

Contents

1	Casos de estudio	1
2	La cuantización	1
3	La expansión	3
4	Nuevo enfoque	4
4.1	La expansión	6
5	Hacia la composición	8
6	El programa METAFONT	9
7	Por el compás completo	14
7.1	Everynote	17
7.2	Se me pasó.	17
7.3	El compás.	17
8	Generalización	18
8.1	Plicas	18
8.2	Espacio	18
8.3	Líneas adicionales	19
8.4	Silencios	19
9	Se acaba el compás. . .	19
9.1	Reflexión	20
9.2	Vuelta a la barra	24
10	Aquí de nuevo	24
10.1	‘Insertos’	25
10.2	Factores primos	25
11	Por hacer	28
12	El espacio. . .	28

¿Cómo hacer para que $\text{T}_{\text{E}}\text{X}$ pueda leer el código del usuario? Sobre todo estoy pensando en que el usuario debe poder definir sus nuevos comandos sin ninguna restricción. Esto implica una etapa intermedia en que lo que el usuario escribió se expande y se convierte en el código musical real. Más aún, si esto fuera así, se podría hacer que la expansión tuviera diferentes efectos para cada una de las tareas (cuantización y composición del texto musical).

Esto sugiere el concepto de ‘comandos primarios’ de $\text{T}_{\text{E}}\text{X}_{\text{m}\mu\text{e}}$. Todos ellos tienen dos significados, uno para la cuantización, otro para la composición. Ninguno contiene minúsculas, que se reservan para las notas.

1 Casos de estudio

El usuario escribió

Escrito
27/12/2002

```
|3\#cedf||egfa||4.g2fe|\bar|3fd||3.d1cd|  
|3af||3.f1ef||3d+\b ba\#g|\bar
```

Esto en realidad está bastante cerca de una serie de comandos primarios. Los únicos problemas son `\b` y `\bar`. El primero debe expandirse como `\FLAT`, el segundo como `\BAR`, y el texto final es

```
|3\#cedf||egfa||4.g2fe|\BAR|3fd||3.d1cd|  
|3af||3.f1er||3d+\FLAT ba\#g|\BAR
```

La mano izquierda fue

```
|4aa+e\#c||ag|\bar|fa+d\b bga|
```

que debe dar algo como

```
|4aa+e\#c||ag|\BAR|fa+d\FLAT bga|
```

Pero como se repite, probablemente el usuario defina

```
\newcommand{\fandango}{%  
|4aa+e\#c||ag|\bar|fa+d\b bga|}
```

y escriba simplemente `\fandango`. Entonces, \TeX debe expandir lo mismo.

2 La cuantización

El texto de comandos primarios es entonces leído por \TeX en la primera etapa, la de ‘cuantización.’ Lo único relevante aquí es lo que tiene que ver con la construcción de la matriz, y el (los) archivo(s) `tmx`.

Escrito
27/12/2002

Para esta primera versión de \TeX , voy a usar la convención ya imaginada: número negativo significa algo hacia la izquierda; número decimal algo hacia la derecha. `METAFONT` es el encargado de encontrar el ancho real del ‘algo,’ basado en la matriz. \TeX no necesita esta información más que para calcular la longitud natural del texto musical; en principio, esto puede ser una aproximación relativamente burda, por ejemplo la suposición de que todos los ‘algos’ tienen, en promedio, un ancho igual a una nota. En refinamientos posteriores se puede inventar un mecanismo para ir calculando este promedio según el texto mismo.

Entonces, \TeX lee el ‘texto primario’ con la única intención de escribir en el archivo `tmx` una cadena de números. Cada vez que encuentra una nota, incrementa el número de la nota anterior, y la escribe. En principio, los números son caracteres activos que modifican la cantidad a añadir: `\current@music@skip`. El punto hace lo mismo. Las alteraciones hacen negativo al número, lo mismo que las claves.

El caracter < invoca una función (cuyo argumento está delimitado por >) para hallar si un acorde contiene segundas y/o alteraciones, cambiando el signo del número y/o añadiendo un decimal. Esa función habrá que escribirla después.

| tiene una función especial: hace que las figuras de valor menor que negra *no* pongan un número decimal (porque no tienen banderita). Cuando vuelve a aparecer, hace que ahora *sí* lo pongan.

Entonces, en el caso de

```
|3\#cedf||egfa||4.g2fe|\BAR|3fd||3.d1cd||3af||3.f1er||3d+\FLAT ba\#g|\BAR
```

vamos a hacer `\current@quantum=1, \flagstrue` y `\quantum@skip=0`.

El sistema que (¡al fin!) encontré para definir números en términos de números funcionó perfectamente. Pero después me di cuenta de que aplicar lo mismo a las letras sería absurdo. La convención se revierte: el mecanismo de expansión convierte todo a mayúsculas, y convierte los números en otros signos...

El grave problema es que, de todos modos, las letras deben ser definidas como activas, y entonces no hay manera de controlar a T_EX. Tres soluciones son posibles:

Escrito
28/12/2002

1. Hacer que T_EX escriba el ‘texto primario’ en un archivo. Entonces la conversión a mayúsculas sí tiene lugar.
2. Pedirle al usuario que escriba las notas en mayúscula.
3. Definir comandos alternativos, en mayúsculas, para controlar a T_EX: por ejemplo, `\let\CATCODE\catcode`, etc.

Vamos a seguir la tercera opción, a ver que pasa: el resultado de

```
|3\#cedf||egfa||4.g2fe|\BAR|3fd||3.d1cd|  
|3af||3.f1er||3d+\FLAT ba\#g|\BAR
```

es

-1, 17, 33, 49, 65, 81, 97, 113, 129, 177, 185, 1, 17, 33, 57, 61, 65, 81, 97, 121, 125, 129, +-145, 161, -177,

Esto está, en principio, bien. Es mejor definir dos condicionales para los ‘algos’ a la izquierda y derecha. Así, la multiplicación por -1 no afecta el siguiente valor. Por otro lado, esto tiene el peligro de que no se sabe qué comandos entren en juego en estas definiciones (y en las de la segunda etapa, sobre todo). Cómo hacer, por ejemplo, que T_EX pueda escribir un texto para METAFONT principalmente en minúsculas? Seguramente hay una manera, pero en realidad esto es aterradorante...

Habiendo visto que la opción 3 es posible en principio, estoy considerando ahora la posibilidad de la opción 1. El mecanismo de expansión escribiría el texto primario en un archivo. Justamente, durante ese primer paso, los comandos primarios de T_EX^{ms} significaran `\write...`

Escrito
30/12/2002

3 La expansión

Tomemos para este caso la mano izquierda, que tiene el ejemplo de un comando definido por el usuario. La expansión se llevaría a cabo por medio de algo como `\musexpand{#1}`, de tal manera que

Escrito
30/12/2002

```
\musexpand{\fandango}
```

produciría, en un archivo externo (nótese que ahora los comandos son en minúscula),

```
|4AA+E\#C||AG|\bar|FA+D\flat BGA|
```

¡Esto parece mucho más sencillo! Por medio de `\uppercase` y una definición adecuada de `\#`, `\b`, `\bar`, etc., esto se lleva a cabo en un paso. Lo único que no está solucionado es el problema de los comandos \LaTeX que el usuario pueda insertar (dentro de "", o algo así).

