Programació Concurrent i en Temps Real

Exercicis programació funcional amb Erlang

Sebastià Vila-Marta

14 de setembre de 2026

1 Taller inicial

1. Instal·lació de l'entorn: Necessiteu instal·lar: l'entorn d'Erlang i el mode Erlang per a emacs

   ```
   $ sudo apt install erlang $ sudo apt install erlang-mode
   ```

2. El procediment de treball amb la shell Erlang:
   - Obriu la shell

     ```
     $ erl
     ```

   - Escriviu ordres a la shell

     ```
     4+5.
     ```

3. El procediment de treball amb la shell Erlang i mòduls:
   - Editeu un mòdul:

     ```
     emacs modul.erl
     ```

   - Compileu el mòdul:

     ```
     $ erlc modul.erl
     ```

   - Obriu la shell

     ```
     $ erl
     ```

   - Escriviu ordres a la shell, incloses crides a les funcions del mòdul:

     ```
     4+5.
     modul:suma(4,5).
     ```

4. Algunes funcions específiques de la shell d'Erlang:
   - Sortir de la shell `q()`
   - Oblidar el binding d'una variable (trampa útil per fer proves) `f(VAR)`
   - Oblidar el binding de totes les variables `f()`
   - Compilar un mòdul `c(MODUL)`
   - Buida la cua de missatges del procés shell `flush()`

2 Exercicis 1

1. Implementeu una funció que passa de graus Fahrenheit a Celsius.
2. Implementeu una funció que retorna els tres darrers elements d'una llista.
3. Implementeu un mòdul amb les següents funcions sobre llistes sense usar BIF's:
   (a) `length(L)` retorna la longitut d'una llista
   (b) `addl(L)` retorna la suma dels elements de la llista (que han de ser nombres).
   (c) `belongs(E,L)` retorna cert ssi `E` és un element de la llista `L`
   (d) `allequal(L)` retorna cert ssi tot els elements de `L` són iguals
4. Amplieu l'exemple del servidor d'echo per que admeti dos tipus de missatges: un que fa un echo (com ara) i un altre que fa un doble echo: envia dos missatges idèntics.
5. Amplieu el servidor anterior per que admeti un missage de «ping». Quan rep aquest missatge simplement contesta `true` a qui l'ha enviat.
6. Implementeu un mòdul amb:
   (a) Una funció que calcula 2^x.
   (b) Una funció que calcula $\sum_{i=a}^{b} i$ donats a i b sense usar la fórmula tancada que coneixeu.
   (c) Una funció que calcula $\sum_{i=a}^{b} i^2$ donats a i b sense usar la fórmula tancada que coneixeu.
   (d) Una funció que calcula $\prod_{i=a}^{b} i$ donats a i b sense usar la fórmula tancada que coneixeu.
7. Fent servir EUnit (`https://www.erlang.org/doc/apps/eunit/chapter.html`) Afegiu tests al mòdul que resulta de l'exercici 1

3 Exercicis 2

1. Escriviu una funció en el enters que calcula x^y. Accelereu-la considerant que per n parells, succeix que $x^n = (x^{n/2})^2$. Quin cost en temps té la funció accelerada?
2. Implementa un mòdul per treballar amb polinomis d'una variable. Ha de contenir, com a mínim, operacions per avaluar el polinomi en un punt, per calcular-ne el polinomi derivada i per a sumar i restar polinomis.
3. Implementa les següents funcions sobre llistes:
   (a) `last(L)` que retorna el darrer element de la llista `L`.
   (b) `pc(L1,L2)` que retorna el producte cartesià dels dos paràmetres com una llista de tuples de dos components. P. ex.:

   ```
   pc([1,2],[a,b]).
   [{1,a}, {1,b}, {2,a}, {2,b}]
   ```

   (c) `append(l1,L2)` que retorna la concatenació de les llistes `L1` i `L2`.
4. [La cua d'en Ratés] Implementa un mòdul `cua` amb les operacions `buida()`, `encua(E,C)`, `desencua(C)` i `buida?(C)`.

