Fundamentos de la Inteligencia Artificial

PED 2 Curso 2020-2021

Autor: Sergio Flor

Prueba online de la PEC: https://balath.github.io/

Tabla de Contenido

1. Descripción del conocimiento del dominio

Bach-machine es un Sistema Basado en Reglas modelado en Prolog que genera estructuras de sucesiones armónicas, inspiradas en la armonía de los corales de J.S. Bach, a partir de una tonalidad dada. Se han asumido algunas simplificaciones para acotar el modelo que se irán comentado en la descripción del modelo del dominio. Esta descripción se realizará desde el nivel más bajo al más alto, de la misma forma que aparece código: notas musicales; intervalos; escalas; acordes; funciones tonales; semifrases y estructura del coral.

Explicar de forma detallada todo el sistema musical y conceptos relacionados con la armonía clásica es algo que sin duda excede de este trabajo, por lo que intentaré explicar de la mejor forma posible el modelo, asumiendo abstracciones de la teoría subyacente a algunos conceptos más avanzados.

1.1. Notas musicales

Las notas musicales son las conocidas naturales do, re, mi, fa, sol, la, si, y además las combinaciones de estas con las alteraciones, el sostenido(♯) y el bemol(♭) que se han representado en el sistema como do_s (do sostenido) y do_b (do bemol).

Las notas musicales representan la "altura" del sonido, que representa la frecuencia de ese sonido. Así, el la de la mitad del piano (aprox.) tiene asignada, por convenio, una frecuencia de entre 440 y 442 Hz

1.2. Intervalos

Las alturas de las diferentes notas están separadas por diferentes intervalos. Simplificando, podemos decir que las unidades de medida de la altura son el tono[1](t) y el semitono(st) que se relacionan de la siguiente forma: 2 st = 1 t. Entre cada par de notas naturales hay un tono de distancia(do-re, re-mi, fa-sol…), excepto entre mi-fa y si-do, donde la distancia es de semitono. Por otro lado, el efecto del sostenidos es subir la altura un semitono y del bemol el de bajarla un semitono.

Los intervalos se denominan con dos "argumentos":

- el primero hace referencia a la distancia entre los nombres de las notas, incluyendo la de partida, por ejemplo, son intervalos de segunda do-re, sol-la, si-do, y de cuarta do-fa, re-sol. También lo seguirían siendo aunque las notas estuvieran alteradas (do_s-re sigue siendo un intervalo de segunda).
- el segundo especifica el tipo de intervalo que puede ser mayor, menor, justo, aumentado o disminuido. Intuitivamente, contando desde la nota do todos los intervalos serán mayores o justos[2] y conforme reducimos la distancia con alteraciones se reducen (menores y disminuidos) o se amplían (aumentados).

En realidad para modelar este sistema solo será necesario usar un cierto conjunto de intervalos, cuya combinación y composición dan lugar a los demás intervalos que se usarán. Estos son las 2as y 3as Mayores y menores y las 4as y 5as justas.

[1] "tono" tiene dos acepciones, puede ser una medida interválica, representando la distancia entre el do y el re, o hacer referencia a la tónica o nota "principal" que da nombre a una tonalidad, el tono de la tonalidad de do Mayor es do.

[2] Por razones históricas relacionadas con la afinación y construcción de los instrumentos (que se encuadran en los campos de la Acústica musical y la Organología), se mantienen la denominación de Mayor-justa para distintos intervalos: las 2as, 3as, 6as y 7as son Mayores o menores, las 4as y 5as son justas, y todos ellos pueden ser además disminuidos o aumentados.

1.3. Escalas de las tonalidades[3] mayores y menores

Estas escalas serán secuencias de 7 notas **sucesivas** (numeradas usualmente con números romanos), que guardan siempre la misma estructura de intervalos entre ellas. De esta forma, conociendo los intervalos que definen el modo mayor(menor), solo hay que aplicarlos a la nota que define el tono y obtenemos la escala mayor(menor) de ese tono. En el código se ve claramente esta relación en las reglas de las escalas de ambos modos.

1.4. Acordes

Los acordes serán grupos de tres(triadas) o cuatro(cuatriadas) notas distintas que suenan de forma **simultánea** (simplificando mucho y acotando a la armonía utilizada en la época). Las notas de los acordes usados se pueden definir lógicamente como una sucesión de terceras mayores y/o menores desde la nota denominada fundamental(I), y así se refleja en el código. Además, el orden "vertical" en el que suenan las notas se puede modificar con las *inversiones* que se identifican mediante un cifrado que indican intervalos resultantes desde la nueva primera nota del acorde.

1.5. Funciones tonales

Las funciones tonales se refieren a ciertos roles que desempeñan los acordes surgidos a partir de cada nota de la escala de una tonalidad y formados por notas, en principio, de la tonalidad.

Cada tipo de acordes tiene una función: los acordes con función de tónica (`ton`) aportan estabilidad en la tonalidad, las subdominantes (`sub`) son tensiones "blandas" o suaves y los acordes de dominante (`dom`) producen sensación de tensión y necesidad de "resolver" la tensión en una tónica.

Además, partiendo de acordes de la tonalidad y mediante la modificación de alguna nota o el añadido de notas "extrañas" a la tonalidad al acorde, se pueden realizar modulaciones (cambios) o enfatizaciones a otras tonalidades. Las enfatizaciones consisten, básicamente, en la introducción de cadencias que resuelven en algún acorde de la tonalidad que en ese momento ejerce de tónica de la tonalidad que se ha enfatizado. Se incluyen en el sistema por ejemplo, el acorde del II grado de la tonalidad se transforma en un acorde perfecto mayor con séptima menor y resulta en ser la dominante de la dominante del tono original, resolviendo en ella y dando la sensación de haber cambiado momentáneamente de tonalidad.

