Linguagens e Ambientes de Programação (2011/2012)

Teórica 02 (28/Fev/2012)

Computação, Algoritmos, Programas e Linguagens de Programação. Paradigmas de Programação

Paradigma imperativo versus paradigma funcional. Elementos essenciais da linguagem OCaml.

Avaliação de expressões em OCaml.

Tipos em OCaml. Tipos básicos e tipos estruturados. Inferência de tipos. Funções monomórficas e funções polimórficas. Não há sobreposição.

Computação

Computação

- Computação significa: processamento automático de informação.
- É a atividade realizada pelos computadores.
- O objetivo da informática é estudar a computação e formas úteis de tirar partido dela para resolver problemas importantes.

Facetas da computação

- **Faceta interna:** a informação é codificada usando símbolos (e.g. bytes), sendo a computação uma atividade de manipulação e transformação automática de símbolos.
- **Faceta externa:** ... mas, geralmente, as computações também estabelecem um diálogo interativo com o ambiente exterior (constituído por humanos e por outras máquinas).

Automatismos e sua especificação

- Qualquer computação é realizada de acordo com regras estabelecidas antes desse processamento se iniciar. É isso que significa o processamento ser automático.
- Daí a necessidade que especificar as computações de forma rigorosa e exaustiva, prevendo todas as eventualidades.

Algoritmos, Programas e Linguagens de Programação

Algoritmo

- Um algoritmo é um conjunto de regras abstratas que determinam, passo a passo, como uma computação vai decorrer.
- Em geral um problema pode ser resolvido usando diversos algoritmos.
- Um algoritmo é independente de qualquer linguagem de programação particular. Podemos imaginá-lo como se fosse para ser executado num computador ideal com memória infinita.

Muhammad ibn Musa al-Khwarizmi

A palavra "algoritmo" deriva da palavra **Algoritmi** que por sua vez corresponde à Latinização do nome do matemático Persa Muhammad ibn Musa al-Khwarizmi, nascido por volta de 780DC. Este cientista, que trabalhou quase toda a sua vida em Bagdad, deu importantes contribuições para a Álgebra, Trigonometria e Aritmética.

Programa

- Um programa é a expressão concreta dum algoritmo.

- Um programa implementa um algoritmo numa linguagem de programação concreta.

Linguagem de programação

- É uma notação para escrever programas.

Exemplo de algoritmo

Algoritmo de Euclides - mdc(m,n)

- Usar dois contadores x e y e inicializar x com m e y com n.
- Se x > y então x recebe x - y
- Se x < y então y recebe y - x
- Repetir os dois passos anteriores até que os valores de x e y fiquem iguais. Quando isso acontecer, esse é o resultado final.

Implementação em C do algoritmo anterior

```
#include <stdio.h>

int main() /* Implementação do algoritmo de Euclides */
{
    int m, n, x, y ;
    printf(">> ") ;
    scanf("%d %d", &m, &n) ;
    x = m ;
    y = n ;
    do {
        if( x > y ) x = x - y ;
        else if( x < y ) y = y - x ;
    } while( x != y ) ;
    printf("mdc(%d,%d) = %d\n", m, n, x) ;
    return 0 ;
}
```

Antiguidade dos Algoritmos

- Os algoritmos mais antigos que se conhecem devem-se aos Babilónios (3000AC-1500AC). Os seus livros de Matemática eram acima de tudo receitas sobre como efetuar os cálculos para resolver determinados problemas. Esses algoritmos eram para executar "à mão".
- A principal motivação para se usarem máquinas para executar algoritmos é a grande velocidade de execução que elas permitem.

Duas questões importantes sobre computação e linguagens

- Existem limites para a computação?

    - Sim, realmente existem problemas com solução para os quais não se consegue inventar qualquer algoritmo que descubra essa solução.
- Existe alguma linguagem de programação que permita implementar todos os algoritmos que se possam imaginar?
    - Sim, qualquer linguagem "normal" permite isso. Do ponto de vista do poder computacional, todas as linguagens normais são equivalentes.

