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

Departamento de Informática
Universidade do Minho

Junho de 2019

Grupo nr.	10
a85504	João Teixeira
a84776	Emanuel Rodrigues
a83683	José 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" [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:

   - $calcula :: Expr \to Int$ 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.*

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

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

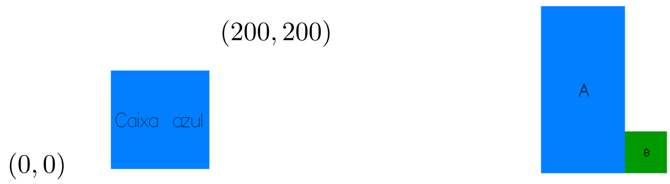

Figura 1: Caixa simples e caixa composta.

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

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.

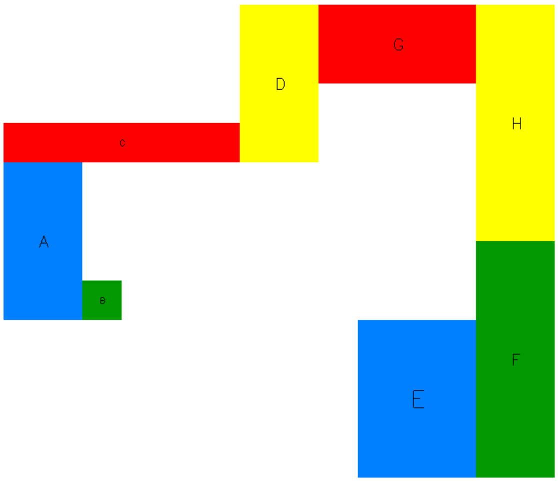

Figura 2: *Layout* feito de várias caixas coloridas.

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

A informação mais relevante deste tipo é a referente à lista de "caixas posicionadas" (tipo $(\mathit{Origem}, \mathit{Caixa})$). 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 $\mathit{calc\_origems}$, que calcula as coordenadas iniciais das caixas no plano:

   $$\mathit{calc\_origems} :: (\mathit{L2D}, \mathit{Origem}) \rightarrow \mathsf{X}\ (\mathit{Caixa}, \mathit{Origem})\ ()$$

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

   $$\mathit{agrup\_caixas} :: \mathsf{X}\ (\mathit{Caixa}, \mathit{Origem})\ () \rightarrow \mathit{Fig}$$

Um segundo problema neste projecto é *descobrir como visualizar a informação gráfica calculada por* $\mathit{desenho}$. 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

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

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

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

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

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

que dada uma frase da linguagem $\mathit{L2D}$ e coordenadas iniciais apresenta o respectivo desenho no ecrã. **Sugestão**: Use a função $\mathit{G.pictures}$ 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

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

Obter-se-á de imediato

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

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:

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

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

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

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

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

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 $cos'\ x$ calcula bem o coseno de π e o coseno de $\pi\ /\ 2$:*

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

[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:

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

O que é para fazer:

1. Definir as funções outFS, baseFS, $(\!|\cdot|\!)$, anaFS e hyloFS.
2. Apresentar, no relatório, o diagrama de $(\!|\cdot|\!)$.
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:*

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

(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.*

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

(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.*

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

(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.*

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

(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.*

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

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.*

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

(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.*

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

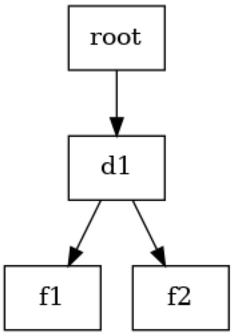

Figura 3: Exemplo de um sistema de ficheiros visualizado em Graphviz.

(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.*

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

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:

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

[6] Exemplos tirados de [2].
[7] Cf. [2], página 102.