1.6. Semifrases

Con las herramientas que representan las funciones tonales, se pueden elaborar multitud de pequeñas frases o semifrases que, enlazando estas funciones tonales de forma adecuada, estructuran sucesiones armónicas que producen diferentes sensaciones al oído:

- de estabilidad o conclusión, cuando buscamos una tensión desde el reposo de la tónica hacia la dominante, y finalmente la resolvemos en la primera.
- de desconcierto o inestabilidad parcial, cuando las resoluciones no vienen de las funciones de mayor tensión o terminan en las de mayor estabilidad, sino que intervienen subdominantes.
- de tensión, cuando se realizan enfatizaciones a otros tonos utilizando funciones tonales ajenas a la tonalidad principal.

[3] Relación de los conceptos de tonalidad, tono y modo: Si tenemos la tonalidad de Si bemol Mayor, el tono es Si_b y el modo es el Mayor.

- de vuelta a la calma, cuando retomamos la tonalidad mediante una subdominante para terminar la frase cadenciando de forma conclusiva.

Estas pequeñas sucesiones que tiene un sentido armónico claro se pueden denominar, en general, cadencias o procesos cadenciales.

1.7. Estructura del coral

El sistema de reglas genera un modelo estándar de coral simplificado en una estructura de cuatro semifrases, de forma que la primera semifrase será conclusiva y "afirmará" la tonalidad, la segunda será semi o no conclusiva, la tercera aumentará la tensión mediante enfatizaciones, y la cuarta retomará la tonalidad finalizando de forma conclusiva.

2. Metodología de desarrollo

La metodología utilizada ha sido partir del nivel lógico más bajo de abstracción del dominio, en este caso han sido las notas musicales, para ir construyendo en cada paso un conjunto de reglas que añadían una nueva capa de complejidad al sistema, en el orden en el que se ha descrito en el apartado anterior.

En un momento dado, ha surgido la necesidad de acotar el problema para poder perfilar un modelo funcional factible y acorde a la realidad del proyecto y que cumpliera con los requisitos de la práctica. Este momento ha sido el de la elección del sistema de enlace de los acordes. Por un lado, podría haberse hecho aportando diferentes posibilidades de enlace a cada acorde, tanto por delante como por detrás, y que la generación del coral se realizara acorde a acorde, lo que hubiera ampliado la complejidad considerablemente. Por otro, la opción elegida de definir semifrases como unidad básica de construcción del coral, no deja de ser fiel a la realidad y reduce la complejidad del sistema.

Con el estado final del modelo, queda abierta la puerta a la adición de más tipos de acordes, más estructuras de semifrases, la posibilidad de incluir modulaciones completas, la adición de diferentes estructuras de corales… que producirán sin duda resultados armónicamente más ricos e interesantes.

3. Descripción de la estructura de la base de reglas

3.1. Estructura

La estructura del sistema de reglas sigue exactamente el orden y desarrollo que se ha descrito en el apartado 1 de la descripción del conocimiento.

En las secciones de código `% 1.` y `% 2.`, se declaran las notas y los hechos correspondientes a las distancias de intervalos.

En la sección `% 3. Modos mayor/menor` se declaran las reglas para formar las escalas del modo mayor y del modo menor, para lo que se utilizan los hechos declarados en la sección de intervalos.

En la sección `% 4. Acordes` se declaran las reglas para los distintos tipos de acordes, definidas en función de los hechos declarados en la sección de intervalos. Para las inversiones, simplemente se han utilizado hechos con dos argumentos, uno de entrada con el acorde original y otro de salida para el invertido.

En la sección `% 5. Funciones tonales` se declaran las reglas para las distintas funciones utilizando la correspondencia del tipo de acorde y la nota de la escala correspondiente, y por tanto las reglas definidas para cada objetivo.

En la sección `% 6.Semifrases cadenciales,` se genera a partir de una nota fundamental `I` la lista de acordes y el cifrado de cada semifrase, que esta formada por una sucesión de unas determinadas funciones tonales en estado fundamental o en distintas inversiones. A su vez, cada grupo de semifrases se ha aglutinado en predicados del tipo `semifrase_conclusiva/3` que incluye como tercer parámetro un entero, y que son seleccionados aleatoriamente por un predicado más general `semifrase_conclusiva/2` en cuyo antecedente se genera el número aleatorio y se hace la selección.

En la sección `% 7. Secuencia completa para un modo de coral standard`, se define la estructurta de 4 semifrases y se realiza la concatención de las listas de acordes con las llamadas recursivas a `coral_std/3`.

En la sección `% 8. Tono del coral: predicados dinámicos y restricciones`, se declara el predicado dinámico al que se añadirá la regla que almacenará el tono elegido por el usuario del sistema, y además se añade una regla para comprobar que este tono es válido (para evitar tonalidades con sostenidos/bemoles dobles).

En la sección `% 9. Predicados recursivos para mostrar armonías por pantalla`, se definen las reglas para consumir el resultado de la generación de listas de acordes y cifrados, teniendo en cuenta que éste tiene dos niveles de anidación.

Por último en la sección `% 10. Interfaz de usuario`, se define la secuencia general del programa y el bucle de la interfaz para probar, más de una vez si se desea, con distintos tonos o con las diferentes combinaciones aleatorias que se generan desde un mismo tono.

