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

Departamento de Informática
Universidade do Minho

Junho de 2019

Grupo nr.	008
a84577	José Pedro Silva
a84302	José Ricardo Cunha
a84464	Válter Carvalho

1 Preâmbulo

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

Na sequência pedagógica dos planos de estudo dos dois cursos que têm esta disciplina, restringe-se a aplicação deste método à programação funcional em Haskell. Assim, o presente trabalho prático coloca os alunos perante problemas concretos que deverão ser implementados em Haskell. Há ainda um outro objectivo: o de ensinar a documentar programas, validá-los, e a produzir textos técnico-científicos de qualidade.

2 Documentação

Para cumprir de forma integrada os objectivos enunciados acima vamos recorrer a uma técnica de programação dita "literária" [**?**], cujo princípio base é o seguinte:

> *Um programa e a sua documentação devem coincidir.*

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

O ficheiro `cp1819t.pdf` que está a ler é já um exemplo de programação literária: foi gerado a partir do texto fonte `cp1819t.lhs`[1] que encontrará no material pedagógico desta disciplina descompactando o ficheiro `cp1819t.zip` e executando

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

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

```
$ cabal install lhs2tex
```

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

```
$ ghci cp1819t.lhs
```

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

Abra o ficheiro `cp1819t.lhs` no seu editor de texto preferido e verifique que assim é: todo o texto que se encontra dentro do ambiente

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

vai ser seleccionado pelo GHCi para ser executado.

3 Como realizar o trabalho

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

Recomenda-se uma abordagem participativa dos membros do grupo de trabalho por forma a poderem responder às questões que serão colocadas na *defesa oral* do relatório.

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

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

```
$ bibtex cp1819t.aux
$ makeindex cp1819t.idx
```

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

```
$ cabal install QuickCheck gloss
```

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

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

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

Problema 1

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

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

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

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

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

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

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

   - $calcula :: Expr \rightarrow Int$ 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

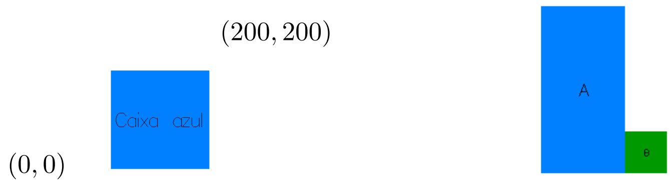

Figura 1: Caixa simples e caixa composta.

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

e onde:

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

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

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

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,

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

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

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

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

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

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

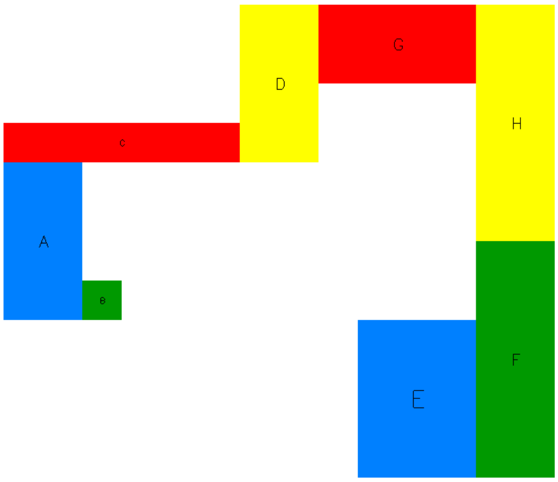

Figura 2: *Layout* feito de várias caixas coloridas.

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

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

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

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

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

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

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

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

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

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

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

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

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

Problema 3

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

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

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

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

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

Problema 4

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

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

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

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

Assumindo estes tipos de dados, o seguinte termo

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

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

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

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

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

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

O que é para fazer:

1. Definir as funções $outFS$, $baseFS$, $cataFS$, $anaFS$ e $hyloFS$.

2. Apresentar, no relatório, o diagrama de $cataFS$.

3. Definir as seguintes funções para manipulação de sistemas de ficheiros usando, obrigatoriamente, catamorfismos, anamorfismos ou hilomorfismos:

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

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

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

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

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

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

(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.*

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

(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.*

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

(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.*

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

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.*

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

(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.*

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

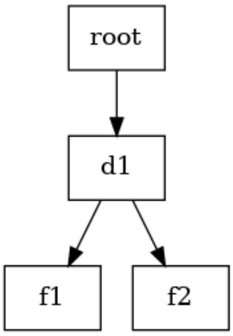

Figura 3: Exemplo de um sistema de ficheiros visualizado em Graphviz.