Entonces, probemos: voy a escribir `|4AA+E|` en el archivo `expansion.tst`. En efecto, quedó escrito. ¿Qué tal `\uppercase{|4aa+e|}`? Pues quedó eso, literalmente. Con `\expandafter{\uppercase{|4aa+e|}}` pasa lo mismo. Igual con otras versiones parecidas... Pero, ¡eureka!, `\uppercase\write...` funciona.

El siguiente problema es el de los comandos. `\write\expansion{|4aa+e\#c|}` no expande el comando. Pero en realidad esto se debe a que ese comando no se expande naturalmente en ASCII. Si lo redefino como `{\hola}`, produce `|4aa+eholac|`. Entonces mejor lo redefino como `{\string\#}`, a ver qué pasa: efectivamente, sí sirve.

En realidad, lo único que se necesita es hacer que los comandos sean inexpandibles; por ejemplo, con `\let...\relax`. Entonces, he aquí las nuevas definiciones para el `fandango`:

```
\let\b\relax
\let\#\relax
\let\bar\relax
\newcommand{\fandango}{|4aa+e\#c||ag|\bar|fa+d\b bga|\bar}
```

Haciendo `\uppercase{\immediate\write\expansion{\fandango}}`, el archivo contiene una nueva línea: `|4aa+e\#c||ag|\bar |fa+d\b bga|\bar .` Pero, por supuesto, ¡`\uppercase` no afecta a `\fandango`!

Antes de tener que aceptar la opción 2, vamos a intentar con *tokens*:

```
\pruebaa{\uppercase{\fandango}}
\immediate\write\expansion{\the\pruebaa}
```

produce `\uppercase {\fandango }`, mientras que

```
\edef\pruebaa{\uppercase{\fandango}}
\immediate\write\expansion{\pruebaa}
```

da `\uppercase {|4aa+e\#c||ag|\bar |fa+d\b bga|\bar }`. Otras opciones tampoco sirven.

Definitivamente, entonces, hay que pedirle al usuario que ponga las letras en mayúscula. Pero de todos modos el mecanismo de `\write` es excelente para implementar los comandos primarios de \TeX *Music*. Al fin y al cabo, la cuantización y la composición del texto musical, es decir, la comunicación entre \TeX y METAFONT, no necesitan más que esos comandos.

4 Nuevo enfoque

En los últimos días he cambiado casi por completo el modelo. La sugerencia de usar un archivo para expandir los comandos hasta primarios de \TeX *Music* me puso a pensar cómo hacer para optimizar el uso de archivos (por ejemplo, con la idea de que el mismo archivo que al principio tiene los primarios es el que va a tener los datos de cuantización de cada instrumento). Pero entonces descubrí que, en el archivo de cuantización, que originalmente era una línea por cada instrumento, una columna por cada nota, es mejor efectuar una rotación: el primer instrumento en la primera columna, el segundo en la segunda. Pero la realización de esta idea hasta sus últimas consecuencias lleva al diseño *completo* de la matriz: cada línea (que corresponde a una verticalidad de la partitura) contiene sólo números para aquellos instrumentos que efectivamente participan en la nota. Una columna inicial se reserva para el quantum de cada nota (para calcular a partir de ella el `\sfcode` de cada carácter), y entonces los números para los instrumentos ya no tienen que especificarla: pueden dedicarse a los elementos que se salen del carácter, un dígito para la izquierda, uno para la derecha. El uso de `\prev@char` y demás ya no es necesario.

Otra buena consecuencia es que los cuanta ya no necesitan ser enteros, los decimales pueden significar fracciones. Esto puede ser muy útil para los “illos”...

El mecanismo es el siguiente: hay dos archivos auxiliares, `.tm1` y `.tm2`. El primer instrumento escribe `.tm1`; el segundo lee la primera nota, compara con la primera línea de `.tm1`, y escribe el resultado (una o dos filas, dependiendo de si ambos instrumentos participan en la primera nota o no) en `.tm2`; el siguiente instrumento lee de `.tm2` y escribe en `.tm1`, etc.

Entonces voy a empezar a implementar el sistema por completo. Primero necesito hacerlo con sólo notas, sin alteraciones, claves, ni nada especial: por ejemplo, el segundo compás de la primera invención de Bach. El texto de entrada es:

```
\newinstrument{righthand}
\newinstrument{lefthand}
\righthand{\rangefrom{G4}|3DGAB||CABG|\rangefrom{D5}|4DGFG|\bar}
\lefthand{\rangefrom{G3}|4GG-|5r3r|GAB||CABG|\bar}
```

Nótese el cambio en el uso de `\rangefrom`: un sólo argumento, porque se especifica *dentro* de cada instrumento.

Escrito
04/01/2003

El archivo `bach.tm1` debería quedar así (11 significa nota sin nada a la izquierda ni a la derecha):

```
0: 11\end@of@line
16: 11\end@of@line
32: 11\end@of@line
48: 11\end@of@line
64: 11\end@of@line
80: 11\end@of@line
96: 11\end@of@line
112: 11\end@of@line
128: 11\end@of@line
160: 11\end@of@line
192: 11\end@of@line
224: 11\end@of@line
256: \@bar\end@of@line
```

Entonces, `\lefthand` empieza a ejecutar: la primera nota cae en 0, y vale 32. O sea que una primera línea será 0: 11, 11. La siguiente nota cae en 32, o sea que hay que poner algo en 16: 16: 11, `\skip@note`. Después, 32: 11, 11. Y así, entonces el archivo `.tm2` queda:

```
0: 11, 11\end@of@line
16: 11, \skip@note\end@of@line
32: 11, 11\end@of@line
48: 11, \skip@note\end@of@line
64: 11, 11\end@of@line
80: 11, \skip@note\end@of@line
96: 11, \skip@note\end@of@line
112: 11, \skip@note\end@of@line
128: 11, 11\end@of@line
144: \skip@note, 11\end@of@line
160: 11, 11\end@of@line
176: \skip@note, 11\end@of@line
192: 11, 11\end@of@line
208: \skip@note, 11\end@of@line
224: 11, 11\end@of@line
256: \@bar, \@bar\end@of@line
```