3.2. Eficiencia

En un primer momento, había utilizado operadores de corte para las reglas que involucraban distintas semifrases, con intención de mejorar la eficiencia. Sin embargo, y tras realizar varias mediciones, se revela que estos cortes no mejoraban sistemáticamente las cifras de inferencia —en el baile de cifras de inferencias hay que considerar que al preguntar coral(_), estos generan de forma aleatoria—:

```
% c:/Learn/Uned/fia/pec/bach-machine.pl compiled 0.02 sec, 223 clauses
?- time(coral(do)).
% 13,657 inferences, 0.016 CPU in 0.009 seconds (174% CPU, 874048 Lips)
true.

?- time(coral(do)).
% 350 inferences, 0.000 CPU in 0.000 seconds (?% CPU, Infinite Lips)
true.

?- time(coral(do)).
% 355 inferences, 0.000 CPU in 0.000 seconds (?% CPU, Infinite Lips)
true.

?- time(coral(fa_s)).
% 327 inferences, 0.000 CPU in 0.000 seconds (?% CPU, Infinite Lips)
true.

?- time(coral(la_b)).
% 363 inferences, 0.000 CPU in 0.000 seconds (?% CPU, Infinite Lips)
true.

?-
```

Versión con operadores de corte

```
% c:/Learn/Uned/fia/pec/prueba.pl compiled 0.02 sec, 225 clauses
?- time(coral(la_b)).
% 13,742 inferences, 0.000 CPU in 0.008 seconds (0% CPU, Infinite Lips)
true.

?- time(coral(do)).
% 395 inferences, 0.000 CPU in 0.000 seconds (?% CPU, Infinite Lips)
true.

?- time(coral(la_b)).
% 369 inferences, 0.000 CPU in 0.000 seconds (?% CPU, Infinite Lips)
true.

?- time(coral(fa_s)).
% 445 inferences, 0.000 CPU in 0.000 seconds (?% CPU, Infinite Lips)
true.

?- time(coral(fa_s)).
% 435 inferences, 0.000 CPU in 0.000 seconds (?% CPU, Infinite Lips)
true.

?- time(coral(fa_s)).
% 401 inferences, 0.000 CPU in 0.000 seconds (?% CPU, Infinite Lips)
true.

?- time(coral(fa_s)).
% 349 inferences, 0.000 CPU in 0.000 seconds (?% CPU, Infinite Lips)
true.

?- time(coral(la_b)).
% 315 inferences, 0.000 CPU in 0.000 seconds (?% CPU, Infinite Lips)
true.

?-
```

Versión sin operadores de corte

Por otro lado, las versiones iniciales del programa hacían uso en determinado momento de predicados con más variables, por ejemplo, para calcular el acorde de 7ª de disminuida de un tono se utilizaba la regla de la escala mayor, que utiliza 7 variables y 7 reglas más para inferir el resultado.

Posteriormente, se cambió por calcular un intervalo de segunda desde el tono original, aprovechando que la segunda también participa en este acorde. Con esto se consiguió reducir los pasos de inferencia en aproximadamente un 60%.

- Regla primera versión:

```
sept_dis_VII(I,[VII,II,IV,VI]) :-
    sept_dis([VII,II,IV,VI]),esc_maj(I,[I,_,_,_,_,_,VII]).
```

- Regla más eficiente:

```
sept_dis_VII(I,[VII,II,IV,VI]) :- sept_dis([VII,II,IV,VI]),seg_maj(I,II).
```

- Comparación en la eficiencia

```
?- time(sept_dis_VII(la_b,_)).
% 98 inferences, 0.000 CPU in 0.000 seconds (?% CPU, Infinite Lips)
true .

?-
% c:/learn/uned/fia/pec/bach-machine compiled 0.00 sec, 0 clauses
?- time(sept_dis_VII(la_b,_)).
% 38 inferences, 0.000 CPU in 0.000 seconds (?% CPU, Infinite Lips)
```

Resultados de inferencia para una primera versión y una versión eficiente tras la recarga del archivo modificado

Además, ya en la redacción de la memoria, y al realizar trazas para los casos de pruebas, se modificó también el orden en que aparecen los antecedentes en las reglas de las funciones tonales: los intervalos están definidos como hechos y los tipos de acordes como reglas, por lo que es más directo inferir primero el hecho y a raíz de ahí aplicar la regla, que aplicar posibilidades de reglas hasta llegar a una que sea conforme al hecho.

En este caso se redujo casi a la mitad el número de inferencias para algunos casos:

```
?-
% c:/learn/uned/fia/pec/bach-machine compiled 0.02 sec, 0 clauses
?- time(dom(mi,Dominante)).
% 38 inferences, 0.000 CPU in 0.000 seconds (?% CPU, Infinite Lips)
Dominante = [si, re_s, fa_s].

?-
% c:/learn/uned/fia/pec/bach-machine compiled 0.02 sec, 0 clauses
?- time(dom(mi,Dominante)).
% 21 inferences, 0.000 CPU in 0.000 seconds (?% CPU, Infinite Lips)
Dominante = [si, re_s, fa_s].
```

Resultados de inferencia para los antecedentes en orden {regla, hecho},
y en orden {hecho, regla} tras la recarga del archivo modificado

Finalmente, tras esta refactorización del código de prolog para las reglas de las funciones tonales se ha alcanzado una mejora de aproximadamente el 80 % en la eficiencia de estas reglas. Esto ha tenido efecto también en la funcionalidad general, ya que en cada generación de un coral se consultan de 16 a 19 reglas de funciones tonales, y se han mejorado los números de inferencia en alrededor de un 60%:

```
?- time(coral(do)).
% 148 inferences, 0.000 CPU in 0.000 seconds (?% CPU, Infinite Lips)
true .

?- time(coral(fa_s)).
% 184 inferences, 0.000 CPU in 0.000 seconds (?% CPU, Infinite Lips)
true .

?- time(coral(la_b)).
% 124 inferences, 0.000 CPU in 0.000 seconds (?% CPU, Infinite Lips)
true .
```

