Butlletí de la Societat Catalana de Matemàtiques
Vol. 17, núm. 2, 2002. Pàg. 69–74

Suplement electrònic a «Matemàtiques a la creació musical»

Jordi Saludes

Com a complement a l'article de Salvador Comalada [5] publicat en el número anterior del Butlletí, descrivim els fitxers que trobareu a l'adreça `http://www.iecat.net/comalada/comalada.tar`. El conjunt de fitxers consta del parell de fragments musicals `bach.mid` i `schoen.mid` en format sonor *midi* i els seus precursors `Bach.lhs` i `Schoenberg.hs`, juntament amb `midi.hs` que proveix definicions auxiliars per a transformar i generar els fragments musicals.

1 Haskell

Els programes s'han fet utilitzant un paquet de programari anomenat *Haskore* escrit en llenguatge *Haskell* [4].

Com que tant el paquet com els exemples estan escrits en aquest llenguatge, és convenient que en parlem una mica, ja que *Haskell* és un llenguatge *purament funcional*. En els llenguatges imperatius el paradigma vol que les variables siguin com contenidors on depositem números: res impedeix —i de fet ho fem contínuament— canviar el valor d'aquests contenidors. En la majoria dels llenguatges clàssics una expressió com ara

$$x = x + 1 \tag{1}$$

té la semàntica: «Substituir el contingut del contenidor x pel valor del contenidor x més 1», o si voleu: $x_{n+1} = x_n + 1$. És a dir, usem el mateix símbol per a referir-nos a diferents termes de la successió x_n on n vé donat per l'etapa del càlcul.

Per contra, en un llenguatge purament funcional, el signe igual denota una regla de reescriptura: el que apareix al membre esquerre de la igualtat s'ha de substituir pel que hi ha al membre dret.

Cal fer notar que en el cas de l'expressió (1) es tracta d'una definició recursiva. Si les regles de reescriptura s'apliquessin immediatament, el programa

entraria en un llaç infinit. Això no passa perquè, a més, *Haskell* és un llenguatge *peresós*. Aquesta mena de llenguatges no avaluen una expressió fins que no és absolutament imprescindible. Això permet, entre altres coses, definir estructures infinites. Per exemple, el següent tros de codi defineix `nat` com la llista dels nombres naturals

```
seq n = n : seq (n+1)
nat = seq 1
```

Val a dir que l'operador «`:`» serveix per afegir un element al començament d'una llista.

Podeu trobar propaganda de programació funcional a [7].

2 Haskore

El paquet *Haskore* [3] és una col·lecció de definicions *Haskell* que permeten d'escriure i transformar fragments musicals. La part executable (és a dir, no funcional) de *Haskore* consisteix en les transformacions que escriuen fitxers *midi*. Aquest és un estàndard per a l'emmagatzematge digital de música que disposa de reproductors en la majoria de sistemes operatius.

Els dos exemples musicals es troben als fitxers `bach.mid` i `schoen.mid`. Aquests fitxers s'han creat executant el codi del fitxer `midi.hs`. La part executable correspon a la funció `main` del final del fitxer. La resta són les definicions dels fragments musicals que es troben als fitxers `bach.hs` i `schoen.hs`, juntament amb algunes funcions auxiliars.

2.1 Instal·lació

Si voleu recrear els fragments musicals o bé explorar el paquet *Haskore* us caldrà instal·lar uns fitxers que trobareu a les referències [3], [1]:

1. Un compilador o intèrpret *Haskell*. Jo ho he fet amb el sistema *hugs* [1] però una altra possibilitat és utilitzar *ghc* [2] en lloc de *hugs*.
2. El paquet *Haskore* format per fitxers de codi *Haskell*.

Primer cal que instal·leu *hugs* seguint les instruccions que acompanyen els fitxers. Després cal que desempaqueteu *Haskore* al directori «lib» dins del directori d'*hugs* i ja estareu a punt per usar-lo.

2.2 Descripció del paquet

El mòdul *Haskore* està orientat a manipular música programàticament. El material bàsic és la Nota

```
Note (n,o) d []
```

on n i o designen respectivament la cnota i l'octava. Així $(C,4)$ denota el do_4. El paràmetre d especifica la duració de la nota. Observem que com a nom de les cnotes s'utilitza la notació A = la, B = si, ..., G = sol, que és usual als països anglosaxons i Alemanya.