Paradigmas de Programação

Já sabemos que computação significa processamento automático de informação, mas esta noção é demasiado vaga. Podemos interrogar-nos sobre quais os mecanismos concretos através dos quais esse processamento se efetua.

Na verdade, uma linguagem não pode deixar de se comprometer com algum conjunto mecanismos primitivos para processar informação. Ao efetuar essa escolha, a linguagem adere a um paradigma de programação particular.

É surpreendente a grande variedade de paradigma de programação que têm sido inventados.

Exemplos de paradigmas de programação e respetivos conceitos de base

- **Paradigma imperativo**
    - Conceitos: estado, atribuição, sequenciação
    - Linguagens: Basic, Pascal, C, Assembler.
- **Paradigma funcional**
    - Conceitos: função, aplicação, avaliação
    - Linguagens: Lisp, ML, OCaml, Haskell.
- **Paradigma lógico**
    - Conceitos: relação, dedução
    - Linguagens: Prolog.
- **Paradigma orientado pelos objetos**
    - Conceitos: objetos, mensagem
    - Linguagens: C++, Java, Eiffel.
- **Paradigma concorrente**
    - Conceitos: processo, comunicação (síncrona ou assíncrona)
    - Linguagens: Occam, Ada, Java.

Aspetos práticos

- Cada paradigma de programação determina uma forma particular de abordar os problemas e de formular as respetivas soluções. De facto, diferentes paradigmas de programação representam muitas vezes visões irreconciliáveis do processo de resolução de problemas.
- O grau de sucesso dum programador depende em parte da coleção de paradigmas que domina e da sua arte em escolher o modelo conceptual (paradigma) mais indicado para analisar e resolver cada problema.

Uma linguagem de programação pode combinar mais do que um paradigma

- C++ --- Paradigma imperativo + paradigma orientado pelos objetos.
- Java --- Paradigma imperativo + paradigma orientado pelos objetos + paradigma concorrente.
- OCaml --- Paradigma funcional + Paradigma imperativo + paradigma orientado pelos objetos.
- Ada --- Paradigma imperativo + paradigma concorrente.

Paradigma imperativo *versus* paradigma funcional

Paradigma imperativo

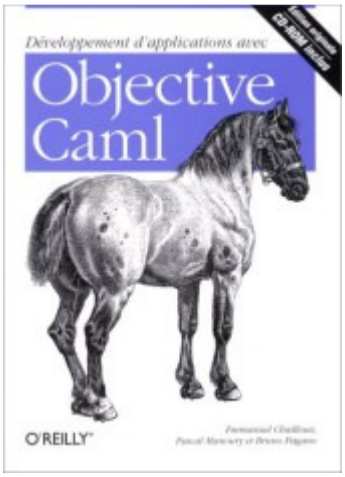
- A computação baseia-se na existência de um "estado" (conjunto de variáveis mutáveis) que vai evoluindo ao longo do tempo.
- O estado é modificado usando um "comando de atribuição" e existe um "operador de sequenciação".
- Linguagens: Basic, Pascal, C, etc., mas o Java também tem uma base imperativa, visto suportar variáveis mutáveis.