La tècnica d'implementació consisteix a representar la cua amb dues llistes `{A,AI}`, `A` conté els elements en ordre d'arribada i `AI` els conté en ordre invertit. Inicialment ambdues són buides.

Quan s'encuen elements es fa en el cap de la llista `A`. Quan es desencuen elements, es treu el cap de la llista `AI`. En cas que `AI` sigui buida, es transfereixen prèviament a l'operació desencuar tots els elements d'`A` cap `AI` invertint l'ordre. El resultat és que `A` esdevé buida i `AI` conté tots els elements que contenia `A` però en ordre invers. Per tant, el cap d'`AI` és ara el darrer element que contenia `A`. D'aquesta manera s'aconsegueixen temps d'accés amortitzats constants (raoneu per què!!).

5. [Difícil] Dissenyeu una funció `pm(L)` que genera la llista de les permutacions dels elements d'`L`. Investigueu prèviament una mica com es pot definir recursivament el conjunt de permutacions d'una llista.

4 Exercicis 3

1. Implementeu una funció que «comprimeix» una llista, i.e.: redueix cada subllista formada pels mateixos elements a una sola instància de l'element. P.ex.: `[1,2,3] = comprimeix([1,2,2,2,3,3])`.
2. Implementeu una funció que «empaqueta» una llista. Això és, «tanca en subllistes» tots els elements consecutius iguals de la llista original. Per ex.: `[1,[2,2],3] = pack([1,2,2,3])`.
3. Defineix una funció d'ordre superior que rep com a paràmetre una funció booleana (predicat) d'un sol paràmetre i retorna el predicat complementari.
4. Defineix una funció d'ordre superior que rep com a paràmetre dues funcions unàries i retorna la composició d'ambdues funcions.
5. El punt fix d'una funció $F(x)$ és el valor y tal que $F(y) = y$. Et pots acostar al punt fix per aproximacions successives a partir d'un valor arbitrari y_0 i fent $y_1 = F(y_0)$, $y_2 = F(y_1)$, etc. fins que y_n és un punt fix o proper a un punt fix. Escriu una funció Erlang que calculi el punt fix d'una funció. Sempre convergeix el mètode?
6. Defineix la funció `vectoritza(F)`. La funció ha de tornar una versió «vectoritzada» de la funció `F(X)`. Per exemple,

   ```
   vectoritza(fun (X) -> X*X end)([1,2,3,4]).
   [1,4,9,16]
   ```

7. Defineix la funció `traça(F)`, que donada una funció unària `F` retorna una funció equivalent però que escriu una traça de la crida i el valor de retorn. Com afegiries una escriptura «elegant» de la traça? Considera la macro `?FUNCTION_NAME` que s'avalua al nom de la funció on s'executa.
8. Una llista pot representar molt bé un arbre binari: senzillament cal considerar la llista com l'arrel, el primer element és el contingut del node i els dos elements següents els dos fills. P. ex.: `[10, [], [16, [], []]]`. Amb aquest format, escriviu funcions que:
   (a) Comptin el nombre de nodes de l'arbre.
   (b) Comptin el nombre de fulles de l'arbre.
   (c) Retorni la profunditat (màxima) de l'arbre.
9. Considerant els arbres de l'exercici anterior, dissenyeu una funció d'ordre superior que faci un recorregut postordre de l'arbre i apliqui a cada node una funció determinada. P. ex.:

   ```
   A = [10, [], [16, [], [30, [], []]]].
   print(N) -> io:write(N).
   ```

```
postordre(fun print/1, A).
30
16
10
```

5 Exercicis 4

1. Escriviu comprehensions que:

   (a) Donada una llista d'enters calculi la llista dels parells

   (b) Donada una llista d'enters calculi la llista dels quadrats

   (c) Donada una llista d'enters calcula la llista dels quadrats parells

2. Considerant el format d'arbre dels exercicis anteriors, dissenyeu una funció `fold_tree(F,I,T)` que faci una operació de fold sobre els nodes de l'arbre aplicant la funció `F(Acumulat, Node)`, amb valor inicial `I`. `fold_tree()` ha de recórrer els nodes en postordre.

   Amb la funció anterior, dissenyeu una funció que sumi els nodes d'un arbre.