C Código fornecido

Problema 1

Tipos:

```
data Expr = Num Int
  | Bop Expr Op Expr deriving (Eq, Show)
data Op = Op String deriving (Eq, Show)
type Codigo = [String]
```

Functor de base:

```
baseExpr f g = id + (f × (g × g))
```

Instâncias:

```
instance Read Expr where
  readsPrec _ = readExp
```

Read para Exp's:

```
readOp :: String → [(Op, String)]
readOp input = do
  (x, y) ← lex input
  return ((Op x), y)
readNum :: ReadS Expr
readNum = (map (λ(x, y) → ((Num x), y))) · reads
readBinOp :: ReadS Expr
readBinOp = (map (λ((x, (y, z)), t) → ((Bop x y z), t))) ·
  ((readNum ‘ou‘ (pcurvos readExp))
    ‘depois‘ (readOp ‘depois‘ readExp))
readExp :: ReadS Expr
readExp = readBinOp ‘ou‘ (
  readNum ‘ou‘ (
  pcurvos readExp))
```

Combinadores:

```
depois :: (ReadS a) → (ReadS b) → ReadS (a, b)
depois _ _ [] = []
depois r1 r2 input = [((x, y), i2) | (x, i1) ← r1 input,
  (y, i2) ← r2 i1]
readSeq :: (ReadS a) → ReadS [a]
readSeq r input
  = case (r input) of
    [] → [([], input)]
    l → concat (map continua l)
      where continua (a, i) = map (c a) (readSeq r i)
        c x (xs, i) = ((x : xs), i)
ou :: (ReadS a) → (ReadS a) → ReadS a
ou r1 r2 input = (r1 input) ++ (r2 input)
senao :: (ReadS a) → (ReadS a) → ReadS a
senao r1 r2 input = case (r1 input) of
  [] → r2 input
  l    → l
readConst :: String → ReadS String
readConst c = (filter ((≡ c) · π1)) · lex
pcurvos = parentesis '(' ')'
```

```
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
```

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

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:

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

Funções auxiliares:

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

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:

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

Outras funções auxiliares

Lógicas:

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

Compilação e execução dentro do interpretador:[8]

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

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

$$
\begin{aligned}
&inExpr :: Int + (Op, (Expr, Expr)) \rightarrow Expr\\
&inExpr = [Num, \widehat{\widehat{Bop}} \cdot (swap \times id) \cdot assocl]\\
&outExpr :: Expr \rightarrow Int + (Op, (Expr, Expr))\\
&outExpr\ (Num\ a) = i_1\ a\\
&outExpr\ (Bop\ e_1\ o1\ e_2) = i_2\ (o1, (e_1, e_2))\\
&recExpr\ f = baseExpr\ id\ f\\
&(\!|g|\!) = g \cdot (recExpr\ (\!|g|\!)) \cdot outExpr\\
&anaExpr\ f = inExpr \cdot (recExpr\ (anaExpr\ f)) \cdot f\\
&hyloExpr\ a\ c = (\!|a|\!) \cdot anaExpr\ c
\end{aligned}
$$

Calcula

$$
\begin{aligned}
&calcula :: Expr \rightarrow Int\\
&calcula\ e = (\!|[id, calcBop]|\!)\ e\\
&calcBop\ (Op\ \texttt{"+"}, (a, b)) = (+)\ a\ b\\
&calcBop\ (Op\ \texttt{"-"}, (a, b)) = (-)\ a\ b\\
&calcBop\ (Op\ \texttt{"*"}, (a, b)) = (*)\ a\ b
\end{aligned}
$$

$$
\begin{tikzcd}
Expr \arrow[r, "outExpr"'] \arrow[d, "calcula"'] & Int + (Op \times (Expr \times Expr)) \arrow[d, "id + id \times calcula^{2}"] \\
Int & Int + (Op \times (Int, Int)) \arrow[l, "g = [id, calcBop]"]
\end{tikzcd}
$$

Show

$$
\begin{aligned}
&show' = (\!|[show, showBop]|\!)\\
&\quad \textbf{where}\ showBop\ (Op\ c, (a, b)) = \texttt{"("} +\!\!+\ a +\!\!+ \texttt{" "} +\!\!+\ c +\!\!+ \texttt{" "} +\!\!+\ b +\!\!+ \texttt{")"}
\end{aligned}
$$

$$
\begin{tikzcd}
Expr \arrow[r, "outExpr"'] \arrow[d, "show'"'] & Int + (Op \times (Expr \times Expr)) \arrow[d, "id + id \times show'^{2}"] \\
String & Int + (Op \times (String, String)) \arrow[l, "g = [show, showBop]"]
\end{tikzcd}
$$