(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]

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

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

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

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 [?].
[7] Cf. [?], página 102.

C Código fornecido

Problema 1

Tipos:

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

Functor de base:

$$baseExpr\ f\ g = id + (f \times (g \times g))$$

Instâncias:

$$\textbf{instance}\ Read\ Expr\ \textbf{where}$$
$$\quad readsPrec\ \_ = readExp$$

Read para Exp's:

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

Combinadores:

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

```
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:

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

Funções auxiliares:

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

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:

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

Outras funções auxiliares

Lógicas:

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

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

Definições base

$$
\begin{array}{l}
inExpr :: Int + (Op, (Expr, Expr)) \to Expr \\
inExpr = [Num, (\widehat{\cdot} \cdot \widehat{\cdot} \cdot \mathit{flip}\ \$\ Bop) \cdot assocl] \\
outExpr :: Expr \to Int + (Op, (Expr, Expr)) \\
outExpr\ (Num\ x) = i_1\ x \\
outExpr\ (Bop\ a\ b\ c) = i_2\ (b, (a, c)) \\
recExpr\ f = baseExpr\ id\ f \\
cataExpr\ g = g \cdot (recExpr\ (cataExpr\ g)) \cdot outExpr \\
anaExpr\ g = inExpr \cdot (recExpr\ (anaExpr\ g)) \cdot g \\
hyloExpr\ h\ g = cataExpr\ h \cdot anaExpr\ g
\end{array}
$$

Calcula

De modo a calcularmos o valor de uma Expr é apenas necessário fazer um catamorfismo sobre a $Expr$ de modo a obter o resultado da aplicação da Op aos Num's.

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

$$
\begin{array}{l}
calcula :: Expr \to Int \\
calcula = cataExpr\ [id, \widehat{foo}] \\
\quad \mathbf{where} \\
\qquad foo :: Op \to (Int, Int) \to Int \\
\qquad foo\ (Op\ x)\ (a, b) = \mathbf{case}\ x\ \mathbf{of}\ \texttt{"+"} \to a + b \\
\qquad\quad \texttt{"-"} \to a - b \\
\qquad\quad \texttt{"*"} \to a * b \\
\qquad\quad \texttt{"/"} \to a \div b
\end{array}
$$

Show'

A função $show'$ tem um comportamento semelhante ao da função calcula. Aplicamos na mesma um catamorfismo e consoante o tipo da $Expr$ devolvemos uma string.

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

$$
\begin{array}{l}
show' = cataExpr\ [show, \widehat{foo}] \\
\quad \mathbf{where} \\
\qquad foo\ (Op\ y)\ (x, z) = \texttt{"("} + x + \texttt{" "} + y + \texttt{" "} + z + \texttt{")"}
\end{array}
$$

Compile

A função compile tem como objetivo transformar uma *String* em código máquina. De modo a podemos aplicar um morfismo convertemos essa *String* para *Expr*. Após fazermos a conversão para uma *Expr* apenas precisamos de aplicar um catamorfismo onde convertemos uma *Expr* para código máquina.

$$
\begin{tikzcd}