Resultados de inferencia con todas las mejoras aplicadas

4. Código Prolog

```
%  1. Notas musicales:

do_b.
do.
do_s.
re_b.
re.
re_s.
mi_b.
mi_b_b.
mi.
mi_s.
fa_b.
fa.
fa_s.
fa_s_s.
sol_b.
sol.
sol_s.
la_b_b.
la_b.
la.
la_s.
si_b_b.
si_b.
si.
si_s.


% 2. Intervalos ascendentes: (min-menor, maj-mayor, aug-aumentada, dis-disminuida)

seg_min(do,re_b).
seg_min(do_s,re).
seg_min(re,mi_b).
seg_min(re_s,mi).
seg_min(mi_b,fa_b).
seg_min(mi,fa).
seg_min(mi_s,fa_s).
seg_min(fa,sol_b).
seg_min(fa_s,sol).
```

```
seg_min(sol,la_b).
seg_min(sol_s,la).
seg_min(la_b,si_b_b).
seg_min(la,si_b).
seg_min(la_s,si).
seg_min(si_b,do_b).
seg_min(si,do).
seg_min(si_s,do_s).

seg_maj(do_b,re_b).
seg_maj(do,re).
seg_maj(do_s,re_s).
seg_maj(re_b,mi_b).
seg_maj(re,mi).
seg_maj(re_s,mi_s).
seg_maj(mi_b,fa).
seg_maj(mi,fa_s).
seg_maj(fa_b,sol_b).
seg_maj(fa,sol).
seg_maj(fa_s,sol_s).
seg_maj(sol_b,la_b).
seg_maj(sol,la).
seg_maj(sol_s,la_s).
seg_maj(la_b,si_b).
seg_maj(la,si).
seg_maj(la_s,si_s).
seg_maj(si_b_b,do_b).
seg_maj(si_b,do).
seg_maj(si,do_s).

ter_min(do_b,mi_b_b).
ter_min(do,mi_b).
ter_min(do_s,mi).
ter_min(re_b,fa_b).
ter_min(re,fa).
ter_min(re_s,fa_s).
ter_min(mi_b,sol_b).
ter_min(mi,sol).
ter_min(mi_s,sol_s).
ter_min(fa_b,la_b_b).
ter_min(fa,la_b).
ter_min(fa_s,la).
```

```
ter_min(fa_s_s,la_s).
ter_min(sol_b,si_b_b).
ter_min(sol,si_b).
ter_min(sol_s,si).
ter_min(la_b,do_b).
ter_min(la,do).
ter_min(la_s,do_s).
ter_min(si_b,re_b).
ter_min(si,re).
ter_min(si_s,re_s).

ter_maj(do_b,mi_b).
ter_maj(do,mi).
ter_maj(do_s,mi_s).
ter_maj(re_b,fa).
ter_maj(re,fa_s).
ter_maj(re_s,fa_s_s).
ter_maj(mi_b,sol).
ter_maj(mi,sol_s).
ter_maj(fa_b,la_b).
ter_maj(fa,la).
ter_maj(fa_s,la_s).
ter_maj(sol_b,si_b).
ter_maj(sol,si).
ter_maj(sol_s,si_s).
ter_maj(la_b,do).
ter_maj(la,do_s).
ter_maj(si_b,re).
ter_maj(si,re_s).

cua_jus(do_b,fa_b).
cua_jus(do,fa).
cua_jus(do_s,fa_s).
cua_jus(re_b,sol_b).
cua_jus(re,sol).
cua_jus(re_s,sol_s).
cua_jus(mi_b,la_b).
cua_jus(mi,la).
cua_jus(mi_s,la_s).
cua_jus(fa,si_b).
cua_jus(fa_s,si).
cua_jus(sol_b,do_b).
```

```
cua_jus(sol,do).
cua_jus(sol_s,do_s).
cua_jus(la_b,re_b).
cua_jus(la,re).
cua_jus(la_s,re_s).
cua_jus(si_b,mi_b).
cua_jus(si,mi).
cua_jus(si_s,mi_s).

qui_jus(do_b,sol_b).
qui_jus(do,sol).
qui_jus(do_s,sol_s).
qui_jus(re_b,la_b).
qui_jus(re,la).
qui_jus(re_s,la_s).
qui_jus(mi_b,si_b).
qui_jus(mi,si).
qui_jus(mi_s,si_s).
qui_jus(fa_b,do_b).
qui_jus(fa,do).
qui_jus(fa_s,do_s).
qui_jus(sol_b,re_b).
qui_jus(sol,re).
qui_jus(sol_s,re_s).
qui_jus(la_b,mi_b).
qui_jus(la,mi).
qui_jus(la_s,mi_s).
qui_jus(si_b,fa).
qui_jus(si,fa_s).

% 3. Modos mayor/menor

esc_maj(I,[I,II,III,IV,V,VI,VII]):- seg_maj(I,II),seg_maj(II,III),seg_min(III,IV),seg_maj(IV,V),seg_maj(V,VI),seg_maj(VI,VII).
esc_min(I,[I,II,III,IV,V,VI,VII]):- seg_maj(I,II),seg_min(II,III),seg_maj(III,IV),seg_maj(IV,V),seg_min(V,VI),seg_maj(VI,VII).


% 4. Acordes

  % 4.1 Triadas

  perf_maj([I,III,V]) :- ter_maj(I,III),ter_min(III,V).
  perf_min([I,III,V]) :- ter_min(I,III),ter_maj(III,V).
```

```
quin_dis([I,III,V]) :- ter_min(I,III),ter_min(III,V).
quin_aug([I,III,V]) :- ter_maj(I,III),ter_maj(III,V).

% 4.2 Cuatriadas

sept_min([I,III,V,VII]) :- ter_maj(I,III),ter_min(III,V),ter_min(V,VII).
sept_maj([I,III,V,VII]) :- ter_maj(I,III),ter_min(III,V),ter_maj(V,VII).
sept_dis([I,III,V,VII]) :- ter_min(I,III),ter_min(III,V),ter_min(V,VII).
sept_sem([I,III,V,VII]) :- ter_min(I,III),ter_min(III,V),ter_maj(V,VII).

% 4.3 Inversiones

inv_6([I,III,V],[III,V,I]).
inv_64([I,III,V],[V,I,III]).

inv_65([I,III,V,VII],[III,V,VII,I]).
inv_43([I,III,V,VII],[V,VII,I,III]).
inv_2([I,III,V,VII],[VII,I,III,V]).

% 5. Funciones tonales

% 5.1 Básicas

ton_maj(I,[I,III,V])              :- perf_maj([I,III,V]).
ton_min(I,[I,III,V])              :- perf_min([I,III,V]).
rel_maj(I,[III,V,VII])            :- ter_min(I,III),perf_maj([III,V,VII]).
ton_III(I,[III,V,VII])            :- ter_maj(I,III),perf_min([III,V,VII]).
dom(I,[V,VII,II])                 :- seg_maj(I,II),perf_maj([V,VII,II]).
sept_dom(I,[V,VII,II,IV])         :- qui_jus(I,V),sept_min([V,VII,II,IV]).
sub_IV_maj(I,[IV,VI,I])           :- perf_maj([IV,VI,I]).
sub_II_maj(I,[II,IV,VI])          :- seg_maj(I,II),perf_min([II,IV,VI]).
sub_VI_maj(I,[VI,I,III])          :- perf_min([VI,I,III]).