Compile

$$
\begin{aligned}
&compile :: String \rightarrow Codigo\\
&compile = (\!|[pushN, stackBop]|\!) \cdot read\\
&\quad \textbf{where}\ pushN\ x = (singl\ (\texttt{"PUSH "} +\!\!+\ show\ x))\\
&stackBop\ (Op\ \texttt{"+"}, (a, b)) = a +\!\!+\ b +\!\!+ [\texttt{"ADD"}]\\
&stackBop\ (Op\ \texttt{"-"}, (a, b)) = a +\!\!+\ b +\!\!+ [\texttt{"SUB"}]\\
&stackBop\ (Op\ \texttt{"*"}, (a, b)) = a +\!\!+\ b +\!\!+ [\texttt{"MUL"}]
\end{aligned}
$$

$$\begin{tikzcd}
String \arrow[d, "read"'] & \\
Expr \arrow[r, "outExpr"'] \arrow[d, "(\!|g|\!)"'] & Int + (Op \times (Expr \times Expr)) \arrow[d, "id + id \times (\!|g|\!)^{2}"] \\
Codigo & Int + (Op \times (Codigo, Codigo)) \arrow[l, "g = [pushN, stackBop]"]
\end{tikzcd}$$

Problema 2

$$\begin{array}{l}
inL2D :: a + (b, (X\ a\ b, X\ a\ b)) \rightarrow X\ a\ b \\
inL2D = [Unid, comp] \\
comp :: (b, (X\ a\ b, X\ a\ b)) \rightarrow X\ a\ b \\
comp\ (b, (l, r)) = Comp\ b\ l\ r \\
outL2D :: X\ a\ b \rightarrow a + (b, (X\ a\ b, X\ a\ b)) \\
outL2D\ (Unid\ a) = i_1\ a \\
outL2D\ (Comp\ a\ f\ g) = i_2\ (a, (f, g)) \\
baseL2D\ f\ g\ h\ i = f + (g \times (h \times i)) \\
recL2D\ f = baseL2D\ id\ id\ f\ f \\
cataL2D\ g = g \cdot (recL2D\ (cataL2D\ g)) \cdot outL2D \\
[\!(g)\!] = inL2D \cdot (recL2D\ [\!(g)\!]) \cdot g
\end{array}$$

Dimensão

Para obter a dimensão de uma $L2D$ aplicasse um catamorfismo.

$$\begin{array}{l}
dimen :: L2D \rightarrow (Float, Float) \\
dimen = cataL2D\ [(fromIntegral \times fromIntegral) \cdot \pi_1, addH] \\
\quad \textbf{where} \\
\qquad addH :: (Tipo, ((Float, Float), (Float, Float))) \rightarrow (Float, Float) \\
\qquad addH\ (V, ((x1, y1), (x2, y2))) = (x1 + zeroer\ (x2 - x1\ /\ 2), y1 + y2) \\
\qquad addH\ (Vd, ((x1, y1), (x2, y2))) = (x1 + x2\quad, y1 + y2\qquad) \\
\qquad addH\ (Ve, ((x1, y1), (x2, y2))) = (max\ x1\ x2, y1 + y2\qquad) \\
\qquad addH\ (H, ((x1, y1), (x2, y2))) = (x1 + x2\quad, y1 + zeroer\ (y2 - y1\ /\ 2)) \\
\qquad addH\ (Ht, ((x1, y1), (x2, y2))) = (x1 + x2\quad, y1 + y2\qquad) \\
\qquad addH\ (Hb, ((x1, y1), (x2, y2))) = (x1 + x2\quad, max\ y1\ y2\qquad) \\
\qquad zeroer\ x \mid x < 0 = 0 \\
\qquad\quad \mid otherwise = x
\end{array}$$

$$\begin{tikzcd}
L2D \arrow[r, "outL2D"'] \arrow[d, "dimen"'] & Caixa + Tipo \times L2D^{\ 2} \arrow[d, "id + id \times dimen^{\ 2}"] \\
(Float, Float) & Caixa + Tipo \times (Float, Float)^{\ 2} \arrow[l, "g = [(fromIntegral \times fromIntegral) \cdot \pi_1, addH]"]
\end{tikzcd}$$