String \arrow[d, "read :: Expr"'] & \\
Expr \arrow[r, "outExpr"] \arrow[d, "compile = (\!|g|\!)"'] & Int + (Op \times (Expr \times Expr)) \arrow[d, "id + (id \times ((\!|g|\!) \times (\!|g|\!)))"] \\
Codigo & Int + (Op \times (Codigo \times Codigo)) \arrow[l, "g = [singl \cdot ((+\!\!+)\ \texttt{"PUSH "} \cdot show), \widehat{foo}]"]
\end{tikzcd}
$$

$$
\begin{aligned}
&compile :: String \to Codigo \\
&compile = cataExpr\ [singl \cdot ((+\!\!+)\ \texttt{"PUSH "} \cdot show), \widehat{foo}] \cdot f \\
&\quad \mathbf{where} \\
&\qquad f\ x = read\ x :: Expr \\
&\qquad foo\ (Op\ x)\ (a, b) = \mathbf{case}\ x\ \mathbf{of}\ \texttt{"+"} \to a +\!\!+ b +\!\!+ \texttt{"ADD"} : [\,] \\
&\qquad\quad \texttt{"*"} \to a +\!\!+ b +\!\!+ \texttt{"MUL"} : [\,] \\
&\qquad\quad \texttt{"-"} \to a +\!\!+ b +\!\!+ \texttt{"MIN"} : [\,] \\
&\qquad\quad \texttt{"/"} \to a +\!\!+ b +\!\!+ \texttt{"DIV"} : [\,]
\end{aligned}
$$

Problema 2

Definições base

$$
\begin{aligned}
&inL2D :: a + (b, (X\ a\ b, X\ a\ b)) \to X\ a\ b \\
&inL2D = [Unid, (\widehat{\cdot} \cdot \widehat{\cdot}\ \$\ Comp) \cdot assocl] \\
&outL2D :: X\ a\ b \to a + (b, (X\ a\ b, X\ a\ b)) \\
&outL2D\ (Unid\ a) = i_1\ a \\
&outL2D\ (Comp\ b\ x\ y) = i_2\ (b, (x, y)) \\
&baseL2D\ f\ g = id + (f \times (g \times g)) \\
&recL2D\ f = baseL2D\ id\ f \\
&cataL2D\ g = g \cdot (recL2D\ (cataL2D\ g)) \cdot outL2D \\
&anaL2D\ g = inL2D \cdot (recL2D\ (anaL2D\ g)) \cdot g
\end{aligned}
$$

CollectLeafs

Ao analisarmos o tipo *L2D* vemos que este se assemelha a uma LTree e por isso só contém informação nas folhas. A função *collectLeafs* devolve todos os valores das folhas.

$$collectLeafs = cataL2D\ [singl, \widehat{(+\!\!+)} \cdot \pi_2]$$

Dimen

A função $dimen$ tem como objetivo calcular as dimensões do retângulo final que engloba todas as caixas. De modo a calcular essa dimensão usamos um catamorfismo que dependendo do tipo de agregação calcula a nova dimensão.

$$\begin{tikzcd}
X\ Caixa\ Tipo \arrow[rr, "outL2D"] \arrow[d, "dimen=(\!|g|\!)"'] & & Caixa + (Tipo \times (X\ Caixa\ Tipo)^2) \arrow[d, "id+(id\times(\!|g|\!)^2)"] \\
(Float, Float) & & Caixa + (Tipo \times (Float, Float)^2) \arrow[ll, "g={[}(fromIntegral\times fromIntegral)\cdot\pi_1, mysum{]}"]
\end{tikzcd}$$

$$\begin{array}{l}
dimen :: X\ Caixa\ Tipo \to (Float, Float) \\
dimen = cataL2D\ [(fromIntegral \times fromIntegral) \cdot \pi_1, mysum] \\
\quad \mathbf{where} \\
\qquad mysum\ (t, ((a, b), (x, y))) = \mathbf{case}\ t\ \mathbf{of} \\
\qquad\quad V \to (max\ a\ (a + x\ /\ 2), b + y) \\
\qquad\quad Vd \to (max\ a\ (a + x), b + y) \\
\qquad\quad Ve \to (max\ a\ x, b + y) \\
\qquad\quad H \to (a + x, max\ b\ (b + y\ /\ 2)) \\
\qquad\quad Hb \to (a + x, max\ b\ y) \\
\qquad\quad Ht \to (a + x, max\ b\ (b + y))
\end{array}$$

calc

Função auxiliar que recebe o $Tipo$ de agregação, a $Origem$ e as $dimens\tilde{o}es$ da caixa e calcula a sua nova origem.

$$\begin{array}{l}
calc :: Tipo \to Origem \to (Float, Float) \to Origem \\
