Tipos abstratos de dados

5.1 Escreva uma função *parent* :: *String* → *Bool* que verifique se uma cadeia de caracteres é uma sequência de parêntesis curvos e rectos correctamente emparelhados; por exemplo:

$$\textit{parent}\ \texttt{"((((()[()]))))"} = \textit{True} \qquad \textit{parent}\ \texttt{"((][))"} = \textit{False}$$

Sugestão: represente parêntesis abertos usando uma pilha dos caracteres '`(`', '`[`' e '`{`'; utilize o módulo *Stack* apresentado na aula teórica.

5.2 Na *notação polaca invertida* (abreviado para *RPN* do inglês "*reverse polish notation*") colocamos cada operador binário após os dois operandos; por exemplo, a expressão $42 \times 3 + 1$ escreve-se "`42 3 * 1 +`". Nesta notação não necessitamos de parêntesis ou de precedências entre operadores.

Pretende-se escrever uma função para calcular o valor de uma expressão em RPN; este cálculo pode ser feito percorrendo a expressão uma só vez usando uma pilha para guardar valores intermédios.

(a) Escreva uma função auxiliar *calc* :: *Stack Float* → *String* → *Stack Float* que implemente uma operação (se o 2º argumento for "+", "*", "-" ou "/" ou coloque um operando na pilha (se o 2º argumento for um numeral); o resultado deve ser a pilha modificada.
Sugestão: utilize a função *read* :: *String* → *Float* do prelúdio-padrão para converter um número em texto para vírgula flutuante.

(b) Usando a função anterior e o módulo *Stack* apresentados nas aulas teóricas, escreva a função *calcular* :: *String* → *Float* que calcula o valor duma expressão em RPN; por exemplo: *calcular* "42 3 * 1 +" = 127.
Sugestão: utilize a função *words* :: *String* → [*String*] do prelúdio-padrão para partir uma cadeia de caracteres em palavras.

(c) Escreva um programa principal que leia uma expressão em RPN da entrada padrão como uma cadeia de caracteres da entrada padrão e calcule o seu valor. Experimente o programa com expressões correctas e incorrectas e interprete os resultados.

5.3 Implemente o tipo abstracto de dados que define *conjuntos* usando listas sem elementos repetidos.

5.4 Considere o tipo abstrato *Set a* para conjuntos finitos de valores de tipo *a* com as seguintes operações:

$$\begin{aligned}
&\textit{empty} :: \textit{Set}\ a\\
&\textit{insert} :: \textit{Ord}\ a \Rightarrow a \to \textit{Set}\ a \to \textit{Set}\ a\\
&\textit{member} :: \textit{Ord}\ a \Rightarrow a \to \textit{Set}\ a \to \textit{Bool}
\end{aligned}$$

Escreva uma implementação deste tipo usando árvores binárias de pesquisa simples.

5.5 Considere as operações de união, interseção e diferença entre conjuntos; todas estas operações têm o mesmo tipo:

$$union, intersect, difference :: Ord\ a \Rightarrow Set\ a \rightarrow Set\ a \rightarrow Set\ a$$

Acrescente estas operações à implementação que fez para o exercício anterior.

5.6 Considere o tipo abstrato $Map\ k\ a$ para associações entre chaves de tipo k e valores de tipo a com as seguintes operações:

$$\begin{aligned} &empty :: Map\ k\ a \\ &insert :: Ord\ k \Rightarrow k \rightarrow a \rightarrow Map\ k\ a \rightarrow Map\ k\ a \\ &lookup :: Ord\ k \Rightarrow k \rightarrow Map\ k\ a \rightarrow Maybe\ a \end{aligned}$$

Escreva uma implementação deste tipo abstrato usando árvores binárias de pesquisa simples.

5.7 O tipo abstracto para conjuntos apresentado na aula teórica está limitado a representar conjuntos finitos. Podemos representar conjuntos infinitos computacionalmente usando funções: um conjunto de valores de tipo a é uma função $a \rightarrow Bool$, i.e. um *predicado*. Por exemplo, o conjunto de todos os inteiros pares pode ser representado pela função $\lambda x \rightarrow x `mod` 2 == 0$ de tipo $Integer \rightarrow Bool$.

Modifique as operações do tipo abstracto para permitir conjuntos infinitos e implemente-o usando esta representação.