Calcular Origens

$$\begin{array}{l}
calcOrigins :: (L2D, Origem) \rightarrow X\ (Caixa, Origem)\ () \\
calcOrigins = [\!(g)\!] \\
\quad \textbf{where} \\
\qquad g\ ((Unid\ c), o) = i_1\ (c, o)
\end{array}$$

$$
\begin{aligned}
&\quad g\ ((\mathit{Comp}\ t\ l\ r), o) = i_2\ ((), ((l, o), (r, \mathit{calc}\ t\ o\ \$\ \mathit{dimen}\ l)))\\
&\mathit{calc} :: \mathit{Tipo} \rightarrow \mathit{Origem} \rightarrow (\mathit{Float}, \mathit{Float}) \rightarrow \mathit{Origem}\\
&\mathit{calc}\ V\ (ox, oy)\ (sx, sy) = (ox + sx\ /\ 2, oy + sy)\\
&\mathit{calc}\ \mathit{Vd}\ (ox, oy)\ (sx, sy) = (ox + oy, oy + sy\quad)\\
&\mathit{calc}\ \mathit{Ve}\ (ox, oy)\ (sx, sy) = (ox\qquad\ , oy + sy\quad)\\
&\mathit{calc}\ H\ (ox, oy)\ (sx, sy) = (ox + sx\ \ , oy + sy\ /\ 2)\\
&\mathit{calc}\ \mathit{Ht}\ (ox, oy)\ (sx, sy) = (ox + sx\ , oy + sy\quad)\\
&\mathit{calc}\ \mathit{Hb}\ (ox, oy)\ (sx, sy) = (ox + sx\ , oy\qquad\quad)
\end{aligned}
$$

$$
\begin{tikzcd}
\mathit{L2D} \times \mathit{Origem} \arrow[r, "g"] \arrow[d, "\mathit{calcOrigins} = (\!|g|\!)"'] & (\mathit{Caixa} \times \mathit{Origem}) + \mathit{Nil} \times ((\mathit{L2D} \times \mathit{Origem})^{2}) \arrow[d, "id + id \times (\!|g|\!)^{2}"] \\
X\ (\mathit{Caixa} \times \mathit{Origem})\ \mathit{Nil} & (\mathit{Caixa} \times \mathit{Origem}) + \mathit{Nil} \times (X\ (\mathit{Caixa} \times \mathit{Origem})\ \mathit{Nil})^{2} \arrow[l, "\mathit{inL2D}"']
\end{tikzcd}
$$

Agrupar Caixas

Para agrupar as caixas numa Fig basta fazer um $\mathit{collectLeafs}$, um catamorfismo que recolhe todas as folhas, e depois mapear a função swap a todos os elementos do output.

$$
\begin{aligned}
&\mathit{agrup\_caixas} :: X\ (\mathit{Caixa}, \mathit{Origem})\ () \rightarrow \mathit{Fig}\\
&\mathit{agrup\_caixas} = \mathsf{map}\ \ \mathit{swap} \cdot \mathit{collectLeafs}\\
&\mathit{collectLeafs} :: X\ a\ b \rightarrow [a]\\
&\mathit{collectLeafs} = \mathit{cataL2D}\ [\mathit{singl}, \widehat{(+\!\!+)} \cdot \pi_2]
\end{aligned}
$$

$$
\begin{tikzcd}
X\ a\ b \arrow[r, "\mathit{outL2D}"'] \arrow[d, "\mathit{collectLeafs}"'] & a + b \times (X\ a\ b \times X\ a\ b) \arrow[d, "id + id \times (\mathit{collectLeafs} \times \mathit{collectLeafs})"] \\
{[a]} & a + b \times (a \times a) \arrow[l, "g = [\mathit{singl}, \widehat{(+\!\!+)} \cdot \pi_2]"]
\end{tikzcd}
$$

Mostrar Caixas

Para criar a função $\mathit{mostra\_caixas}$ basta utilizar as funções definidas até agora. Primeiro calculasse as origens de cada caixa com a função $\mathit{calcOrigins}$. Depois criasse uma lista de Fig com a função $\mathit{agrup\_caixas}$. Em seguida, convertesse cada Fig para uma $G.\mathit{Picture}$, com recurso a uma função auxiliar que faz uso da $\mathit{crCaixa}$, e juntasse todas com recurso à função $G.\mathit{Pictures}$. Por fim, utilizasse a função pré-definida $\mathit{display}$.

$$
\begin{aligned}
&\mathit{caixasAndOrigin2Pict} :: (\mathit{L2D}, \mathit{Origem}) \rightarrow G.\mathit{Picture}\\
&\mathit{caixasAndOrigin2Pict} = G.\mathit{Pictures} \cdot (\mathsf{map}\ \ \mathit{figToPic}) \cdot \mathit{agrup\_caixas} \cdot \mathit{calcOrigins}\\
&\quad \mathbf{where}\\
&\qquad \mathit{figToPic}\ (o, ((sx, sy), (s, c))) = \mathit{crCaixa}\ o\ (\mathit{fromIntegral}\ sx)\ (\mathit{fromIntegral}\ sy)\ s\ c\\
&\mathit{mostra\_caixas} :: (\mathit{L2D}, \mathit{Origem}) \rightarrow \mathsf{IO}\ ()\\
&\mathit{mostra\_caixas} = \mathit{display} \cdot \mathit{caixasAndOrigin2Pict}
\end{aligned}
$$