calc\ t\ o\ f = calcAux\ t\ \$\ \langle \pi_1 \times \pi_1, \pi_2 \times \pi_2 \rangle\ (o, f) \\
\quad \mathbf{where} \\
\qquad calcAux\ V = (\widehat{(+)} \cdot (id \times (/2))) \times \widehat{(+)} \\
\qquad calcAux\ Vd = \widehat{(+)} \times \widehat{(+)} \\
\qquad calcAux\ Ve = \pi_1 \times \widehat{(+)} \\
\qquad calcAux\ H = \widehat{(+)} \times (\widehat{(+)} \cdot (id \times (/2))) \\
\qquad calcAux\ Ht = \widehat{(+)} \times \widehat{(+)} \\
\qquad calcAux\ Hb = \widehat{(+)} \times \pi_1
\end{array}$$

calcOrigins

Função que recebendo um $(X\ Caixa\ Tipo, Origem)$ calcula a origem para todas as caixas.

$$\begin{tikzcd}
(X\ Caixa\ Tipo, Origem) \arrow[rr, "g"] \arrow[d, "calcOrigins=[\!(g)\!]"'] & & (Caixa, Origem) + (() \times ((X\ Caixa\ Tipo), Origem)^2) \arrow[d, "id+(id\times[\!(g)\!]^2)"] \\
X\ (Caixa, Origem)\ () & & (Caixa, Origem) + (() \times (X\ (Caixa, Origem)\ ())^2) \arrow[ll, "inL2D"]
\end{tikzcd}$$

$$
\begin{array}{l}
calcOrigins :: ((X\ Caixa\ Tipo), Origem) \rightarrow X\ (Caixa, Origem)\ ()\\
calcOrigins = anaL2D\ g\\
\quad \mathbf{where}\\
\qquad g\ (Unid\ x, o) = i_1\ (x, o)\\
\qquad g\ (Comp\ t\ x\ y, o) = \mathbf{let}\ dimensao = dimen\ x\\
\qquad\quad \mathbf{in}\ i_2 \cdot (,)\ ()\ \$\ (,)\ (x, o)\ (y, calc\ t\ o\ dimensao)
\end{array}
$$

agrupCaixas

A função $agrupCaixas$ recebe um $X\ (Caixa, Origem)\ ()$ e transforma numa Fig. Como o input é um $X\ (Caixa, Origem)\ ()$ podemos aplicar um catamorfismo. Sabendo que **type** $Fig = [(Origem, Caixa)]$ esta função fica bastante simples pois apenas precisamos de trocar a ordem de $(Caixa, Origem)$ para $(Origem, Caixa)$.

$$
\begin{tikzcd}
X\ (Caixa, Origem)\ () \arrow[r, "outL2D"] \arrow[d, "agrup\_Caixas = (\![g]\!)"'] & (Caixa, Origem) + (() \times (X\ (Caixa, Origem)\ ())^2) \arrow[d, "id + (id \times (\![g]\!)^2)"] \\
Fig & (Caixa, Origem) + (() \times Fig^2) \arrow[l, "g = [singl \cdot swap, \widehat{(+\!\!+)} \cdot \pi_2]"]
\end{tikzcd}
$$

$$
\begin{array}{l}
agrup\_caixas :: X\ (Caixa, Origem)\ () \rightarrow Fig\\
agrup\_caixas = cataL2D\ [singl \cdot swap, \widehat{(+\!\!+)} \cdot \pi_2]
\end{array}
$$

caixasAndOrigin2Pict

De modo a transformarmos um par $(X\ Caixa\ Tipo, Origem)$ numa $G.picture$ usamos a função $caixasAndOrigin2Pict$. A função começa por calcular as origens de todas as caixas ($calcOrigins$), após isso usa a $collectLeafs$ para obter o conteúdo das folhas. Após executarmos essas 2 funções temos agora uma lista de $(Caixa, Origem)$. De modo a criarmos uma $G.picture$ aplicamos a cada elemento da lista a função $crCaixa$ e seguidamente concatenamos com a função $G.pictures$.

$$
\begin{array}{l}
caixasAndOrigin2Pict = G.pictures \cdot \mathsf{fmap}\ func \cdot collectLeafs \cdot calcOrigins\\
\quad \mathbf{where}\\
\qquad func = \lambda((x, (i, j)), o) \rightarrow crCaixa\ o\ (f\ x)\ (g\ x)\ i\ j\\
\qquad (f, g) = (fromIntegral \cdot \pi_1, fromIntegral \cdot \pi_2)
\end{array}
$$

mostra_caixas

Esta função transforma um par $(L2D, Origem)$ num $IO\ ()$, isto é, pega no par e dá display desta imagem graficamente.