dom_VII_maj(I,[VII,II,IV])        :- seg_maj(I,II),quin_dis([VII,II,IV]).
sept_dis_VII(I,[VII,II,IV,VI])    :- seg_maj(I,II),sept_dis([VII,II,IV,VI]).
sept_sen_VII(I,[VII,II,IV,VI])    :- seg_maj(I,II),sept_sem([VII,II,IV,VI]).

% 5.2 Dominantes y subdominantes secundarias

dom_dom(I,X) :- qui_jus(I,V),dom(V,X).
dom_dom_sept(I,X):- qui_jus(I,V),sept_dom(V,X).
sub_IV_dom(I,X):- qui_jus(I,V),sub_IV_maj(V,X).
```

```
dom_sub_sept(I,X):- cua_jus(I,IV),sept_dom(IV,X).

% 6. Semifrases cadenciales

  % 6.1 Procesos cadenciales que reafirman la tonalidad

  cad_perf(I,([T,S64,D65,T],["T","S64","D65","T"])) :-
      ton_maj(I,T),sub_IV_maj(I,S),inv_64(S,S64),sept_dom(I,D7),inv_65(D7,D65).
  cad_647(I,([T,S64,D64,D7,T],["T","S64","D64","D7","T"])) :-
      ton_maj(I,T),sub_IV_maj(I,S),inv_64(S,S64),inv_64(T,D64),sept_dom(I,D7).
  cad_SIIDT(I,([T,SII,D65,T],["T","SII","D65","T"])) :-
      ton_maj(I,T),sub_II_maj(I,SII),sept_dom(I,D7),inv_65(D7,D65).
  cad_SIISDT(I,([T,SII,S64,D65,T],["T","SII","S64","D65","T"])) :-
      ton_maj(I,T),sub_II_maj(I,SII),sub_IV_maj(I,S),inv_64(S,S64),sept_dom(I,D7),inv_65(D7,D65).
  cad_SVISDT(I,([T,SVI6,S64,D65,T],["T","SVI6","S64","D65","T"])):-
      ton_maj(I,T),sub_VI_maj(I,SVI),inv_6(SVI,SVI6),sub_IV_maj(I,S),inv_64(S,S64),sept_dom(I,D7),inv_65(D7,D65).

      % 6.1.1 Elección aleatoria de frases conclusivas

      semifrase_conclusiva(I,(Acordes,Cifrado)):- random_between(1,5,X),semifrase_conclusiva(I,(Acordes,Cifrado),X).

      semifrase_conclusiva(I,(Acordes,Cifrado),1):- cad_perf(I,(Acordes,Cifrado)).
      semifrase_conclusiva(I,(Acordes,Cifrado),2):- cad_647(I,(Acordes,Cifrado)).
      semifrase_conclusiva(I,(Acordes,Cifrado),3):- cad_SIIDT(I,(Acordes,Cifrado)).
      semifrase_conclusiva(I,(Acordes,Cifrado),4):- cad_SIISDT(I,(Acordes,Cifrado)).
      semifrase_conclusiva(I,(Acordes,Cifrado),5):- cad_SVISDT(I,(Acordes,Cifrado)).

  % 6.2 Semi-conclusivas y no conclusivas que comienzan en subdominante

  cad_plag_III(I,([S64,III,S,T],["S64","TIII","S","T"])):-
      ton_maj(I,T),ton_III(I,III),sub_IV_maj(I,S),inv_64(S,S64).
  cad_plag_VI(I,([SII,SVI6,S64,T],["SII","SVI6","S64","T"])):-
      sub_II_maj(I,SII),ton_maj(I,T),sub_VI_maj(I,SVI),inv_6(SVI,SVI6),sub_IV_maj(I,S),inv_64(S,S64).
  cad_semicad(I,([D6,III64,S6,D6],["D6","TIII64","S6","D6"])):-
      ton_III(I,III),inv_64(III,III64),sub_IV_maj(I,S),inv_6(S,S6),dom(I,D),inv_6(D,D6).
  cad_rota(I,([SII,S64,D65,SVI6],["SII","S64","D65","TVI6"])):-
      sub_II_maj(I,SII),sub_IV_maj(I,S),inv_64(S,S64),sept_dom(I,D7),inv_65(D7,D65),sub_VI_maj(I,SVI),inv_6(SVI,SVI6).

      % 6.2.1 Elección aleatoria de frases no conclusivas

      semifrase_no_conclusiva(I,(Acordes,Cifrado)) :- random_between(1,4,X),semifrase_no_conclusiva(I,(Acordes,Cifrado),X).
```

```
    semifrase_no_conclusiva(I,(Acordes,Cifrado),1):- cad_plag_III(I,(Acordes,Cifrado)).
    semifrase_no_conclusiva(I,(Acordes,Cifrado),2):- cad_plag_VI(I,(Acordes,Cifrado)).
    semifrase_no_conclusiva(I,(Acordes,Cifrado),3):- cad_semicad(I,(Acordes,Cifrado)).
    semifrase_no_conclusiva(I,(Acordes,Cifrado),4):- cad_rota(I,(Acordes,Cifrado)).

% 6.3 Con enfatizaciones a otros tonos
cad_DD(I,([T,SII64,DD43,D6],["T","SII64","D/D43","D6"])):-
    ton_maj(I,T),sub_II_maj(I,SII),inv_64(SII,SII64),dom_dom_sept(I,DD),inv_43(DD,DD43),dom(I,D),inv_6(D,D6).
cad_DS(I,([T,III64,DS2,S6],["T","III64","D/S2","S6"])):-
    ton_maj(I,T),ton_III(I,III),inv_64(III,III64),dom_sub_sept(I,DS),inv_2(DS,DS2),sub_IV_maj(I,S),inv_6(S,S6).
cad_DS_DD(I,([T,DS,S64,DD,D64],["T","D/S","S64","D/D","D64"])) :-
    ton_maj(I,T),dom_sub_sept(I,DS),sub_IV_maj(I,S),inv_64(S,S64),dom_dom_sept(I,DD),dom(I,D),inv_64(D,D64).
cad_rel_min(I,([T,D6,VIIdis,VI6],["T","D6","VIIdis","VI6"])):-
    ton_maj(I,T),dom(I,D),inv_6(D,D6),sept_dis_VII(I,VIIdis),sub_VI_maj(I,VI),inv_6(VI,VI6).

    % 6.3.1 Elección aleatoria de frases con enfatizaciones

    semifrase_modulante(I,(Acordes,Cifrado)):- random_between(1,4,X),semifrase_modulante(I,(Acordes,Cifrado),X).

    semifrase_modulante(I,(Acordes,Cifrado),1):- cad_DD(I,(Acordes,Cifrado)).
    semifrase_modulante(I,(Acordes,Cifrado),2):- cad_DS(I,(Acordes,Cifrado)).
    semifrase_modulante(I,(Acordes,Cifrado),3):- cad_DS_DD(I,(Acordes,Cifrado)).