Por el momento, vamos a implementar eso. El macro que manda a escribir datos a los archivos funciona básicamente así: se saben `\current@quantum` y `\instrument@number`. Se lee una línea del archivo de lectura, y se extrae el quantum que representa (`\extracted@number`). Después de la comparación, se define y se escribe el macro `\to@write`. Si acaso `\current@quantum` es mayor que `\extracted@number`, se escribe la línea actual, añadiéndole un `\skip@note`, y se lee la siguiente línea del archivo. Cuando pasa lo contrario, es decir que el instrumento actual *inserta* una nota, tiene que escribir tantos `\skip@note`'s como instrumentos anteriores haya, y además tiene que desactivar la lectura de la siguiente línea (porque el siguiente valor de `\current@quantum` se va a comparar con la línea ya leída). Esa es la función de `\ifrepeat@test`. El macro es el siguiente:

```
\newif\ifrepeat@test\repeat@testfalse
```

```

\def\@note{\ifrepeat@test\else\read\read@tm@file to \old@line%
\expandafter\extract@from@line\old@line\fi%
\repeat@testfalse%
\let\next\relax%
\ifnum\extracted@number=\current@quantum
\edef\to@write{\the\extracted@number:\the\extracted@line,%
11\string\end@of@line}%
\else\ifnum\extracted@number>\current@quantum
\repeat@testtrue%
\temp=\instrument@number
\edef\to@write{\the\current@quantum:\noexpand@gobble}%
\loop\ifnum\temp>1 \edef\to@write{\to@write,\string\skip@note}%
\advance\temp-1 \repeat
\edef\to@write{\to@write,11\string\end@of@line}%
\else%
\let\next\@note%
\edef\to@write{\the\extracted@number:%
\the\extracted@line,\string\skip@note\string\end@of@line}%
\fi\fi
\immediate\write\written@tm@file\expandafter{\to@write}%
\next}
\def\extract@from@line#1:#2\end@of@line{\extracted@number=#1\relax\extracted@line{#2}}

```

y funciona a las mil maravillas, siempre y cuando el archivo de lectura exista y no se termine a destiempo. Ahora bien, siempre va a existir desde el segundo instrumento; y se hace la convención de que al final de los archivos hay siempre una línea 9999: `\skip@note\end@of@note`. Si se está haciendo el primer instrumento, antes de empezar todo se escribe un archivo con esa misma línea. Y si alguna vez `\current@quantum` llega a 9999, se emite un error.

Nótese: como está ahora, cuando el instrumento actual tiene menos notas que los ya escritos, el macro no escribe las líneas para las notas siguientes.

Escrito
09/02/2003

4.1 La expansión

El proceso de expansión por medio de `\write` hace que se escriban tantos archivos auxiliares como instrumentos hay en la partitura. Estos archivos se escriben por bloques, generalmente equivalentes a compases. Cuando se está componiendo el texto, sendas variables contendrán el bloque actual.

Escrito
04/01/2003

Es en el proceso de expansión que los números son traducidos a comandos, y que, en todo caso, las letras minúsculas se convertirían a mayúsculas. Estos archivos deben contener *toda* la información que se va a METAFONT. En particular, `|` debe ser traducido inmediatamente. Hay variables `\everynote` y `\afternote`, y una de las misiones de `|` es añadir `\add@tobeam` a la primera; el `|` que cierra lo que hace es añadir `\close@beam` a la segunda. Los comandos de dinteles tendrán un argumento opcional para la clave del dintel, que por defecto es el nombre del instrumento. El argumento obligatorio de `\add@tobeam` es el número de dinteles en la nota particular. `\next@note` separa las notas.

Entonces, el resultado de `\righthand` en la invención debe ser algo como

```

\next@note\open@beam\sixteenth@notes\add@tobeam{2}\add@notes{d5}
\next@note\add@tobeam{2}\add@notes{g4}

```

```

\next@note\add@tobeam{2}\add@notes{a5}
\next@note\add@tobeam{2}\add@notes{b5}\close@beam
\next@note\open@beam\add@tobeam{2}\add@notes{c5}
\next@note\add@tobeam{2}\add@notes{a5}
\next@note\add@tobeam{2}\add@notes{b5}
\next@note\add@tobeam{2}\add@notes{g4}\close@beam
\next@note\open@beam\eight@notes\add@tobeam{1}\add@notes{d5}
\next@note\add@tobeam{2}\add@notes{g5}
\next@note\add@tobeam{2}\add@notes{f5}
\next@note\add@tobeam{2}\add@notes{g5}\close@beam

```

todo, claro está, en una sola línea. He puesto los `\next@note` al principio de cada nota porque en realidad se añade al empezar cada nota, inmediatamente antes de `\everynote`.

Caí en cuenta de que todas las notas deben indicar la figura rítmica. No puede haber una declaración global para eso, porque cambia de instrumento a instrumento. Más aún, no se puede depender tanto de `\everynote`, porque en el modelo descrito muchas cosas cambian ese valor, y crean conflicto entre sí. Por el momento, sin embargo, vamos a implementar `|` como se ha dicho.

Escrito
05/01/2003

Ya no va a haber argumento para `\add@tobeam`, porque todas las notas tienen la figura rítmica. Hay un problema con los macros del usuario: no tienen efecto si no fueron definidos bajo el alcance de los `\catcodes` especiales, *ni siquiera definidos como tokens*.

Escrito
09/01/2003

Una cosa más, sobre el mecanismo de las definiciones. Los comandos especiales, aquellos cuyos nombres generalmente no son activos, deben ser definidos dentro de un grupo (pero `\global`). El usuario teclea los comandos de cada instrumento dentro de un entorno que se llama como éste, y cuya función es cambiar los códigos de carácter.

Por otro lado, `\everynote` se implementa como un macro de lista (Ap. D en el *TeXBook*), en realidad como `\@every@note`. Por convención, `\addto@beam` va a la derecha de la lista, `\son` y demás indicadores de valor rítmico van a la izquierda. Esto, sin embargo, queda aplazado para después: por el momento no hay cosas como `\staccato`, etc.

5 Hacia la composición

Bueno, ya se puede suponer que `TeX` produjo archivos `.tm*` adecuados. Y ahora, ¿qué? Ahora viene lo más bonito: `TeX` programa a `METAFONT`.

Escrito
09/02/2003

Debe partir de la matriz: la primera línea indica “0: 11,11\end@of@line” y significa que debe haber notas en ambos instrumentos. La primera nota en `\righthand` es “\open@beam\add@tobeam\s@n\add@note{d5}” (esto lo saqué directamente del resultado de los macros de expansión); en `\lefthand` (inventado) es “\open@beam\add@tobeam\e@n\add@note{g3}”.

El resultado en `METAFONT` debe ser algo como el despliegue que sigue. Nótese: `find_pos(sfcode)` encuentra (en `METAFONT`), para la nota actual, el `spacefactor`, el `width`, y el `stretched_width`, para lo que usa varios valores que todavía `TeX` no le ha dado. Esto sale del ejemplo del artículo (el *Fandango*).

Escrito
13/02/2003

La segunda línea de cada pentagrama es 0 respecto de él. Las notas corresponden, por defecto, a la clave de sol: `g4` tiene un valor de 0. Estas correspondencias están incluidas en `texmuse.mf`. Pero cada pentagrama, en METAFONT, tiene unas constantes, a saber el desplazamiento vertical hacia arriba (que T_EX encuentra sólo al construir la línea entera), y la conversión de las notas por diferentes claves. En este caso, el pentagrama inferior, clave de fa, tiene una conversión de 6 (`g4=6`). El comando `instrument` hace que METAFONT cambie el valor de las constantes.

`add_notes` pone cabezas, por defecto centradas horizontalmente. En concreto, las pone en una posición horizontal conocida, que T_EX puede cambiar si es necesario.

