Linguagens e Ambientes de Programação (2018/2019)

Prática 02

Funções e listas. Exercícios de 13 a 15.

- 13 - Infira o tipo de cada uma das seguintes funções:

```
let f1 x = x + 1 ;;
let f2 x = f1 x ;;
let f3 x = 1 + x 5 ;;
let f4 x y = x < y x ;; (* Procure o tipo de "<" aqui *)
```

Nota: Na resolução de problemas de inferência, o primeiro passo é sempre ver quantos argumentos tem a função - o tipo duma função com n argumentos tem sempre n setas no nível exterior. O segundo passo é descobrir o tipo de cada argumento com base na sua utilização do lado direito da função. O tipo do resultado pode ser descoberto no final do processo, ou misturado com a descoberta do tipo dos argumentos. Exemplos.

Para cada um dos tipos abaixo, dê também um exemplo duma função:

```
(int->int) -> int
bool -> float -> string
```

Ambiente de trabalho usando o IDE Eclipse com o plugin OcaIDE instalado

Apresenta-se a forma recomendada para trabalhar com a linguagem OCaml nas próximas aulas práticas. Explica-se apenas o uso do interpretador. A parte do compilador fica para mais tarde.

Durante a aula, mantenha duas janelas abertas:

- Uma com um Web browser ativo, para acesso à página de LAP, onde se encontra o enunciado dos exercícios;
- Outra com o Eclipse aberto para editar e correr programas em OCaml.

O processo de criação dum projeto novo para OCaml é o seguinte:

1. Da primeira vez que se usa o Eclipse, é necessário escolher explicitamente a perspetiva "O'Caml". Faz-se desta maneira, usando os menus:
   - Window > Open Perspective > Other ... escolher "O'Caml" e carregar no botão OK.
2. Agora cria-se o projeto. Faz-se da seguinte maneira, usando os menus:
   - File > New > OCaml Project (ocamlbuild) ... escolher o Project name "LAP" e carregar no botão FINISH.
3. Agora cria-se um módulo de trabalho. Faz-se desta maneira, usando os menus:
   - File > New > Module ... escolher o File name "aula2" e carregar no botão FINISH.
4. Está ponto a usar. Escreva algumas funções e expressões na janela de edição "aula2.ml". Para testar o código, basta selecionar as partes pretendidas e carregar na tecla F6. Os resultados aparecem na view "OCaml Toplevel".
5. Repare que a view "OCaml Toplevel está dividida em duas partes. Também é possível introduzir funções e expressões para avaliar através do painel de baixo. Nesse caso é preciso carregar na tecla "Return" quando o cursor estiver colocado imediatamente a seguir à sequência ";;", no final da linha.
6. Para aceder ao manual completo deste plugin, dentro do Eclipse siga: Help > Help Contents > OCaml Development User Guide.

Depois da criação do Project "LAP", nas execuções seguintes o Eclipse arranca com esse Project já aberto. Se desejar que os exercícios das diversas aulas não fiquem misturados no mesmo módulo, crie um módulo novo no início de cada aula.

Estas regras ensinam a usar apenas o interpretador. Na aula prática 4 veremos como gerar programas standalone executáteis.

Instalação: Já está instalado nos laboratórios. Para instalar no seu computador pessoal, primeiro instala-se o sistema OCaml, que pode ser obtido na página da bibliografia. Depois instala-se o Eclipse. Finalmente instala-se o plugin. Na página WEB do plugin há regras de instalação fáceis de seguir.

Questões de compatibilidade: O plugin para OCaml funciona bem com a maioria das combinações de Eclipse/Java 8, desde que a versão do Eclipse seja suficientemente recente. Recomenda-se o uso de Eclipse Neon, ou qualquer outro Eclipse mais recente como o Eclipse 2018-12. Em versões mais antigas do Eclipse, existem problemas. Por exemplo, na combinação Eclipse Luna/Java 8 funciona o interpretador mas não o compilador.

- 14 - Escreva em OCaml uma função

```
succAll : int list -> int list
```

que produza a lista dos sucessores duma lista de inteiros.

Exemplos:
```
succAll [] = []
succAll [3; 6; 1; 0; -4] = [4; 7; 2; 1; -3]
```

NOTA: Este exercício será resolvido no quadro e a solução servirá de orientação para a resolução do problema 15.