$$
\begin{array}{l}
mostra\_caixas :: (L2D, Origem) \rightarrow \mathsf{IO}\ ()\\
mostra\_caixas = display \cdot caixasAndOrigin2Pict
\end{array}
$$

Problema 3

Através da fórmula que dá a série de Taylor da função cosseno conseguimos deduzir que:

$$e\ x\ 0 = 1$$
$$e\ x\ (n+1) = cos'\ x\ n + h\ x\ n$$
$$h\ x\ 0 = -x^2\ /\ 2$$
$$h\ x\ (n+1) = h\ x\ n * (-x^2\ /\ s\ n)$$
$$s\ 0 = 12$$
$$s\ (n+1) = s\ n + j\ n$$
$$j\ 0 = 18$$
$$j\ (n+1) = j\ n + 8$$

Solução:

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

Valorização

O mesmo resultado mas em código C.

$$\begin{array}{l} double\ cos'\ (double\ x, double\ n)\ \{ \\ \quad double\ e = 1; double\ h = -x^2\ /\ 2; \\ \quad double\ s = 12; double\ j = 18; \\ \quad int\ i; \\ \quad \mathsf{for}\ i = 0; i < n+1; i +\!\!+\ \{ \\ \quad\quad e = e + h; \\ \quad\quad h = h * (-x^2)\ /\ s; \\ \quad\quad s = s + j; \\ \quad\quad j = j + 8; \\ \quad \} \\ \quad return\ e; \\ \} \end{array}$$

Problema 4

Triologia "ana-cata-hilo":

$$outFS\ (FS\ l) = \mathsf{fmap}\ (id \times outNode)\ l$$
$$outNode\ (File\ b) = i_1\ b$$
$$outNode\ (Dir\ x) = i_2\ x$$
$$baseFS\ f\ g\ h = \mathsf{fmap}\ (bimap\ f\ (g+h))$$
$$cataFS :: ([(a, b+c)] \rightarrow c) \rightarrow FS\ a\ b \rightarrow c$$
$$cataFS\ g = g \cdot (recFS\ (cataFS\ g)) \cdot outFS$$
$$anaFS :: (c \rightarrow [(a, b+c)]) \rightarrow c \rightarrow FS\ a\ b$$
$$anaFS\ g = inFS \cdot (recFS\ (anaFS\ g)) \cdot g$$
$$hyloFS\ g\ h = cataFS\ g \cdot anaFS\ h$$

Diagrama geral cataFS

$$\begin{tikzcd} FS\ a\ b \arrow[r, "outFS"] \arrow[d, "f = (\![g]\!)"'] & (a \times (b + FS\ a\ b))^* \arrow[d, "\mathsf{fmap}\ (id \times (id + (\![g]\!)))"] \\ C & (a \times (b + C))^* \arrow[l, "g"] \end{tikzcd}$$

Outras funções pedidas:

check

O objetivo da função $check$ é verificar que não existem identificadores repetidos dentro da mesma diretoria. Para isso percorre-se o $File\ \ System$ e verifica-se em todas as diretorias do mesmo se existem identificadores iguais, caso isso se verifique a função $check$ retornará $False$.

$$\begin{tikzcd}
FS\ a\ b \arrow[r, "outFS"] \arrow[d, "check=(\!|g|\!)"'] & (a \times (b + FS\ a\ b))^* \arrow[d, "\mathsf{fmap}\ (id\times(id+(\!|g|\!)))"] \\
Bool \arrow[d, "\neg"'] & (a \times (b + Bool))^* \arrow[l, "g=\pi_2\cdot foldr\ foo\ ([\,],False)"] \\
Bool &
\end{tikzcd}$$

$check :: (Eq\ a) \Rightarrow FS\ a\ b \rightarrow Bool$
$check = \neg \cdot cataFS\ (\pi_2 \cdot foldr\ foo\ ([\,], False))$
  where
    $foo\ (a, (i_2\ b))\ acc@(x, y) = \textbf{if}\ y\ \textbf{then}\ acc\ \textbf{else}\ (a : x, y \vee b \vee elem\ a\ x)$
    $foo\ (a, \_)\ acc@(x, y) = \textbf{if}\ y\ \textbf{then}\ acc\ \textbf{else}\ (a : x, y \vee elem\ a\ x)$

tar

A função tar tem como objetivo recolher todos os ficheiros de um $File\ \ System$ para um lista. Estes são indexados juntamente com o seu respetivo $Path$. De modo a percorrer o $File\ \ System$ usamos um catamorfismo e vamos adicionando todos os ficheiros à lista.

$$\begin{tikzcd}