`staff` es un caso especial de `staff_lines`, a saber el de 5 líneas.

```
beginchar(1,2framing_space+horizontal_axis,gral_height,gral_depth);
  find_pos(1);
  instrument(1);
  add_notes(g3);
  staff;
  instrument(2);
  add_notes(d5);
  staff;
endchar;
```

Además, gracias a `\open@beam`, se definen los registros `\lefthand@beam` y `\righthand@beam`, que van a tener las coordenadas de las notas bajo dintel. Por ahora, como sólo hay una nota por voz, van a ser tres coordenadas: el índice de la nota, el número de dinteles, y la nota misma. Eventualmente se tendrá que hacer de cinco coordenadas, añadiendo la nota superior y el número de instrumento. Antes de eso, sin embargo, se tiene que lograr que T_EX pueda ordenar las notas de un mismo acorde de abajo para arriba (esto también se necesita para pillar notas que deben ser puestas hacia un lado). Entonces, por ahora, `\lefthand@beam{1,1,g3}`, `\righthand@beam{1,2,d5}`;

La segunda nota sólo tiene mano derecha¹. Por otro lado, todavía no se ha implementado el argumento opcional de las notas, - ó +; esto debe estar en `\@add@note`. La segunda nota es: ⇐

```
beginchar(2,2framing_space+horizontal_axis,gral_height,gral_depth);
  find_pos(1);
  instrument(1); staff;
  instrument(2);
  add_notes(g4);
  staff;
endchar;
```

Además, `\righthand@beam` es ahora `1,2,d5,2,2,g4`.

¹Aquí noto algo importante: al construir la matriz (archivos `.tm*`) se debe partir del pentagrama inferior.

Antes de enviar la tercera nota ya se sabrá que el nuevo `\lefthand@beam` es `1,1,g3,3,1,g2` (y lo correspondiente para `\righthand@beam`), entonces se puede llamar la rutina `beam` (sigue también la cuarta nota):

```
beginchar(3,2framing_space+horizontal_axis,gral_height,gral_depth);
  find_pos(1);
  instrument(1);
  add_notes(g2);
  staff;
  instrument(2);
  add_notes(a5);
  beam(1,1,g3,3,1,g2);
  staff;
endchar;
beginchar(4,2framing_space+horizontal_axis,gral_height,gral_depth);
  find_pos(1);
  instrument(1);
  staff;
  instrument(2);
  add_notes(b5);
  beam(1,2,d5,2,2,g4,3,2,a5,4,2,b5);
  staff;
endchar;
```

Supongamos que aquí termina la línea. Hay que determinar la separación entre los pentagramas, para lo cual probablemente lo mejor es que METAFONT vaya recordando la nota más alta de cada uno. Lo mismo pasará con `gral_depth` y `gral_height`. Por otro lado, hay que determinar los `\sfcodes`, y la longitud total de la caja que contiene a la música.

Escrito
15/02/2003

6 El programa METAFONT

Pero por ahora me voy a concentrar en el programa METAFONT: `texmuse.mf` (en estas pruebas el archivo se llamará `txmstst.mf`). Este archivo contiene las rutinas de cálculo, los “macros”. Las variables `picture`, es decir la fuente misma, está en `tmsfont`. El usuario puede especificar otro archivo con algún comando para el efecto. La fuente se carga antes de los macros, porque éstos usan aquélla.

Escrito
15/02/2003

Lo siguiente viene del artículo:

```
numeric index[];
numeric spacefactor[];
numeric width[];
numeric stretched_width[];
def find_pos(expr sfcode)=
  index:=index+1;
  spacefactor[index]=sfcode;
```

```

width[index]=w;
stretched_width[index]:=w+spacefactor[index]%
*0.6total_natural_length/total_space_factors;
enddef;

```

Nótese que la constante 0.6 aquí está sujeta a cambiar, probablemente a convertirse en una función de las notas que hay en la línea. Pero por ahora la dejo como está. ⇐

`total_natural_length` y `total_space_factors` se declaran y definen al terminar cada línea, y se insertan antes de los caracteres.

```

numeric vert_shift, instr_shift[];
numeric clef_conversion, instr_clef[];
def instrument(expr number) =
  vert_shift:=instr_shift[number];
  clef_conversion:=instr_clef[number];
enddef;

numeric a[], b[], c[], d[], e[], f[], g[];
for octave=1 step 1 until 8:
  g[octave]=3.5*(octave-4);
  a[octave]=g[octave]+.5;
  b[octave]=g[octave]+1;
  c[octave]=g[octave]+1.5;
  d[octave]=g[octave]+2;
  e[octave]=g[octave]+2.5;
  f[octave]=g[octave]+3;
endfor;
def add_notes (text nhs) =
  for s=nhs:
    fill note_head%
    shifted(w/2,interline*(s+clef_conversion+vert_shift));
  endfor; enddef;

```

Por supuesto, `add_notes` tendrá que cambiar para permitir notas no centradas. ⇐

```

def staff = staff_lines(-1,0,1,2,3);
def staff_lines(text t) =
  pickup pencircle scaled line_thick;
  for i=t:
    draw (0,(i+vert_shift)*interline)--(stretched_width[index],(i+vert_shift)*interline);
  endfor;
enddef;

```

Y ahora viene lo interesante: `beam`. Lo primero es extraer grupos de tres argumentos, para mandárselos a `make_beam`:

```

def beam(text t) =
  j:=0;

```

```

for i=t:
  j:=j+1;
  for_beam[j]=i;
endfor;

```

Después hay que encontrar las notas extremas, para decidir la dirección de las plicas:

```

top_beam_note:=-99; bottom_beam_note:=99;
for i=3 step 3 until j:
  if top_beam_note < for_beam[i] : top_beam_note:=for_beam[i]; fi;
  if bottom_beam_note > for_beam[i] : bottom_beam_note:=for_beam[i]; fi;
endfor;
if (top_beam_note-1)>=(bottom_beam_note+1):
  beam_lift:=bottom_beam_note-stem_length;
  stem_hor_shift:=-horizontal_axis;
  coef:=-1;
else:
  beam_lift:=top_beam_note+stem_length;
  stem_hor_shift:=horizontal_axis;
  coef:=1;
fi;
curr_part:=index;
for i=j step -3 until 1:
  make_beam(for_beam[i-2], for_beam[i-1],%
    for_beam[i]+clef_conversion+vert_shift);
endfor;
enddef;

```

Y, finalmente, make_beam:

```

def make_beam(expr ind, beams_no, note) =
  hor_start:=w/2+stem_hor_shift;
  hor_end:=hor_start;
  for i=index-1 step -1 until curr_part:
    hor_start:=hor_start-stretched_width[i];
  endfor;
  for i=index step -1 until ind:
    hor_end:=hor_end-stretched_width[i];
  endfor;
  beam_total_height:=(3*beams_no-1)/4;
  pickup penrazor xscaled(line_thick);
  draw (hor_start, interline*(beam_lift+beam_total_height))--(hor_start, note*interline);
  pickup penrazor xscaled (beam_total_height*interline) rotated 90;
  draw (hor_start, interline*(beam_lift+beam_total_height/2))%
    --(hor_end, interline*(beam_lift+beam_total_height/2));
  for i=beams_no-1 step -1 until 1:
    pickup penrazor xscaled (interline/4) rotated 90;
    undraw (hor_start, interline*(beam_lift+coef*(i/2+(i-1)/4+1/8))--%
      (hor_end, interline*(beam_lift+coef*(i/2+(i-1)/4+1/8)));
  endfor;
enddef;

