudo) ao tar do sistema de ficheiros e fazemos o untar, tendo assim criado um novo ficheiro com o conteúdo do ficheiro original.

$cp :: (Eq\ a) \Rightarrow Path\ a \rightarrow Path\ a \rightarrow FS\ a\ b \rightarrow FS\ a\ b$
$cp = cpaux$

$cpaux\ s\ d\ f = \mathbf{if}\ (s \equiv [\,] \vee d \equiv [\,])\ \mathbf{then}\ f\ \mathbf{else}\ untar\ ([(d, conteudo)] +\!\!+\ ltar)$
  $\mathbf{where}$
    $ltar = tar\ (f)$
    $conteudo = getconteudo\ s\ ltar$

$getconteudo :: (Eq\ a) \Rightarrow Path\ a \rightarrow [(Path\ a, b)] \rightarrow b$
$getconteudo\ d\ [(l, b)] = b$
$getconteudo\ d\ ((l, b) : xs) = \mathbf{if}\ (l \equiv d)\ \mathbf{then}\ b\ \mathbf{else}\ (getconteudo\ d\ xs)$

g)

$rm :: (Eq\ a) \Rightarrow (Path\ a) \rightarrow (FS\ a\ b) \rightarrow FS\ a\ b$
$rm = \perp$

$auxJoin :: ([(a, b + c)], d) \rightarrow [(a, b + (d, c))]$
$auxJoin = \perp$

Valorização:
Definimos um anamorfismo sobre Exp que a cada elemento presente no sistema de ficheiros, $(a, Node\ a\ b)$ através da função inTerm obtemos pares nome e sistema de ficheiros inferior, que será vazio caso seja um ficheiro. No final através da função rootTerm acrescentámos a root do sistema.

$cFS2Exp :: a \rightarrow FS\ a\ b \rightarrow (Exp\ ()\ a)$
$cFS2Exp = curry\ (rootTerm \cdot (id \times \mathsf{map}\ (anaExp\ (inTerm)) \cdot noFS))$

$noFS\ (FS\ l) = l$

$inTerm :: (a, Node\ a\ b) \rightarrow () + (a, [(a, Node\ a\ b)])$
$inTerm\ (a, File\ b) = i_2\ (a, [\,])$
$inTerm\ (a, Dir\ (FS\ c)) = i_2\ (a, c)$

$rootTerm\ (a, b) = Term\ a\ b$

Índice

Referências

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

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