Una sèrie de comandes facilita la creació de notes amb les durades més corrents:

nom	durada
`fn`	rodona
`hn`	blanca
`qn`	negra
`en`	corxera
`sn`	semicorxera.

Per a les pauses usem

```
Rest d
```

Per a connectar fragments musicals usem els operadors `:+:` i `:=:`; el primer per a situar un fragment darrera l'altre, el segon per a fer-los sonar simultàniament.

3 El fragment de Bach

Descrivim ara el contingut del fitxer `Bach.lhs`, que permet construir el fragment de les *Invencions* de Bach que trobem a [5].

Definim el mòdul Bach i importem Haskore

```
> module Bach where
> import Haskore
```

`cor` i `scor` permeten de construir una corxera i semicorxera respectivament

```
> transposa = trans
> lmap f l = line (map f l++[Rest 0])
> nota d n = n d []
> scor p o = nota sn (p o)
> cor p o = nota en (p o)
```

Definim el fragment com la superposició de les seqüències melòdiques `veu1` i `veu2`

```
> bach = veu1 :=: veu2
> veu1 = snr :+: sA :+: s1a :+: sC :+: s1b :+: sEs
> sEs = sE :+: (scor g 5) :+: sE' :+: (scor e 5) :+: sF'
           :+: (scor c 5) :+: sF :+: (cor a 4)
> veu2 = hnr :+: sB :+: s2a :+: qnr :+:  sD :+: s2b :+: s2c
> base = map f4  [C, D, E, F, D, E, C]
>        where f4 p = (p,toInt 4)
```

`inverteix` aplica la transformació $\mathcal{I}_p(n) = 2p - n$ tal com apareix a [5, pàg. 66]. És a dir, la inversió que deixa fixa la nota d'alçada p. `fes` aplica la transformació f a la seqüència `base` i s'obté una nova seqüència de semicorxeres (`sn`)

```
> fes f = lmap (fsn . f) base
>   where fsn p = Note p sn  []
> inverteix :: Pitch -> Pitch -> Pitch
> inverteix p n = pitch (2*(absPitch p) - absPitch n)
```

Per fi definim les seqüències, `sA` inalterada, `sC` i `sB` transposades 7 semitons amunt i 12 avall respectivament. `sD` és a `sB` com `sC` és a `sA`.

```
> sA = fes id
> sC = fes (transposa 7)
> sB = snr :+: fes (transposa (-12))
> sD = snr:+: fes (transposa 7 . transposa (-12))
```

La transformació `tE` correspon a la composició $t_E = \mathcal{T}_{21} \circ \mathcal{I}_{\mathrm{do}_4}$ mentre que $t_F = \mathcal{T}_{-10} \circ t_E$. Definim `sE` com la seqüència base modificada per t_E.

```
> tE = transposa 21 . inverteix (C, 4)
> tF = transposa (-10) . tE
> sE = fes tE
```

Les seqüències E', F i F' s'obtenen de la base aplicant $\mathcal{T}_{-4} \circ t_E$, t_F i $\mathcal{T}_3 \circ t_F$. Cal, però, ajustar E' i F' a la tonalitat, cosa que fem amb `ajustaP` que es definirà més endavant

```
> sE'= ajustaP (fes (transposa (-4) . tE))
> sF = fes tF
> sF'= ajustaP (fes (transposa 3 . tF))
```

Ara anem a definir les sequències auxiliars (en gris a [5], fig. 3). Algunes d'aquestes porten un ornament anomenat *tremblament* definit com una breu alternança ràpida de dos sons veïns (vid. [6]: ornaments). Atès que es tracta d'un ornament he pensat que el codi queda més clar si s'aplica el `trill` *a posteriori*, un cop tot el fragment està definit. La definició de `trembla` que segueix fa que el tremblament s'apliqui només a les *cnotes* equivalents a l'argument `rp`. El periode del trinat s'ha ajustat a $1/48$ d'una rodona.

```
> trembla rp i n@(Note (p,_) _ _)
>   | rp == p  = trill i (1%48) n
> trembla rp i (m:+:ms) = trembla rp i m :+: trembla rp i ms
> trembla rp i m = m
```