```

```

endfor;
curr_part:=curr_part-1;
enddef;

```

Esto por supuesto no está implementando la inclinación de los dinteles. Pero ahora que lo desarrollo, hay un método elegantísimo para hacerlo, usando las transformaciones de METAFONT. Asumiendo que se conocen el punto inicial y el final del dintel (P_1, P_2) , éste se puede seguir trazando horizontalmente, poniendo antes la transformación adecuada. Ésta consiste en elongación vertical y traslación horizontal. Sean s y h los coeficientes respectivos: $s = \frac{y_2 - y_1}{x_2 - x_1}$, y $h = x_1$, como se puede demostrar fácilmente. Ahora, en el trazado mismo de los dinteles (que se transforma según s y h), se debe restar h de la coordenada horizontal de todos los puntos. De resto, no hay que hacer nada más: la coordenada vertical se maneja de la misma manera que el `make_beam` citado atrás lo hace.²

Escrito
03/15/2003

Acabo de insertar dos líneas en el macro anterior, las que trazan las plicas. Eso no merece mayor comentario. Pero lo que sí es importante es que el macro, como está, está trazando dinteles que “se pasan” de la nota de la izquierda. El error se debe, en parte, a que cada parte del dintel se traza de derecha a izquierda (`hor_start > hor_end` siempre). Hay que revertir el trazado, para que la cosa sea más intuitiva. Eventualmente, cuando cada parte no sólo trace su dintel sino que borre los anteriores, `hor_end` será, más precisamente, `hor_middle`, que es exactamente la posición de la plica a trazar a cada momento.

Escrito
04/19/03

←←

Pero hay que pensarlo todo de nuevo. Definitivamente, para trazar cada parte del dintel se necesita saber el punto donde está la nota anterior. Por eso, `make_beam` tiene que salir de las cuentas, porque como es un grupo no puede comunicarse con el mundo exterior para decirle cuál es la nota anterior. Entonces, todo lo hace `beam`:

```

def beam(text t) =
  j:=0;
  for i=t:
    j:=j+1;
    for_beam[j]:=i;
  endfor;
  top_beam_note:=-99; bottom_beam_note:=99;
  for i=3 step 3 until j:
    if top_beam_note < for_beam[i] : top_beam_note:=for_beam[i]; fi;
    if bottom_beam_note > for_beam[i] : bottom_beam_note:=for_beam[i]; fi;
  endfor;
  top_beam_note:=top_beam_note+clef_conversion;
  bottom_beam_note:=bottom_beam_note+clef_conversion;
  if (top_beam_note-1)<=(bottom_beam_note+1):
    beam_lift:=bottom_beam_note-stem_length+vert_shift;
    coef:=1;

```

²Una prueba llevada a cabo hoy, 23 de Abril, muestra que esto es totalmente cierto. No se puede hacer todavía la versión final porque no tengo el algoritmo para calcular finalmente cuál es el ángulo de elongación.

```

else:
    beam_lift:=top_beam_note+stem_length+vert_shift;
    coef:=-1;
fi;
stem_hor_shift:=-coef*horizontal_axis/2;
hor_end:=w/2+stem_hor_shift;
prev_beams_no:=for_beam[j-1];
beam_total_height:=(3*prev_beams_no-1)/4;
pickup penrazor xscaled(line_thick);
draw (hor_end, interline*(beam_lift))--%
    (hor_end, interline*(for_beam[j]+clef_conversion+vert_shift));
for i=j-3 step -3 until 1:
    hor_start:=w/2+stem_hor_shift;
    for k=index-1 step -1 until for_beam[i-2]:
        hor_start:=hor_start-stretched_width[k];
    endfor;
    beam_total_height:=(3*prev_beams_no-1)/4;
    pickup penrazor xscaled(line_thick);
    draw (hor_start, interline*(beam_lift))--%
        (hor_start, interline*(for_beam[i]+clef_conversion+vert_shift));
    pickup penrazor xscaled (beam_total_height*interline) rotated 90;
    draw (hor_start, interline*(beam_lift+beam_total_height/2))%
        --(hor_end, interline*(beam_lift+beam_total_height/2));
    for k=prev_beams_no-1 step -1 until 1:
        pickup penrazor xscaled (interline/4) rotated 90;
        undraw (hor_start, interline*(beam_lift+coef*(k/2+(k-1)/4+1/8)))--%
            (hor_end, interline*(beam_lift+coef*(k/2+(k-1)/4+1/8)));
    endfor;
    hor_end:=hor_start; prev_beams_no:=for_beam[i-1];
endfor;
enddef;

```

De nuevo, esto está incompleto en varios sentidos: las plicas todavía están mal puestas; no hay inclinación de dinteles; y no se borran partes trazadas anteriormente. Pero el programa produce, así como está, lo siguiente³: ⇐

Nótese que la fuente se puede escalar como sea y el espaciado apropiado no se pierde.

7 Por el compás completo

Estoy haciendo un nuevo documento T_EX, que va a contener las versiones finales de los macros (como si fueran cargados por medio de `\usepackage`) y el código para el compás de Bach. El presente documento sigue siendo la bitácora general.

³Gasté mucho tiempo tratando de descubrir por qué al principio aparecían unos huecos extraños entre las notas. Se trata del elemento vacío que hay que poner al final del último carácter, para que éste también se elongue.

En `inv1.tex`, el preámbulo es lo que será el paquete `texmuse`. Después, el usuario escribirá:

```
\newinstrument{righthand}
\newinstrument{lefthand}
\righthand{\rangefrom{G4}|3DGAB|CABG|\rangefrom{D5}|4DGFG|\bar}
\lefthand{\rangefrom{G3}|4GG-|5r3r|GAB|CABG|\bar}
\begin{center}
\musicbox{righthand,lefthand}
\end{center}
```

¿Qué tal? Es `\musicbox` el que desencadena la escritura de archivos (porque ésta depende de los instrumentos incluidos). Pero `\newinstrument` tiene que definir los instrumentos de todos modos. Por todo esto es que no se puede hacer que cada instrumento sea un entorno como imaginé antes, sino que tiene que ser un registro: sólo cuando se ejecuta `\musicbox` es que puede empezar todo el mecanismo.

Bueno: el registro `\righthand` es aquel al que el usuario tiene acceso (cuando él dice `\righthand{\rangefrom...}` está definiendo el contenido del registro). Dentro de `\musicbox` se abren los archivos, y se leen esos registros.

En la cuantización falta la implementación de `\bar`, y también volver a escribir las notas de instrumentos anteriores cuando ya se han acabado las del presente.

Ya arreglé el último punto. Queda por hacer la implementación de `\bar`, lo que incluye decidir su naturaleza final.

Escrito
Abr. 20, 2003

Por otro lado, acabo de terminar un nuevo paso de la instrucción `\musicbox`: ya escribe los archivos de la matriz *y* el contenido de los diferentes instrumentos. Como dice al principio de la sección 5, “ya se puede suponer que `TEX` produjo archivos `.tm*` adecuados”. Ya es un hecho. ¿Y ahora? Sigue la lectura de estos archivos, gobernados por la matriz.