Problema 3

Solução:

$$
\begin{aligned}
&\mathit{cos}'\ x = \mathit{prj} \cdot \mathsf{for}\ \mathit{loop}\ \mathit{init}\ \mathbf{where}\\
&\quad \mathit{loop}\ (e, h, s, g) = (h + e, -(x \uparrow 2)\ /\ (s * g) * h, 2 + s, 2 + g)\\
&\quad \mathit{init} = (1, -(x \uparrow 2)\ /\ 2, 4, 3)\\
&\quad \mathit{prj}\ (e, h, s, g) = e
\end{aligned}
$$

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

Seja $e\ x\ n = \sum_{i=0}^{n} \frac{(-1)^i}{(2i)!} x^{2i}$ a função que dá essa aproximação. Podemos ver que $e\ x\ 0 = 1$ e que $e\ x\ (n+1) = e\ x\ n + \frac{(-1)^{n+1}}{(2(n+1))!} x^{2(n+1)}$. Se definirmos $h\ x\ n = \frac{(-1)^{n}+1}{(2(n+1))!} x^{2(n+1)}$ teremos $e\ x$ e $h\ x$ em recursividade mútua. Repetindo o processo para $h\ x\ n$ etc obteremos:

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

Valorização

```
float cos (float x, int n) {
   float e = 1;
   float h = -(x * x) / 2;
   float s = 4;
   float g = 3;
   float i;
   for i = 1; i ⩽ n; i ++ {
      e = e + h;
      h = (-(x * x) / (s * g)) * h;
      s += 2;
      g += 2;
   }
   return e;
}
```

Problema 4

Triologia "ana-cata-hilo":

$$\begin{array}{l}
outFS\ (FS\ l) = \mathsf{map}\ \ (id \times outNode)\ l \\
outNode\ (File\ b) = i_1\ b \\
outNode\ (Dir\ fs) = i_2\ fs \\
baseFS\ f\ g\ h = \mathsf{map}\ \ (f \times (g + h)) \\
(\!|\cdot|\!) :: ([(a, b + c)] \to c) \to FS\ a\ b \to c \\
(\!|g|\!) = g \cdot (recFS\ (\!|g|\!)) \cdot outFS \\
anaFS :: (c \to [(a, b + c)]) \to c \to FS\ a\ b \\
anaFS\ g = inFS \cdot (recFS\ (anaFS\ g)) \cdot g \\
hyloFS\ g\ h = (\!|g|\!) \cdot anaFS\ h
\end{array}$$

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

Outras funções pedidas:

Check

$$
\begin{array}{l}
\mathit{check} :: (\mathit{Eq}\ a) \Rightarrow \mathit{FS}\ a\ b \rightarrow \mathit{Bool} \\
\mathit{check} = (\!|\, a \,|\!) \\
\quad \textbf{where}\ a\ b = (\mathsf{length}\ \ b \equiv (\mathsf{length}\ \$ \mathit{nub}\ \$\ \mathsf{map}\ \ (\pi_1)\ \$\ b)) \wedge ((\mathsf{length}\ \$ \mathit{filter}\ ([\underline{\mathit{True}}, \mathit{id}] \cdot \pi_2)\ b) \equiv 0)
\end{array}
$$

$$
\begin{tikzcd}
FS\ A\ B \arrow[r, "outFS"'] \arrow[d, "check"'] & A \times (B + FS\ A\ B) \arrow[d, "id \times (id + check)"] \\
C & A \times (B + C) \arrow[l, "a"]
\end{tikzcd}
$$