FS\ a\ b \arrow[r, "outFS"] \arrow[d, "tar=(\!|g|\!)"'] & (a \times (b + FS\ a\ b))^* \arrow[d, "\mathsf{fmap}\ (id\times(id+(\!|g|\!)))"] \\
Bool & (a \times (b + Bool))^* \arrow[l, "g=concatMap\ foo"]
\end{tikzcd}$$

$tar :: FS\ a\ b \rightarrow [(Path\ a, b)]$
$tar = cataFS\ (concatMap\ foo)$
  where
    $foo\ (a, i_1\ b) = [([a], b)]$
    $foo\ (a, i_2\ l) = \mathsf{fmap}\ (((+\!\!+)\ [a]) \times id)\ l$

untar

A função $untar$ faz o inverso da função tar desenvolvida anteriormente. De modo a transformar um $(Path\ a, b)^*$ num $File\ System$ usamos um anamorfismo em que o objetivo é colocar cada ficheiro na sua respetiva diretoria. No final usamos a função $joinDupDirs$ de modo a evitar diretorias duplicadas.

$$\begin{tikzcd}
(Path\ a, b)^* \arrow[r, "g=\mathsf{fmap}\ foo"] \arrow[d, "untar=[\!(g)\!]"'] & (a \times (b + (Path\ a \times b)^*))^* \arrow[d, "\mathsf{fmap}\ (id\times(id+[\!(g)\!]))"] \\
FS\ a\ b \arrow[d, "joinDupDirs"'] & (a \times (b + FS\ a\ b))^* \arrow[l, "inFs"] \\
FS\ a\ b &
\end{tikzcd}$$

$$
\begin{array}{l}
\mathit{untar} :: (\mathit{Eq}\ a) \Rightarrow [(\mathit{Path}\ a, b)] \rightarrow \mathit{FS}\ a\ b\\
\mathit{untar} = \mathit{joinDupDirs} \cdot \mathit{anaFS}\ (\mathsf{fmap}\ \mathit{foo})\\
\quad \textbf{where}\\
\qquad \mathit{foo}\ (a : [\,], b) = (a, i_1\ b)\\
\qquad \mathit{foo}\ (a : t, b) = (a, i_2\ [(t, b)])
\end{array}
$$

find

A função *find* tem como objetivo retornar o Path de um ficheiro no $\mathit{File\ System}$. Neste exercício usamos 2 catamorfismos, um para filtrar aqueles que tem o identificador igual e outro para juntar esses identicadores todos.

$$
\begin{tikzcd}
\mathit{FS}\ a\ b \arrow[r, "\mathit{outFS}"] \arrow[d, "f"'] & (a \times (b + \mathit{FS}\ a\ b))^* \arrow[d, "\mathsf{fmap}\ (id \times (id + (\!|g|\!)))"] \\
\mathit{FS}\ a\ b \arrow[d, "f'"] & (a \times (b + \mathit{FS}\ a\ b))^* \arrow[l, "g = \mathit{inFS} \cdot \mathit{foo}"] \arrow[d, "\mathsf{fmap}\ (id \times (id + (\!|g|\!)))"] \\
\mathit{FS}\ a\ b & (a \times (b + \mathit{FS}\ a\ b))^* \arrow[l, "g = \mathit{concatMap}\ \mathit{goo}"]
\end{tikzcd}
$$

$$
\begin{array}{l}
\mathit{find} :: (\mathit{Eq}\ a) \Rightarrow a \rightarrow \mathit{FS}\ a\ b \rightarrow [\mathit{Path}\ a]\\
\mathit{find}\ a = \mathit{cataFS}\ (\mathit{concatMap}\ \mathit{goo}) \cdot \mathit{cataFS}\ (\mathit{inFS} \cdot \mathit{foo})\\
\quad \textbf{where}\\
\qquad \mathit{foo} = \mathit{filter}\ (\lambda(x, y) \rightarrow x \equiv a)\\
\qquad \mathit{goo}\ (c, i_1\ \_) = [[c]]\\
\qquad \mathit{goo}\ (c, i_2\ l) = \mathsf{fmap}\ ((+\!\!+)\ [c])\ l
\end{array}
$$

new

A função new é utilizada para criar um ficheiro dentro de um $\mathit{File\ System}$. De modo a fazer isto usamos um anamorfismo para percorrer o $\mathit{File\ System}$ e quando estamos na diretoria destino do ficheiro criamos o ficheiro adicionando-o ao $\mathit{File\ System}$.

$$
\begin{tikzcd}
((\mathit{Path}\ a \times b) \times \mathit{FS}\ a\ b) \arrow[r, "g"] \arrow[d, "\mathit{new} = [\!(g)\!]"'] & (a \times (b + ((\mathit{Path}\ a \times b) \times \mathit{FS}\ a\ b)))^* \arrow[d, "\mathsf{fmap}\ (id \times (id + [\!(g)\!]))"] \\