Dos cosas para empezar:

1. Como se puede tener un número máximo de archivos abiertos al mismo tiempo (16), si los instrumentos son demasiados se debe abrir y cerrar cada archivo, cada vez que se vaya a leer. Esto implica que `TEXmuse` debe tener un mecanismo para decidir si lo hace así o no. Desgraciadamente, esto no se puede implementar en `\newinstrument`, porque de pronto el usuario exige menos instrumentos, o uno mismo varias veces. Por ahora, vamos a suponer que siempre hay menos de 16 instrumentos (¿ó 15, o menos?) \Leftarrow
2. La decisión del `\sfcode` de cada caracter se basa en los datos de *dos* líneas de la matriz consecutivas. La mejor manera de hacer esto es que cada vez que se lee una línea se decide el `\sfcode` del caracter *anterior* (se ha guardado en memoria el cuántum de éste).

Pues me ha ido muy, muy bien. Lo siguiente es lo que el programa, como está hoy en `inv1.tex`, escribió en el archivo `inv1.mf` (para el primer caracter):

Escrito
Abr. 21,2003

```

beginchar(1,line_width,line_height,line_depth);
find_pos(1);
instrument(1);
add_notes(g3);
staff;
instrument(2);
add_notes(d5);
staff;
endchar;

```

Es exactamente lo que se pedía unas páginas atrás. Claro, todavía no calcula dinteles, ni `\sfcodes`. Todavía no va al siguiente caracter. Ya hasta se puede poner `\makeatother` antes de empezar, aunque esto significó que hay que encerrar algunas definiciones entre `\makeatletter` y `\makeatother`⁴.

En los últimos minutos hice que se lean todos los caracteres de un mismo bloque. Cómo pasar al siguiente bloque queda todavía abierto, y depende de la solución al problema de `\bar`. Por ahora me concentro, entonces, en completar el programa para un compás.

Por supuesto, ahora noto un error. Yo hice que el programa escribiera una línea en METAFONT con `find_pos(n)`, donde `n` es el número del caracter. Pero `n` debería ser el número para el `\sfcode`. Es relativamente grave, porque, como he dicho, esto se tiene que escribir cuando ya se sabe el siguiente cuántum. Como está ahora, el programa va escribiendo las líneas inmediatamente, o sea que el cambio debe ser bastante sustancial. Eso es lo siguiente que hay que hacer...

Pues aquí estoy, decidido a solucionar este problema. A propósito, se me ocurrió recientemente que se puede usar para el efecto el archivo `.tm*` que contiene la versión inútil de la matriz, si es que decido hacerlo con archivos. Pero primero voy a intentar los registros, por supuesto tratando de reciclar `\to@write`. Pero resulta que éste no es un registro, sino un comando... Por eso hice un duplicado del archivo, y ahora me dispongo a modificarlo todo para hacer de `\to@write` un registro.

Escrito
22 Abr., 2003

No: esto último no se puede, porque los otros comandos no se expanden (*¿?*). Toca, simplemente, re-definirlo como registro.

Sigo escribiendo hoy, Abril 22. Esto es hermoso. Voy a dejar `inv1.tex` como está, porque es un documento lindo. La fuente la hizo el programa por sí solo (todavía no hay plicas ni dinteles, y no calcula las distancias verticales). Por ahora hay que concentrarse en los dinteles, las plicas, y el mecanismo para el cambio de compás.

Una cosa para notar: los ‘tres pasos’ fueron una ilusión, pues el programa que está en `inv1.tex` produce el compás de la invención en la primera compilación. Sin embargo, esto no es del todo bueno: cuando hay varios comandos `\musicbox`

Escrito
Abr. 23, 2003

⁴En concreto, las instrucciones que leen los archivos deben ejecutarse en un contexto en el que la arroba es una letra.

(por ejemplo), la fuente se va a escribir y cargar muchas veces, y no parece aconsejable que esto ocurra sin algún orden. Esto es algo a tener en cuenta.

Pero el cambio más sustancial que me propongo para hoy es la automatización del espacio vertical. Intento, entre otras cosas, que sea METAFONT, y no T_EX, el que calcula las distancias verticales, lo cual implica que lo que T_EX le pasa a METAFONT es un macro METAFONT, que guarda la información de los caracteres hasta que llega el final de la línea (o de la caja). He aquí como funciona:

1. Cada vez que METAFONT recibe un caracter de T_EX, hace una variable `picture` que contiene el dibujo de un instrumento en un caracter. `(0,0)` será el punto que interseca el eje de notas y la segunda línea del pentagrama.
2. Además, METAFONT va modificando la variable `highest_note[]` para cada instrumento.
3. Cuando recibe la instrucción `make_line`, METAFONT simplemente hace los cálculos de altura y profundidad, y añade las variables `picture` a sendos caracteres entre `beginchar` y `endchar`, y `shifted` de acuerdo a la necesidad.

Esto significa muchas cosas: cambiar toda la orientación de los macros ya existentes, para que no usen la variable `w`. Por eso, voy a dejar `tmstst.mf` como está, y empezar a trabajar en `tmsnew.mf`. Por otro lado, en el proceso decidí cambiar el mecanismo de la traslación vertical de los pentagramas: ahora se hace cuando se añaden los dibujos, por medio de `shifted`.

T_EX tiene que darle a METAFONT, al principio, el número de instrumentos, para que se puedan definir las variables `highest_note` y `lowest_note`.

Nota: estoy usando de nuevo `firstpassartificial.mf`. La versión original (que produjo las notas unas páginas atrás) se puede reconstruir desde el texto y `infl.mf`.

Para ahorrar memoria en METAFONT, las variables `picture` se sobrescriben cuando se empieza una nueva línea. T_EX le da a METAFONT la orden `make_line`, cuyo argumento es el número de caracter del primer caracter de la línea.

Esto fue relativamente fácil de implementar. Ahora hay que hacer los cambios correspondientes en T_EX, en el archivo nuevo `inven.tex`.

Ya logré eso, y poner los dinteles. Pero entonces me di cuenta de que hay un problema con las figuras rítmicas, es un desorden como se van poniendo `\q@n` y las demás. Siguiendo paso: racionalizar `\everynote` y todo ese mecanismo.

7.1 Everynote

Todas las notas van a dar el dato de su valor rítmico, así que va a haber un comando `\@value` que vale `\q@n`, `\e@n`, o las demás.

Además, todas van a tener información sobre la notación gráfica de ese ritmo: plica, dintel, o silencio. Eso será la variable `\rhythm@not`.

Eventualmente, se añadirán variables para las articulaciones.

Escrito
Abr. 24, 2003

7.2 Se me pasó...

Pues me acabo de dar cuenta, gracias a que algo estaba saliendo mal, que estoy aplicando el algoritmo de elongación redundantemente. Si ya lo implementé en METAFONT, ¿para qué poner a T_EX a hacer lo mismo, con todos los riesgos que implican las conversiones de medidas, etc.? No hay ninguna razón por la cual los caracteres no tengan la longitud *ya elongada*, y eso evita toda clase de malentendidos entre los dos programas, el uso de `\makebox`, el cambio de `\sfcodes`, etc.