    semifrase_modulante(I,(Acordes,Cifrado),4):- cad_rel_min(I,(Acordes,Cifrado)).

% 6.4 Retoman la tonalidad comenzando en la subdominante

cad_sub_perf(I,([S64,T,D65,T],["S64","T","D65","T"])) :-
    ton_maj(I,T),sub_IV_maj(I,S),inv_64(S,S64),sept_dom(I,D7),inv_65(D7,D65).
cad_sub_647(I,([S64,D64,T,D43,T],["S64","T","D64","D43","T"])) :-
    ton_maj(I,T),sub_IV_maj(I,S),inv_64(S,S64),inv_64(T,D64),sept_dom(I,D7),inv_43(D7,D43).
cad_sub_SIIDT(I,([SII,T,D65,T],["SII","T","D65","T"])) :-
    ton_maj(I,T),sub_II_maj(I,SII),sept_dom(I,D7),inv_65(D7,D65).
cad_sub_SIISDT(I,([SII,T,S64,D65,T],["SII","T","S64","D65","T"])) :-
    ton_maj(I,T),sub_II_maj(I,SII),sub_IV_maj(I,S),inv_64(S,S64),sept_dom(I,D7),inv_65(D7,D65).
cad_sub_SVISDT(I,([SVI6,T,S64,D65,T],["SVI6","T","S64","D65","T"])):-
    ton_maj(I,T),sub_VI_maj(I,SVI),inv_6(SVI,SVI6),sub_IV_maj(I,S),inv_64(S,S64),sept_dom(I,D7),inv_65(D7,D65).

    % 6.4.1 Elección aleatoria de frases conclusivas

    semifrase_conclusiva_sub(I,(Acordes,Cifrado)) :- random_between(1,5,X),semifrase_conclusiva_sub(I,(Acordes,Cifrado),X).

    semifrase_conclusiva_sub(I,(Acordes,Cifrado),1):- cad_sub_perf(I,(Acordes,Cifrado)).
```

```
      semifrase_conclusiva_sub(I,(Acordes,Cifrado),2):- cad_sub_647(I,(Acordes,Cifrado)).
      semifrase_conclusiva_sub(I,(Acordes,Cifrado),3):- cad_sub_SIIDT(I,(Acordes,Cifrado)).
      semifrase_conclusiva_sub(I,(Acordes,Cifrado),4):- cad_sub_SIISDT(I,(Acordes,Cifrado)).
      semifrase_conclusiva_sub(I,(Acordes,Cifrado),5):- cad_sub_SVISDT(I,(Acordes,Cifrado)).


% 7. Secuencia completa para un modo de coral standard

coral :- tono(I), coral(I).
coral(I):- coral_std(I,(Acordes,Cifrado),[1,2,3,4]),mostrar(Acordes,Cifrado) ,!.

coral_std(I,([A|AS],[C|CS]),[1|XS]) :- semifrase_conclusiva(I,(A,C)),coral_std(I,(AS,CS),XS).
coral_std(I,([A|AS],[C|CS]),[2|XS]) :- semifrase_no_conclusiva(I,(A,C)),coral_std(I,(AS,CS),XS).
coral_std(I,([A|AS],[C|CS]),[3|XS]) :- semifrase_modulante(I,(A,C)),coral_std(I,(AS,CS),XS).
coral_std(I,([A],[C]),[4]) :- semifrase_conclusiva_sub(I,(A,C)).

% 8. Tono del coral: predicados dinámicos y restricciones.

:- dynamic tono/1.

tono_valido(N) :-
      (N == do_b;N == do;N == do_s;N == re_b;N == re;N == mi_b;N == mi;N == fa;N == fa_s;N == sol;N == la_b;N == la;
       N == si_b;N == si).

% 9. Predicados recursivos para mostrar armonías por pantalla.

mostrar(A,C) :- write('Cifrado'),write('\t\tNotas del acorde'),salto,mostrar_frase(A,C).

mostrar_frase([],[]).
mostrar_frase([A|AS],[C|CS]) :- mostrar_semifrase(A,C),mostrar_frase(AS,CS).

mostrar_semifrase([],[]) :- write('(reposo)'),salto.
mostrar_semifrase([A|AS],[C|CS]) :- write(C), write('\t\t'),write(A),salto,mostrar_semifrase(AS,CS).

% 10. Interfaz de usuario

bach-machine :- salto,cabecera,generador.
generador:- io(tono),coral,io(fin).

salto :- writeln('').
```

```
cabecera :-
writeln('=============================================================================================================='),salto,
writeln('                        Bach-machine, su generador "inteligente" de armonías corales                          '),salto,
writeln('=============================================================================================================='),salto,
writeln('Para generar la armonía de un coral, se le solicitará introducir el tono en el que estará el coral.'),
writeln('La nota correspondiente al tono se introduce en minúsculas seguida de un punto cuando es natural:'),salto,
writeln('\tDo Mayor -> do.'),salto,
writeln('En caso de que el tono sea una nota alterada, se utilizará _s para los sostenidos y _b para los bemoles:'),salto,
writeln('\tSi bemol Mayor -> si_b.'),
writeln('\tDo sostenido Mayor -> do_s.'),salto,
writeln('Nota: El sistema trabaja con tonalidades desde 7 bemoles a 7 sostenidos.'),salto.

io(tono) :- salto,write('Introduzca el tono para generar un coral\t-> '),read(T),salto,
            (tono_valido(T) -> assert(tono(T));io(tono,fallo)).

io(fin) :- salto,write('\t¿Desea probar otra vez? (si/no) '),read(R),salto,retractall(tono(_)),io(fin,R).

io(fin,si):- generador.
io(fin,no):- writeln('Gracias por utilizar Bach-machine, ¡Hasta pronto!').

io(OP,_):- write('-> Error al introducir la opción'),io(OP).

%EOF
```