3. Una expressió aritmètica es representa amb avantatge sobre un arbre binari en el que les fulles són enters i els nodes interns operacions. Per exemple, l'expressió $(3+4)*2$ és pot representar amb l'arbre `[mult [2, [], []], [suma, [3, [], []], [4, [], []]]]`

   Per avaluar una expressió en forma d'arbre, podeu aplicar una funció recursiva senzilla: proveu d'escriure una funció que avalúa una expressió en forma d'arbre.

   Si haguéssiu de definit una funció avaluadora basada en `fold_tree()`, quina estratègia seguiríeu? Noteu que us caldria una pila auxiliar. Per què?

4. [2-3 trees] Un 2-3-tree és un arbre de cerca balancejat. Això és, la seva profunditat és igual o similar per a totes les fulles. La propietat de ser balancejat garanteix que les cerques sempre tinguin cost temporal en cas pitjor $T(n) \in O(\log_2 n)$.

   Per simplificar l'exercici, assumim que les claus de l'arbre són valors enters. Un 2-3-tree té alguns nodes amb una clau i altres amb dues claus.

   - Els nodes amb una clau k_1 tenen dos fills: l, r. l conté les claus $k \leq k_1$ i r conté les claus $k > k_1$.
   - Els nodes amb dues claus ($k_1 < k_2$) tenen tres fills: l, m, r. l conté les claus $k \leq k_1$, m conté les claus $k_1 < k \leq k_2$ i r les claus $r > k_2$.

   Un exemple d'arbre 2-3 podria ser aquest:

```
[{2,5},
  [{1}, [], []],
  [{3,4}, [], [], []],
  [[6,7}, [], [], []]
]
```

   Es demana que implementeu un mòdul amb una funció que torni un arbre buit i una altra que pugui inserir un element a un arbre.

   La inserció en aquests arbres és delicada: cal que preservi l'ordre i, a la vegada, mantingui l'arbre balancejat. Per a entendre com fer la inserció estudieu-vos aquesta referència:

- `https://www.cs.princeton.edu/~dpw/courses/cos326-12/ass/2-3-trees.pdf`

5. Amplieu el mòdul anterior amb:
   - Una funció consultora que retorni el mínim valor desat en un arbre.
   - Un pedicat que, donat una arbre i un valor, digui si l'arbre conté el valor.
   - Una funció que permeti construir un arbre a partir d'una llista de valors.

6 Exercicis 5

1. Usant una aproximació de Taylor a la funció cos(), implementeu en un mòdul una funció `cos(X)` que calcula el cosinus d'un angle espressat en radians. Recordeu la l'aproximació de Taylor pel cosinus és $\cos(x) \approx 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots$ o, si voleu una expressió compacta,

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

2. Afegiu al mòdul anterior una funció tal que, donada una llista de reals, retorna la llista del cosinus de cada element. Penseu en la llistacom un vector i assumiu que definiu la funció `cos()` per a un vector.

   Per a provar-la genereu un vector aleatori de valors $v_i \in [0, \frac{\pi}{2}]$ i crideu a la funció. Potser us serà útil afegir uan funció que generi aquests vectors. Useu la funció `timer:tc()` per a saber quan triga l'execució. Feu una mica de test dels temps d'execució per a vectors grans i preneu-ne nota en una taula.

3. Anem a construïr un processador pel cas anterior que trenqui el vector (llista) en K subllistes i envii cada subllista a processar a un procés «calculador de cosinus» diferent.

   Primer dissenyeu un procés calculador de cosinus al que se li envia un vector d'uns pocs elements i retorna el vector cosinus corresponent.

   Després definiu una funció separadora, que donat una llista gran de valors, torna una llista de llistes agrupant els valors de la primera de M en M. Useu la funció `lists:split()`.

   Per últim, dissenyeu una funció que donada una llista, la separa en K subllistes i les envia a calcular una a cada procés. En obtenir els resultats de cada procés, forma una llista final que retorna. Tingueu present que els resultats dels processos calculadors poden arribar en un ordre diferent al que es van enviar!

   Es guanya temps amb aquesta estratègia? per què?