7.3 El compás...

Bueno, queda una cosa por hacer en el compás de la invención: implementar el -. Los silencios faltan, pero simplemente es dibujar el caracter (completar la fuente, que será dentro de mucho...). Lo del - (y el +) resultó bastante complicado. Hice uso de los comandos L^AT_EX `\@ifnextchar`, `\@firstoftwo`, y el resto, son buenísimos pero hacen la vaina muy abstracta. En todo caso, no vale la pena implementar el uso de varios signos de estos sucesivos. El usuario tiene en todo momento acceso a 3 octavas, eso es suficiente.

Probé también definir comandos como el usuario lo haría. Los comandos hay que definirlos con los `\catcodes` en efecto; pero esto es dentro de un grupo, y entonces cuando el programa llega a leerlos efectivamente, ya no están definidos. Hay que hacer la cosa por medio de registros, y el usuario tendrá un `\muscommand`. Esto, de todos modos, hay que pensarlo bien, porque hay que ver si se puede permitir transposición.

Entonces, en efecto, ya es posible hacer el segundo compás de la invención de Bach exactamente como se hará cuando el programa esté terminado.

Para los silencios, se trata simplemente de insertarlos en el caracter. Aquí sí se permite una serie de menos o masas, que mueven los silencios hacia arriba o hacia abajo tantos `interline` como signos haya.

8 Generalización

8.1 Plicas

Las plicas resultan ser un procedimiento complicado. He aquí como he decidido hacerlo: una plica normal se hace en tres procedimientos: `find_extremes` “ordena” las notas: le da a `aggr_note[1]` el valor de la nota superior y a `aggr_note[-1]` el de la inferior (ya con conversión de clave en ambos casos). Además, decide si `stem_coef` es 1 ó -1. Esta función es tan genérica que la estoy incluyendo dentro de `add_notes`.

Después, `make_stem`, cuyo argumento es 1 (para plicas hacia arriba) ó -1 encuentra el punto de partida y de llegada de la plica, manejando correctamente los valores `aggr_note`. En plicas automáticas, el argumento de esta función es `stem_coef`, que `find_extremes` ha decidido.

Escrito
Abril 26, 2003

Hasta aquí, sin embargo, no es suficiente: las plicas ‘normales’ deben llegar o pasar la tercera línea del pentagrama. Además, el programa tiene que hacer disponible el punto extremo de la plica, para añadir las banderas, etc. Y, por otro lado, ¿cómo podremos variar la longitud de la plica?

La solución a todo esto es hermosa: el signo del argumento sigue indicando la dirección, pero su valor indica la longitud de la plica. El tercer paso se cumple dentro de `regular_note`, que le da a `make_stem` el argumento `stem_coef*stem_length` y modifica después si es necesario.

8.2 Espacio

Hay que definir lo del espacio. Ahora trabajando para las plicas estaba haciendo experimentos con negras, y sí es cierto que quedaban muy cerca una de la otra, la diferencia con las corcheas era imperceptible. ¿Cómo definir el factor por el que se multiplica la longitud natural?

Bueno, hay que notar lo siguiente: de las semicorcheas para abajo se puede decir que el espacio después de la nota debe ser alrededor de $\frac{3}{4}$ la anchura de la nota misma (de la cabecita). Para las corcheas se podría decir que es la anchura misma. Para las negras, en cambio, uno diría que es por lo menos 4 veces el ancho de la nota. Y las blancas, por ahí unas 7 veces. Las redondas ni se diga, por lo menos 12.

Esa es la idea: `TeX` va calculando, de acuerdo a las notas que encuentra (en la matriz), la longitud ideal. Por ejemplo, una línea con 16 semicorcheas (como el compás de la invención de Bach): he venido trabajando con 1.6; para lograr esta cifra se diría que la semicorchea añade 9.6 anchos, a $.6 (\frac{3}{5})$ cada nota (fracciones de un `regular_width`, que es el ancho de la cabeza y dos `framingspaces`). Siguiendo, por ejemplo, la sección áurea (aunque los resultados parecen muy tímidos), tenemos lo siguiente:

Nota	Cuanta	Elongación
Redonda	256	4.237
Blanca	128	2.618
Negra	64	1.618
Corchea	32	1
Semicorchea	16	.618

Hay que notar que estas no son las proporciones de separación entre notas (que se van doblando de nota en nota), sino sólo la longitud a añadir a la línea. Voy a implementarlo ahora mismo.

Y lo hice, pero qué lío. Hay un peligro constante de que se desborde la capacidad aritmética de `METAFONT`...

8.3 Líneas adicionales

Otra cosa que hay que hacer en las notas normales es encontrar y dibujar las líneas adicionales. Esto va a ser parte de `add_notes`. Se puede complicar para permitir que no se tracen líneas adicionales.

8.4 Silencios

Para los silencios se usa la misma variable de los dinteles: `\beam@no`. En la fuente hay una matriz de `pictures`, llamada `rest`, cuyos subíndices corresponden al `\beam@no` de cada figura. Queda por hacer la implementación de `-` y `+` para silencios.

Que hoy acometí y solucioné.

Escrito
Abr. 27, 2003

9 Se acaba el compás. . .

Bueno, ya no puedo darle más vueltas al problema del cambio de compás. En principio, lo único que sé es que con la instrucción `\bar` se cambia de “bloque”, es decir, el siguiente cuántum es 0. Eso trae una cantidad de problemas. Por el momento, voy a asumir que el compás no está vacío en ningún instrumento.

Los problemas son inesperados. Lo que esperaba que fuera muy difícil en realidad no lo fue tanto, y en cambio ahora me doy cuenta de que esto abre todo un nuevo universo, porque involucra caracteres cuyo ancho se escapa de lo normal (la barra).

He aquí el nuevo plan: \TeX escribe la instrucción `new_char` con un argumento que es exactamente la línea de la matriz, es decir una serie de números, separados por comas, que identifican el tipo de caracter en cada instrumento. `METAFONT` tiene un arreglo llamado `widths[]`, en donde se guarda la información del ancho de cada caracter, y cuando lee `new_char` decide cuál es el máximo.

Esa es la idea básica, pero en realidad queda por implementar la decisión de si los elementos que “se salen” del caracter van a ser tenidos en cuenta o no. Eso debe hacerse en \TeX , porque sólo él puede leer la matriz.

Bueno, ya estoy produciendo dos compases, pero quedan varias cosas por hacer: en primer lugar, automatizar el bucle, que por el momento es manual. Pero, más importante, parece que hay que automatizar las barras por completo, al menos por defecto.

9.1 Reflexión

A ver, esto es lo que pasó ayer con las barras de compás: primero que todo me di cuenta de que la barra que atraviesa varios pentagramas no puede ser trazada sino hasta que la línea completa se traza, porque se necesita saber el alto de cada pentagrama, etc. Esto significa que, dentro de `make_line` debe haber una rutina para el trazado de esas barras. De todos modos, la barra individual sería trazada por el instrumento mismo (así se permite que haya barras propias para un instrumento).