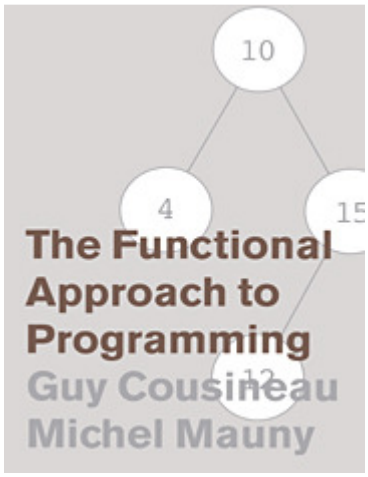
Paradigma funcional

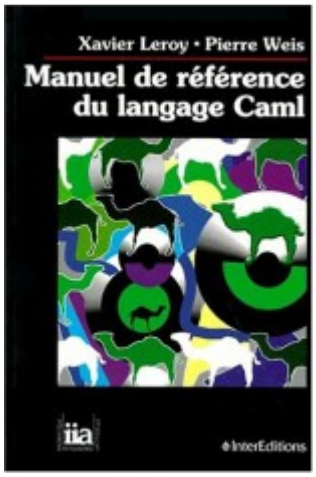
- A computação consiste na avaliação de "expressões" nas quais podem ocorrer chamadas de funções.
- O objetivo da avaliação duma expressão é a produção dum "resultado" ou "resposta".
- Não há "atribuição" nem variáveis cujo valor possa ser alterado.

- Cada função limita-se a produzir um resultado a partir dos seus argumentos.
- Linguagens: Lisp, ML, Haskell, Scheme.

Linguagem OCaml







Xavier Leroy

Didier Rémy

Damien Doligez

Jérôme Vouillon

Algumas características

- O OCaml foi desenvolvido no INRIA, a partir de 1991, por Xavier Leroy e Damien Doligez, a que se juntaram Jérôme Vouillon, Didier Rémy e outros em 1996.
- A linguagem OCaml é uma das muitas linguagens que derivam da linguagem ML. [O ML começou por ser a metalinguagem do

demonstrador de teoremas LCF, desenvolvido no final dos anos 70 por Robin Milner e outros na University of Edinburgh. Mas o ML evoluiu para passar a ser usado como linguagem de programação"normal". Eis alguns dialetos da linguagem ML: Standard ML, Caml, OCaml, Alice, F#.]

- O OCaml é uma linguagem onde se unificam os paradigmas funcional, imperativo e orientado pelos objetos. Na nossa cadeira estamos interessados em estudar apenas a parte funcional pura da linguagem.
- Tem um sistema de tipos estático com suporte para polimorfismo e inferência de tipos.
- As implementações de OCaml colocam um grande ênfase na velocidade de execução. A velocidade é superior à do C++, para programas orientados pelos objetos.
- Tipos básicos: caracteres, inteiros, reais, booleanos.
- Tipos derivados: funções, listas, tuplos, registos, tipos soma.
- Tem gestão automática de memória.
- Modularidade e compilação separada.
- Biblioteca.
- A linguagem não tem nenhum padrão reconhecido por um organismo oficial normativo. A implementação do INRIA funciona como padrão *de facto*.

Porque vamos trabalhar com a linguagem OCaml?

Na nossa cadeira, vamos trabalhar com a parte funcional da linguagem OCaml por diversas razões:

- Ter a experiência de usar um novo paradigma de programação: aprender as técnicas básicas da programação funcional e aprender a usar essas técnicas para resolver problemas relativamente complicados.
- Tomar contacto com uma linguagem cujo sistema de tipos é completamente estático e que suporta inferência de tipos.
- Ter a experiência de usar uma linguagem que disponibiliza um interpretador, além dum compilador.
- Tomar contacto com o sistema de módulos do OCaml, que é exemplar.

Elementos essenciais da linguagem OCaml

- O OCaml é uma linguagem multiparadigma, mas nesta cadeira vamos usar apenas o sua componente funcional, para aprender o estilo de programação funcional, sem distrações.
- Um programa é uma sequência de funções. As funções podem ser recursivas.
- São dois os principais mecanismos que podem ser usados na escrita do corpo das funções em OCaml:
  - APLICAÇÃO (aplicação duma função a argumentos). Ex: `sqrt 9`
  - IF-THEN-ELSE. Ex: `if prime x then x else x/2`

Exemplo de função que usa os dois mecanismos:

```
let rec fact x =
    if x = 0 then 1 else x * fact (x-1)
;;
```

Avaliação de expressões em OCaml

A ideia de **avaliação** está no centro do processo de execução de programas funcionais. Merece pois a nossa melhor atenção.