\mathit{FS}\ a\ b & (a \times (b + \mathit{FS}\ a\ b))^* \arrow[l, "\mathit{inFS}"]
\end{tikzcd}
$$

$$
\begin{array}{l}
\mathit{new} :: (\mathit{Eq}\ a) \Rightarrow \mathit{Path}\ a \rightarrow b \rightarrow \mathit{FS}\ a\ b \rightarrow \mathit{FS}\ a\ b\\
\mathit{new} = \mathit{curry} \cdot \mathit{curry} \cdot \mathit{anaFS}\ \$\ g\\
\quad \textbf{where}\\
\qquad g = (\mathit{cond}\ ((>1) \cdot \mathsf{length}\ \cdot \pi_1 \cdot \pi_1)\ \mathit{skip}\ \mathit{newCond}) \cdot ((id \times id) \times (\mathsf{fmap}\ \mathit{distr} \cdot \mathit{outFS}))\\
\qquad \mathit{newCond} = \mathit{cond}\ ((\equiv 1) \cdot \mathsf{length}\ \cdot \pi_1 \cdot \pi_1)\ \mathit{addFile}\ \mathit{skip2}\\
\qquad \mathit{addFile} = \mathsf{conc} \cdot \langle \mathit{singl} \cdot (\mathit{head}_1) \cdot \pi_1, (\mathsf{fmap} \cdot \widehat{\mathit{curry}\ ([\mathit{f1}, \mathit{f2}]} \cdot \mathit{distr})) \rangle\\
\qquad \mathit{skip} = (\mathsf{fmap} \cdot \widehat{\mathit{curry}\ ([\mathit{f1}, \mathit{func}]} \cdot \mathit{distr}))\\
\qquad \mathit{skip2} = (\mathsf{fmap} \cdot \widehat{\mathit{curry}\ ([\mathit{f1}, \mathit{f2}]} \cdot \mathit{distr}))\\
\qquad \mathit{f1} = (id_1) \cdot \pi_2\\
\qquad \mathit{f2} = \langle \pi_1 \cdot \pi_2, i_2 \cdot ((\mathit{nil} \times id) \times \pi_2) \rangle\\
\qquad \mathit{func} = \mathit{cond}\ (\widehat{(\equiv)} \cdot ((\mathit{head} \cdot \pi_1) \times \pi_1))\ \mathit{r1}\ \mathit{f2}\\
\qquad \mathit{r1} = \langle \pi_1 \cdot \pi_2, i_2 \cdot ((\mathit{tail} \times id) \times \pi_2) \rangle
\end{array}
$$

Questão

No caso de substituirmos a propriedade *checkFiles* pela propriedade mais fraca *check* a prop_new deixa de ser válida no caso de tentarmos criar um ficheiro com um identificador igual a uma diretoria, pois esta não permite que tal aconteça.

cp

A função *cp* é utilizada para copiar um ficheiro existente num *File System*. De modo a fazer isto usamos um anamorfismo para percorrer o *File System* e quando estamos na diretoria do ficheiro em questão, copiamos o seu conteúdo para o seu novo *Path*. A inserção do ficheiro no seu novo *Path* é feita recorrendo à função previamente desenvolvida *new*.

$$
\begin{tikzcd}
((Path\ a) \times FS\ a\ b) \arrow[r, "g"] \arrow[d, "(\!|g|\!)"'] & (a \times (b + ((Path\ a) \times FS\ a\ b)))^* \arrow[d, "\mathsf{fmap}\ (id \times (id + (\!|g|\!)))"] \\
FS\ a\ b \arrow[d] & (a \times (b + FS\ a\ b))^* \arrow[l, "inFS"] \\
(((Path)\ a \times FS\ a\ b) \times FS\ a\ b) \arrow[d, "new"'] & \\
FS\ a\ b &
\end{tikzcd}
$$

$$
\begin{array}{l}
cp :: (Eq\ a) \Rightarrow Path\ a \rightarrow Path\ a \rightarrow FS\ a\ b \rightarrow FS\ a\ b \\
cp\ pa\ pb\ fs = maybeAdd\ pb\ fs\ \$\ anaFS\ g\ (pa, fs) \\
\quad \mathbf{where} \\
\qquad maybeAdd\ pb\ fs\ (FS\ x) = \mathbf{if}\ null\ x\ \mathbf{then}\ fs\ \mathbf{else}\ new\ pb\ ((\lambda(File\ b) \rightarrow b) \cdot \pi_2\ \$\ head\ x)\ fs \\