Les seqüències que venen ara són bàsicament de corxeres (`en`). Observem que `s1a` i `s1b` tenen un tremblament en el *si* i el *fa* respectivament.

```
> s1a = trembla B 1 (lmap (nota en) [g 4, c 5, b 4, c 5]
        :+: (scor d 5))
> s1b = trembla F (2) (lmap (nota en) [d 5, g 5, f 5, g 5]
        :+: (scor e 5))
> s2a = lmap (nota en) [g 3, g 2]
> s2b = lmap (nota en) [c 4, b 3, c 4, d 4, e 4, g 3, a 3, b 3]
> s2c = lmap (nota en) [c 4, e 3, fs 3, g 3, a 3, b 3]
        :+: nota  (5%16) (c 4)
```

3.1 Tonalitat

La tonalitat d'una peça es dóna distingint un conjunt de notes *naturals*. Aquí representem la tonalitat amb una llista d'enters (els representants de les cnotes corresponents) i definim les tonalitats de *Do major* i *La menor* (que com a conjunt són el mateix)

```
> type Tonalitat = [Int]
> doMajor = map pitchClass [C, D, E, F, G, A, B]
> laMenor = map pitchClass [A, B, C, D, E, F, G]
```

Una nota n pertany a la tonalitat T si la cnota corresponent hi és. És a dir, si existeix $p \in T$ tal que $n - p \bmod 12 = 0$

```
> esATo :: Tonalitat -> Music -> Bool
> esATo t (Note (n,_) _ _) = filter mod12 t /= []
>          where  mod12 p = mod (p - np) 12 == 0
>                 np = pitchClass n
```

Per posar a to una nota, mirem primer si ja és a to. En aquest cas no fem res; en cas contrari hi apliquem la transformació `i`. La resta de la definició de `aTo` permet aplicar la comanda a qualsevol fragment de música amb la seguretat que la comanda s'aplicarà a cada nota individual. Observeu que per posar a to un fragment transposat (`(Trans j m)`) cal transposar també la tonalitat sumant j a cada cnota de la tonalitat.

```
> aTo :: (AbsPitch -> AbsPitch) -> Tonalitat -> Music -> Music
> aTo i t n@(Note p d v)
>       |esATo t n = n
>       |otherwise   = Note pf d v
>           where pf = pitch (i (absPitch p))
> aTo i t (Tempo r m) = Tempo r (aTo i t m)
> aTo i t (Trans j m) = Trans j (aTo i t' m)
>       where t' = map (+j) t
> aTo i t (m :+: ms)  = aTo i t m :+: aTo i t ms
> aTo i t (m :=: ms)  = aTo i t m :=: aTo i t ms
> aTo _ _ (Rest d)    = Rest d
```

Finalment definim comandes per ajustar a la tonalitat de Do major sigui pujant o baixant un semitò.

```
> ajustaP = aTo (+1) doMajor
> ajustaB = aTo (+(-1)) doMajor
```

Referències

[1] Hugs98. `http://haskell.cs.yale.edu/hugs` i `http://www.haskell.org/hugs`

[2] The Glasgow Haskell Compiler. `http://www.haskell.org/ghc/`

[3] Haskore. `http://www.haskell.org/haskore`

[4] Haskell. `http://www.haskell.org`

[5] COMALADA, S. «Matemàtiques a la creació musical». *Butlletí SCM*, 17-1, 65–78.

[6] CANDÉ, R. DE. *Diccionari de la música*. Edicions 62, 1982.

[7] HUGHES, J. «Why functional programming matters». *The Computer Journal*, **32**, 2, 1989. També a: David A. Turner (ed.). *Research Topics in Functional Programming*, Addison-Wesley, 1990.

DEPARTAMENT DE MATEMÀTICA APLICADA II
UNIVERSITAT POLITÈCNICA DE CATALUNYA
EDIFICI TR5
COLOM, 11
08223 TERRASSA
jordi.saludes@upc.es