Em OCaml as expressões são avaliadas usando uma estratégia chamada *call-by-value*: uma funções só é aplicada aos seus argumentos depois de eles terem sido avaliados (ou, mais simplesmente, uma função só pode ser aplicada a valores). Esta é a estratégia usada na maioria das linguagens incluindo: Java, C, C++, Pascal, etc.

Exemplo de avaliação:

```
(fun x -> x+1)(2+3)
= (fun x -> x+1)5
= 5+1
= 6
```

Exemplo de avaliação que não termina. Considere a função `loop`, assim definida:

```
let rec loop x = loop x ;;
```

Avaliação:

```
(fun x -> 5)(loop 3)
= (fun x -> 5)(loop 3)
= (fun x -> 5)(loop 3)
= (fun x -> 5)(loop 3)
```

```
= ...  não termina
```

Mais uma avaliação. Considere a função `fact`, assim definida:

```
let rec fact x =
    if x = 0 then 1 else x * fact (x-1)
;;
```

Avaliação:

```
fact 3 =
= 3 * fact 2
= 3 * (2 * fact 1)
= 3 * (2 * (1 * fact 0))
= 3 * (2 * (1 * 1))
= 3 * (2 * 1)
= 3 * 2
= 6
```

Tipos em OCaml

Um **tipo** representa uma coleção de valores e tem associados um conjunto de **literais** e um conjunto de **operações**. (Um literal é uma expressão que não precisa de ser avaliada e denota um valor particular.)

Tipos básicos

TIPO	LITERAIS	OPERAÇÕES MAIS USADAS
int	5 -456	+ - * / mod min_int max_int int_of_float
float	3.14e-21	+. -. *. /. sqrt exp log sin ceil floor float_of_int
string	"" "ola"	^ String.length String.sub String.get
bool	false true	not || &&
char	'a' '$'	int_of_char char_of_int
unit	()	ignore

Tipos compostos

TIPO	LITERAIS	OPERAÇÕES MAIS USADAS
'a->'b	(fun x -> x+1)	aplicação
'a*'b	(5, 5.6)	fst snd *emparelhamento de padrões*
'a list	[] [3;5;7]	*emparelhamento de padrões*
tipos produto	*diversos*	*. emparelhamento de padrões*
tipos soma	*diversos*	*emparelhamento de padrões*

Inferência de tipos

O tipo dos argumentos e do resultado das funções não se declaram em OCaml. A implementação faz **inferência de tipos**: ela infere para cada função o tipo mais geral que é possível atribuir a essa função.

Qual o tipo das seguintes funções anónimas?

```
fun x -> x+1                    : int -> int
fun x -> x +. 1.0               : float -> float
fun x -> x ^ x                  : string -> string
fun (x,y) -> x + y              : (int * int) -> int
fun (x,y) -> (y,x)              : ('a*'b) -> ('b*'a)
fun x y -> (y,x)                : 'a -> 'b -> ('b*'a)
```

As quatro primeiras funções dizem-se **monomórficas** porque só aceitam argumentos de tipos fixos.
A duas últimas funções dizem-se **polimórficas** pois aceitam argumentos de tipos diversos.
De todas estas funções, a última é a única que aceita mais do que um argumento, neste caso dois. A razão pela qual o seu tipo tem duas setas será estudada mais tarde.

Não há sobreposição (overloading)

A linguagem OCaml não suporta sobreposição (overloading) de nomes ou operadores. Por exemplo, em OCaml o operador "+" denota apenas a soma inteira (a soma real denota-se "+.").

Para contrastar, as linguagens C, C++ e Java suportam sobreposição. Por exemplo, em Java o operador "+" é usado para denotar três operações: soma de inteiros, soma de reais, concatenação de strings.

Operadores

Para alguns dos operadores mais usados em OCaml, eis uma tabela de correspondência entre o OCaml e o Java:

OCaml	**Java**
&&	&&
||	||
not	!
mod	%
=	==
<>	!=
+	+
+.	+
^	+
-	-
-.	-

Comentários

```
(* Assim *)
```

#140