- 15 - Implemente o tipo de dados conjunto. Para representar os conjuntos, use listas não ordenadas mas sem repetições. As funções pretendidas são as seguintes:

```
belongs: 'a -> 'a list -> bool       (* teste de pertença *) (* para ser resolvido no quadro *)
union: 'a list -> 'a list -> 'a list (* união de conjuntos *)
inter: 'a list -> 'a list -> 'a list (* intersecção de conjuntos *)
diff: 'a list -> 'a list -> 'a list  (* diferença de conjuntos *)
power: 'a list -> 'a list list       (* conjunto potência: conjunto dos subconjuntos - difícil *)
```

Exemplos:
```
belongs 4 [1;2;3;4;5;6] = true
belongs 4 [1;2] = false
union [7;3;9] [2;1;9] = [7;3;2;1;9]
inter [7;3;9] [2;1;9] = [9]
diff [7;3;9] [2;1;9] = [7;3]
power [] = [[]]
power [2;3] = [[]; [2]; [3]; [2;3]]
power [1;2;3] = [[]; [2]; [3]; [2;3]; [1]; [1;2]; [1;3]; [1;2;3]] (* não interessa a ordem dos elementos *)
```

Nota: Nas funções que recebem duas listas pode surgir a dúvida: faz-se a redução a problemas mais simples no primeiro ou no segundo argumentos da função? Geralmente tenta fazer-se a redução usando o primeiro argumento e chega-se a uma solução satisfatória. Apenas em casos raros é necessário fazer a redução no segundo argumento, ou até nos dois argumentos ao mesmo tempo.

- 16 - Escreva em OCaml uma função

```
nat : int -> int list
```

que, dado um inteiro n, gere a lista dos n primeiros números naturais, ordenada decrescentemente.

Exemplos:
```
nat 0 = []
nat 1 = [0]
nat 2 = [1;0]
nat 5 = [4;3;2;1;0]
```

Nota: Este exercício tem a ver com listas, mas repare que o que determina a estrutura duma função são os seus argumentos. Como o argumento desta função é um inteiro, é provável que esta função tenha uma estrutura parecida com a da função fatorial, do exercício 6.

- 17 - O Departamento de Engenharia do Ambiente possui um dispositivo que regista a temperatura ambiente segundo a segundo. No final de cada ciclo de 24 horas, o dispositivo envia para um computador a lista dos 86400 valores reais que representam as amostras do dia.

Mas a temperatura ambiente varia lentamente. Assim é de esperar que uma lista de temperaturas contenha muitos valores repetidos consecutivos. Analisemos a parte inicial duma tal lista:

```
[10.1; 10.1; 10.1; 10.1; 10.1; 10.1; 10.1; 10.1; 10.1; 10.1; 10.1;
 10.1; 10.1; 10.1; 10.1; 10.1; 10.1; 10.1; 10.1; 10.1; 10.1; 10.1;
 10.1; 10.1; 10.1; 10.1; 10.0; 10.0; 10.0; 10.0; 10.0; ...]
```

Esta lista indica que à meia-noite a temperatura era de 10.1 graus centígrados. Depois, a temperatura não sofreu alteração sensível durante mais 24 segundos, mas ao 26º segundo baixou para os 10.0 graus centígrados [Excitante, não é?!].

O que nos interessa aqui é a questão do armazenamento destas listas de temperaturas no computador. A verdade é que não vale a pena guardar listas tão longas. É preferível guardá-las numa forma compactada para aproveitar melhor a memória.

- a) Escreva em OCaml uma função

```
pack : 'a list -> ('a * int) list
```

para compactar listas, substituindo cada subsequência de valores repetidos por um par ordenado que indica qual o valor que se repete e qual o comprimento da subsequência. Dois exemplos:

```
pack [] = []
pack [10.1; 10.1; 10.1; 10.1; 10.1; 10.1; 10.1; 10.0; 10.0; 10.1; 10.0] =
                [(10.1, 7); (10.0, 2); (10.1, 1); (10.0,1)]
```

- b) Escreva agora uma função para descompactar listas.

```
unpack : ('a * int) list -> 'a list
```

Dois exemplos:

```
unpack [] = []
unpack [(10.1, 7); (10.0, 2); (10.1, 1); (10.0,1)] =
    [10.1; 10.1; 10.1; 10.1; 10.1; 10.1; 10.1; 10.0; 10.0; 10.1; 10.0]
```

- 18 - a) Escreva em OCaml uma função de ordem superior sobre listas, que permita aplicar sucessivamente uma operação binária associativa à esquerda a todos os elementos duma lista. A função também precisa de receber o elemento neutro da operação.

```
fold_left : ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a
```

Esboço do funcionamento da função:

```
fold_left f neutro [a1; a2; ...; an] = f (... (f (f neutro a1) a2) ...) an
```

b) Escreva uma função de ordem superior sobre listas, semelhante à anterior mas aplicável a operações binárias associativas à direita.

```
fold_right : ('a -> 'b -> 'b) -> 'a list -> 'b -> 'b
```

Esboço do funcionamento da função:

```
fold_right f [a1; a2; ...; an] neutro = f a1 (f a2 (... (f an neutro) ...))
```

c) Programe a função de ordem superior `map` recorrendo a uma destas funções.

- 19 - Procure no manual de referência da linguagem OCaml (disponível na página da cadeira, na coluna da esquerda - veja "Basic Operations") a lista completa das operações associadas aos tipos `bool`, `int`, `float`, `string` e `char`.

- 20 - Para cada um dos seguintes tipos, invente uma função com esse tipo:

```
int -> (int->int)
(int->int) -> (int->int)
int -> float -> string -> char
```