5. Casos de prueba

- Intervalos con una variable y todas las escalas mayores

```
?- seg_min(do,X).
X = re_b.

?- cua_jus(sol_s,X).
X = do_s.

?- esc_maj(Tónica,Escala).
Tónica = do_b,
Escala = [do_b, re_b, mi_b, fa_b, sol_b, la_b, si_b] ;
Tónica = do,
Escala = [do, re, mi, fa, sol, la, si] ;
Tónica = do_s,
Escala = [do_s, re_s, mi_s, fa_s, sol_s, la_s, si_s] ;
Tónica = re_b,
Escala = [re_b, mi_b, fa, sol_b, la_b, si_b, do] ;
Tónica = re,
Escala = [re, mi, fa_s, sol, la, si, do_s] ;
Tónica = mi_b,
Escala = [mi_b, fa, sol, la_b, si_b, do, re] ;
Tónica = mi,
Escala = [mi, fa_s, sol_s, la, si, do_s, re_s] ;
Tónica = fa_b,
Escala = [fa_b, sol_b, la_b, si_b_b, do_b, re_b, mi_b] ;
Tónica = fa,
Escala = [fa, sol, la, si_b, do, re, mi] ;
Tónica = fa_s,
Escala = [fa_s, sol_s, la_s, si, do_s, re_s, mi_s] ;
Tónica = sol_b,
Escala = [sol_b, la_b, si_b, do_b, re_b, mi_b, fa] ;
Tónica = sol,
Escala = [sol, la, si, do, re, mi, fa_s] ;
Tónica = la_b,
Escala = [la_b, si_b, do, re_b, mi_b, fa, sol] ;
Tónica = la,
Escala = [la, si, do_s, re, mi, fa_s, sol_s] ;
Tónica = si_b,
Escala = [si_b, do, re, mi_b, fa, sol, la] ;
Tónica = si,
Escala = [si, do_s, re_s, mi, fa_s, sol_s, la_s].
```

- Funciones tonales e inversiones

```
?- ton_maj(re,X).
X = [re, fa_s, la].

?- ton_maj(re,AcordeEnEstadoFundamental),inv_64(AcordeEnEstadoFundamental,AcordeInvertido).
AcordeEnEstadoFundamental = [re, fa_s, la],
AcordeInvertido = [la, re, fa_s].
```

- Semifrases aleatorias

```
?- semifrase_modulante(sol,(Acordes,Cifrado)).
Acordes = [[sol, si, re], [fa_s, si, re], [fa, sol, si, re], [mi, sol, do]],
Cifrado = ["T", "III64", "D/S2", "S6"] ;


?- semifrase_modulante(sol,(Acordes,Cifrado)).
Acordes = [[sol, si, re], [fa_s, la, re], [fa_s, la, do, mi_b], [sol, si, mi]],
Cifrado = ["T", "D6", "VIIdis", "VI6"] ,

?- semifrase_conclusiva_sub(la,(Acordes,Cifrado)).
Acordes = [[la, do_s, fa_s], [la, do_s, mi], [la, re, fa_s], [sol_s, si, re, mi], [la, do_s, mi]],
Cifrado = ["SVI6", "T", "S64", "D65", "T"] ,

?- semifrase_conclusiva_sub(la,(Acordes,Cifrado)).
Acordes = [[si, re, fa_s], [la, do_s, mi], [sol_s, si, re, mi], [la, do_s, mi]],
Cifrado = ["SII", "T", "D65", "T"] ,
```

- Semifrases concretas

```
?- cad_DD(sol,(Acordes,Cifrado)).
Acordes = [[sol, si, re], [mi, la, do], [mi, sol, la, do_s], [fa_s, la, re]],
Cifrado = ["T", "SII64", "D/D43", "D6"] ,

?- cad_perf(fa_s,(Acordes,Cifrado)).
Acordes = [[fa_s, la_s, do_s], [fa_s, si, re_s], [mi_s, sol_s, si, do_s], [fa_s, la_s, do_s]],
Cifrado = ["T", "S64", "D65", "T"]
```