Tar

$$
\begin{array}{l}
\mathit{tar} :: \mathit{FS}\ a\ b \rightarrow [(\mathit{Path}\ a, b)] \\
\mathit{tar} = (\!|\, x \,|\!) \\
\quad \textbf{where}\ x = \mathit{concatMap}\ (\lambda(a, b) \rightarrow [\lambda x \rightarrow [([a], x)], \mathsf{map}\ \ (\lambda(x, y) \rightarrow ((a : x), y))]\ b)
\end{array}
$$

$$
\begin{tikzcd}
FS\ A\ B \arrow[r, "outFS"'] \arrow[d, "tar"'] & A \times (B + FS\ A\ B) \arrow[d, "id \times (id + tar)"] \\
C & A \times (B + C) \arrow[l, "x"]
\end{tikzcd}
$$

Untar

$$
\begin{array}{l}
\mathit{untar} :: (\mathit{Eq}\ a) \Rightarrow [(\mathit{Path}\ a, b)] \rightarrow \mathit{FS}\ a\ b \\
\mathit{untar} = \mathit{anaFS}\ x \\
\quad \textbf{where}\ x = \mathsf{map}\ \ (\lambda(a, (b, c)) \rightarrow \textbf{if}\ (\mathit{null}\ b)\ \textbf{then}\ (a, (i_1\ c))\ \textbf{else}\ (a, i_2\ [(b, c)])) \cdot \mathsf{map}\ \ (\lambda(a, b) \rightarrow (\mathit{head}\ a, (\mathit{tail}
\end{array}
$$

$$
\begin{tikzcd}
FS\ A\ B & A \times (B + FS\ A\ B) \arrow[l, "inFS"'] \\
C \arrow[u, "untar"'] \arrow[r, "x"] & A \times (B + C) \arrow[u, "id \times (id + untar)"]
\end{tikzcd}
$$

Find

$$
\begin{array}{l}
\mathit{find} :: (\mathit{Eq}\ a) \Rightarrow a \rightarrow \mathit{FS}\ a\ b \rightarrow [\mathit{Path}\ a] \\
\mathit{find}\ o = (\!|\, f \,|\!) \\
\quad \textbf{where}\ f = \mathit{concatMap}\ (\lambda(a, b) \rightarrow \textbf{if}\ (a \equiv o)\ \textbf{then}\ [\underline{[[a]]}, \mathsf{map}\ \ ((:)\ a)]\ b\ \textbf{else}\ [\underline{[\,]}, \mathsf{map}\ \ ((:)\ a)]\ b)
\end{array}
$$

$$
\begin{tikzcd}
FS\ A\ B \arrow[r, "outFS"'] \arrow[d, "find"'] & A \times (B + FS\ A\ B) \arrow[d, "id \times (id + find)"] \\
C & A \times (B + C) \arrow[l, "f"]
\end{tikzcd}
$$

New

$$
\begin{array}{l}
\mathit{new} :: (\mathit{Eq}\ a) \Rightarrow \mathit{Path}\ a \rightarrow b \rightarrow \mathit{FS}\ a\ b \rightarrow \mathit{FS}\ a\ b \\
\mathit{new}\ a\ b\ c = \mathit{untar}\ ((a, b) : \mathit{tar}\ c)
\end{array}
$$

Cp

$$
\begin{array}{l}
cp :: (Eq\ a) \Rightarrow Path\ a \rightarrow Path\ a \rightarrow FS\ a\ b \rightarrow FS\ a\ b \\
cp\ a\ e\ c = untar\ \$\ f\ \$\ tar\ c \\
\quad \mathbf{where}\ f\ x \mid null\ (g\ x) = x \\
\qquad \mid otherwise = (e, (\pi_2\ \$\ head\ \$\ g\ x)) : x \\
\qquad g\ lst = \mathbf{if}\ (a \equiv [\,])\ \mathbf{then}\ [\,]\ \mathbf{else}\ filter\ ((elem\ (head\ a)) \cdot \pi_1)\ lst
\end{array}
$$

Rm

$$
\begin{array}{l}
rm :: (Eq\ a) \Rightarrow (Path\ a) \rightarrow (FS\ a\ b) \rightarrow FS\ a\ b \\
rm\ a\ b = untar\ \$\ filter\ (\neg \cdot (\equiv)\ a \cdot \pi_1)\ \$\ tar\ b
\end{array}
$$

auxJoin

$$
\begin{array}{l}
auxJoin :: ([(a, b + c)], d) \rightarrow [(a, b + (d, c))] \\
auxJoin = \bot
\end{array}
$$

cFS2Exp

$$
\begin{array}{l}
cFS2Exp :: a \rightarrow FS\ a\ b \rightarrow (Exp\ ()\ a) \\
cFS2Exp = \bot
\end{array}
$$

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.