\qquad g\ (a, FS\ l) = \mathbf{if}\ null\ a\ \mathbf{then}\ empty\ \mathbf{else}\ foo\ a\ (aux\ a\ l) \\
\qquad aux\ a\ l = filter\ (\lambda z@(x, y) \rightarrow (head\ a) \equiv x)\ l \\
\qquad foo\ a\ l = \mathsf{fmap}\ (\lambda fs \rightarrow \mathbf{case}\ fs\ \mathbf{of}\ (x, File\ f) \rightarrow (x, i_1\ f) \\
\qquad\quad (x, Dir\ f) \rightarrow (x, i_2\ (tail\ a, f)))\ l
\end{array}
$$

rm

A função *rm* é utilizada para remover um ficheiro de um *File System*. Para alcançar esse objetivo usamos um anamorfismo que vai percorrendo o *File System* e quando encontra o ficheiro em questão, isto é para remover, não o escreve no *File System*.

$$
\begin{tikzcd}
((Path\ a) \times FS\ a\ b) \arrow[r, "g"] \arrow[d, "rm = (\!|g|\!)"'] & (a \times (b + ((Path\ a) \times FS\ a\ b)))^* \arrow[d, "\mathsf{fmap}\ (id \times (id + (\!|g|\!)))"] \\
FS\ a\ b & (a \times (b + FS\ a\ b))^* \arrow[l, "inFS"]
\end{tikzcd}
$$

$$
\begin{array}{l}
rm :: (Eq\ a) \Rightarrow (Path\ a) \rightarrow (FS\ a\ b) \rightarrow FS\ a\ b \\
rm = curry \cdot anaFS\ \$\ g \\
\quad \mathbf{where} \\
\qquad g = (cond\ ((>1) \cdot \mathsf{length}\ \cdot \pi_1)\ skip\ newCond) \cdot (id \times (\mathsf{fmap}\ distr)) \cdot (id \times outFS) \\
\qquad newCond = cond\ ((\equiv 1) \cdot \mathsf{length}\ \cdot \pi_1)\ remove\ skip2 \\
\qquad remove = \mathsf{fmap}\ [id_1, id \times (i_2 \cdot \langle nil, id \rangle)] \cdot (filter \cdot \widehat{curry}\ func) \\
\qquad func = [\widehat{(\not\equiv)} \cdot (head \times \pi_1), \underline{True}] \cdot distr \\
\qquad skip = (\mathsf{fmap} \cdot \widehat{curry\ ([f1}, f2] \cdot distr))
\end{array}
$$

$$
\begin{aligned}
&(f1, f2) = ((id_1) \cdot \pi_2, cond\ (\widehat{(\equiv)} \cdot (head \times \pi_1))\ r1\ r2)\\
&(r1, r2) = (\langle \pi_1 \cdot \pi_2, i_2 \cdot (tail \times \pi_2)\rangle, \langle \pi_1 \cdot \pi_2, i_2 \cdot (nil \times \pi_2)\rangle)\\
&skip2 = (\mathsf{fmap} \cdot curry\ \widehat{([f1, aux] \cdot distr)})\\
&aux = \langle \pi_1, i_2 \cdot (nil \times id)\rangle \cdot \pi_2
\end{aligned}
$$

auxJoin

A função $auxJoin$ é usada para a manipulação dos argumentos de um par $((a, b + c)^*, d)$. Como podemos ver pela assinatura da função o tipo produzido $(a, b + (d, b))^*$.

$$
\begin{array}{l}
auxJoin :: ([(a, b + c)], d) \rightarrow [(a, b + (d, c))] \\
auxJoin = \mathsf{fmap}\ g \cdot \widehat{zip} \cdot (id \times repeat) \\
\quad \mathbf{where} \\
\qquad g = (id \times [i_1 \cdot \pi_1, i_2 \cdot swap]) \cdot (id \times distl) \cdot assocr
\end{array}
$$

Valorização

A função $cFS2Exp$ permite visualizar um $File\ System$ em $Graphviz$. Para isso usamos um anamorfismo que percorre o $File\ System$.

$$
\begin{array}{l}
cFS2Exp :: a \rightarrow FS\ a\ b \rightarrow (Exp\ ()\ a) \\
cFS2Exp = curry \cdot anaExp\ \$\ g \\
\quad \mathbf{where} \\
\qquad g\ (a, FS\ l) = i_2 \cdot (,)\ a\ \$\ \mathsf{fmap}\ func\ l \\
\qquad func = \lambda(x, y) \rightarrow \mathbf{case}\ y\ \mathbf{of}\ File\ f \rightarrow (x, FS\ [\,]); Dir\ f \rightarrow (x, f)
\end{array}
$$