- Traza de formación de una función tonal. Se ha elegido una función que resulta en un acorde que comienza en re (todas las declaraciones de hechos comienzan por los referidos a la nota do que es la anterior a re) para reducir un poco las vueltas atrás y el tamaño de la traza:

```
[trace]  ?- dom_dom(do,Acorde).
   Call: (10) dom_dom(do, _3356) ? creep
   Call: (11) qui_jus(do, _3814) ? creep
   Exit: (11) qui_jus(do, sol) ? creep
   Call: (11) dom(sol, _3356) ? creep
   Call: (12) seg_maj(sol, _3904) ? creep
   Exit: (12) seg_maj(sol, la) ? creep
   Call: (12) perf_maj([_3892, _3898, la]) ? creep
   Call: (13) ter_maj(_3892, _3898) ? creep
   Exit: (13) ter_maj(do_b, mi_b) ? creep
   Call: (13) ter_min(mi_b, la) ? creep
   Fail: (13) ter_min(mi_b, la) ? creep
   Redo: (13) ter_maj(_3892, _3898) ? creep
   Exit: (13) ter_maj(do, mi) ? creep
   Call: (13) ter_min(mi, la) ? creep
   Fail: (13) ter_min(mi, la) ? creep
   Redo: (13) ter_maj(_3892, _3898) ? creep
   Exit: (13) ter_maj(do_s, mi_s) ? creep
   Call: (13) ter_min(mi_s, la) ? creep
   Fail: (13) ter_min(mi_s, la) ? creep
   Redo: (13) ter_maj(_3892, _3898) ? creep
   Exit: (13) ter_maj(re_b, fa) ? creep
   Call: (13) ter_min(fa, la) ? creep
   Fail: (13) ter_min(fa, la) ? creep
   Redo: (13) ter_maj(_3892, _3898) ? creep
   Exit: (13) ter_maj(re, fa_s) ? creep
   Call: (13) ter_min(fa_s, la) ? creep
   Exit: (13) ter_min(fa_s, la) ? creep
   Exit: (12) perf_maj([re, fa_s, la]) ? creep
   Exit: (11) dom(sol, [re, fa_s, la]) ? creep
   Exit: (10) dom_dom(do, [re, fa_s, la]) ? creep
Acorde = [re, fa_s, la] .
```

- Cabecera del programa y menú principal. La llamada al menú principal se realiza mediante la pregunta `bach-machine`.

```
% c:/Learn/Uned/fia/pec/bach-machine.pl compiled 0.03 sec, 234 clauses
?- bach-machine.

================================================================================================================

                        Bach-machine, su generador "inteligente" de armonías corales

================================================================================================================

Para generar la armonía de un coral, se le solicitará introducir el tono en el que estará el coral.
La nota correspondiente al tono se introduce en minúsculas seguida de un punto cuando es natural:

        Do Mayor -> do.

En caso de que el tono sea una nota alterada, se utilizará _s para los sostenidos y _b para los bemoles:

        Si bemol Mayor -> si_b.
        Do sostenido Mayor -> do_s.

Nota: El sistema trabaja con tonalidades desde 7 bemoles a 7 sostenidos.


Introduzca el tono para generar un coral        -> si_b.

Cifrado         Notas del acorde
T               [si_b,re,fa]
SII             [do,mi_b,sol]
S64             [si_b,mi_b,sol]
D65             [la,do,mi_b,fa]
T               [si_b,re,fa]
(reposo)
SII             [do,mi_b,sol]
SVI6            [si_b,re,sol]
S64             [si_b,mi_b,sol]
T               [si_b,re,fa]
(reposo)
T               [si_b,re,fa]
III64           [la,re,fa]
D/S2            [la_b,si_b,re,fa]
S6              [sol,si_b,mi_b]
(reposo)
S64             [si_b,mi_b,sol]
T               [si_b,re,fa]
D65             [la,do,mi_b,fa]
T               [si_b,re,fa]
(reposo)

        ¿Desea probar otra vez? (si/no) |: si.


Introduzca el tono para generar un coral        -> |: si_b.
```

```
Introduzca el tono para generar un coral        -> |: si_b.

Cifrado         Notas del acorde
T               [si_b,re,fa]
S64             [si_b,mi_b,sol]
D64             [fa,si_b,re]
D7              [fa,la,do,mi_b]
T               [si_b,re,fa]
(reposo)
S64             [si_b,mi_b,sol]
TIII            [re,fa,la]
S               [mi_b,sol,si_b]
T               [si_b,re,fa]
(reposo)
T               [si_b,re,fa]
III64           [la,re,fa]
D/S2            [la_b,si_b,re,fa]
S6              [sol,si_b,mi_b]
(reposo)
SVI6            [si_b,re,sol]
T               [si_b,re,fa]
S64             [si_b,mi_b,sol]
D65             [la,do,mi_b,fa]
T               [si_b,re,fa]
(reposo)

        ¿Desea probar otra vez? (si/no) |: no.

Gracias por utilizar Bach-machine, ¡Hasta pronto!
true .
```

6. Dificultades encontradas

La principal dificultad sin duda ha sido asimilar el paradigma de programación lógica. Aunque tiene semejanzas a otro paradigma de estilo declarativo como pueda ser el funcional respecto a la recursividad y el tratamiento de las listas, la forma de pensar y estructurar mentalmente el flujo de información es totalmente distinto, hay que pensar primero en el resultado final —los consecuentes— y después ir acumulando las condiciones que se deben cumplir. Sin embargo, una vez asimilado se tornó bastante intuitivo y la dificultad se convirtió en acotar los límites del problema para poder elaborar un sistema funcional manejable y acorde al objetivo del trabajo.

En este caso, el problema da muchísmo más de si, tanto en complejidad y riqueza armónica como en funcionalidades adicionales, especialmente la de que el coral generado "suene".

Como paso previo a esta funcionalidad, aunque se sale del ámbito de la práctica, he intentado conectar el sistema de reglas de Prolog con Java mediante el módulo JPL, pero presenta muchos problemas de versiones. Finalmente me he decantado por una librería en JavaScript denominada Tau-Prolog, realizada por José Antonio Riaza, doctorando e investigador en el campo de la programación lógica en la UCLM. La versión web del sistema de reglas de esta pec que utiliza Tau-Prolog para realizar las consultas, está disponible, de momento, en:

https://balath.github.io/