Entonces, haciendo los primeros experimentos, sucedía algo extrañísimo: aparecía la barra en el pentagrama inferior, pero el superior hacía locuras, literalmente. Después de mucho, mucho perseguir el problema, caí en cuenta de que, por error, había puesto `\bar` en lugares diferentes de cada voz. Entonces el programa llegaba a la primera que había (que era la del pentagrama inferior,

Escrito
Abr. 28, 2003

en este caso), y sencillamente ignoraba el resto del compás que quedara en los demás pentagramas.

Eso no es un problema de naturaleza: la definición del ‘bloque’ (y el compás es un caso especial de ‘bloque’) es justamente que todos los instrumentos lo comparten. Pero el riesgo de error por parte del usuario es inmenso. Por eso digo que habría que automatizar las barras de compás para música ‘normal’. Por otro lado, la barra siempre produce espacio a su derecha, lo cual es un problema si es la barra final de la línea. Mejor dicho, hay que rediseñarlo todo.

Pensando cómo sería la cosa si los dos pentagramas tuvieran barras independientes. En términos de cuanta, las barras no deben añadir nada. La barra y la nota que le siguen están ambas en el mismo cuántum. Eso quiere decir que la barra es parte de la nota que le sigue. Además, siempre añade espacio, es decir, el espacio entre ella y la nota que le sigue siempre es tenido en cuenta. El usuario puede cambiar este espacio, que por defecto será igual a una cabecita.

Pero el problema no acaba ahí: eso funciona perfectamente sólo si la barra no es la barra final de la línea. Si es la barra final, se debe divorciar de la nota. Entonces, podríamos pensar que el comando `\bar` modifica un condicional: si está vigente durante la nota siguiente, o bien a ésta se le añade una barra o bien `make_line` la hace. En ambos casos, el condicional se vuelve falso de nuevo.

Esto, por otro lado, obliga ya a pensar cómo va a ser el desplazamiento de las notas cuando su ancho no es el normal. METAFONT va a tener una variable para cada carácter (llamada `note_axis`). Generalmente es 0, pero cuando hay barra (por ejemplo), va a ser positiva, con un valor adecuado a la distancia a desplazar. `make_beam` es el que ejecuta esta variable.

Voy a implementar eso por el momento.

Siguen los problemas. Pero por lo menos los identifiqué un poco más: esta vez el problema es con los cuanta. Si pongo la barra en 256, y la nota siguiente en 0, queda un carácter espurio. Si pongo la barra y la nota en 0, con nada en 256, el último instrumento puede borrar muchas de las líneas de los anteriores en la matriz. Lo mismo si pongo ambas cosas en 256.

Entonces vamos a separar. Bloques son una cosa, barras son otra. La instrucción `\barline`, eventualmente, producirá una barra independiente en el propio pentagrama. Los bloques funcionan de otra manera: hay un valor `\block@period` que contiene el número de cuanta de un compás (256 para un cuarto). En la cuantización, si el cuántum actual sobrepasa ese valor, se pondrá una línea 9998, y al siguiente cuántum se le restará el período. Si hay que evadir el mecanismo, `\block@period` se hace igual a 9999 (lo cual quiere decir que no puede haber compases mayores que 156 cuartos).

Entonces, la instrucción `\meter` ajusta el valor de `\block@period`. También `\nometer`, que lo pone en 9999. Además, hay un `\block@separator`, que por defecto contiene a `\barline`.

Esto fue un gran lío, lo de automatizar la barra. ¡Y sólo he hecho la cuantización! Me quedó horrible, pura fuerza bruta. Pero bueno... Ahora sigue la confección de los archivos auxiliares. Aquí también se debe llevar la cuenta de los cuanta, porque hay que decidir dónde terminan los bloques.

Esto también fue muy complicado, ¡eh! Pero bueno, al fin lo logré. Ahora

Escrito
Abr. 29, 2003

viene la traducción a METAFONT, y tengo que leer lo que escribí esta mañana, ¡para acordarme del plan!

Bueno, finalmente logré que T_EX escribiera un archivo METAFONT adecuado. En realidad, falta poner el argumento de `barline`, pero es que hay que pensar del todo el mecanismo de las cosas que ‘se salen’ de las notas. Lo que puedo pensar por ahora es que todo se puede reducir a un solo caso, bien todo se sale hacia la izquierda, o bien todo hacia la derecha. Mañana veré. . .

Creo que lo tengo. Todos los ‘elementos exógenos’ se consideran hacia la izquierda. Cada vez que se construye la nota $n - 1$ de un instrumento, se hace *cierto* condicional (cuando no hay nota, este condicional sigue siendo *falso*). Entonces, si la nota n tiene un elemento hacia la izquierda, éste entrará en el ancho del carácter si y sólo si el condicional es cierto. Si la nota $n - 1$ tuvo un elemento hacia la derecha, su ancho se ha guardado en la memoria, y se añade a n si allí hay nota.

Ahora, eso quiere decir que el ancho del carácter no se va a poder saber de antemano (a menos que se le diera un número de identificación a todas las posibles combinaciones de elementos exógenos). Eso no tiene problema para el trazado de las notas, que no usan el ancho, pero sí para el trazado de los pentagramas y de los dinteles (y demás). Eso, según parece, hay que posponerlo para el final del carácter.

Entonces, he aquí como funciona la construcción de un carácter:

1. Se trazan todas las notas y cosas que no dependen del ancho del carácter.
2. Se añaden las cosas que sí dependen del ancho (`staff`, `beam`, etc.) a una variable de texto, `to_do`[], asociada a cada instrumento.
3. Se repite lo mismo para el resto de los instrumentos.
4. Conforme se hace lo anterior, se deciden el eje de las notas (`note_axis`) y el ancho que habrá que añadir a la izquierda del carácter.
5. Se ejecutan los `to_do`'s mediante `scantokens`. Esto requiere repasar todas las variables `picture` de los instrumentos, pero bueno, toca así.

Ayer al fin logré una función METAFONT que encuentra el ancho del dibujo dado. Es increíble que no haya un `widthof` que haga eso inmediatamente. Para lograrlo finalmente tuve que renunciar a la infabilidad. El procedimiento es ir borrando líneas desde muy lejos, e ir acercándolas (búsqueda binaria, todo es una variación de `solve`) hasta que en algún momento se cruzan con el dibujo. Pero esto no es infalible: la vertical de las líneas no puede ser `-infinity-infinity`, y la búsqueda no se puede hacer desde `infinity`, porque toma demasiado tiempo, lo que sería absurdo. Entonces hay que implementar unas variables internas, que el usuario puede cambiar: horizontalmente, la línea se empieza a borrar desde `5regular_width`, y tienen una altura de `20interline` (10 para cada lado).

Ahora bien, vamos a ver qué hacemos con esta función. Lo que hay que cuidar es que el extremo izquierdo de la nota actual no se cruce con el derecho

Escrito
Abr. 30, 2003

Escrito
Mayo 2, 2003

de la pasada. Eso quiere decir que el espacio en blanco entre el extremo derecho de la anterior y el izquierdo de la actual debe ser menor que ese espacio. Ese espacio, además, es el ancho ya elongado menos `note_axis` menos el extremo derecho de la nota pasada. Entonces:

$$\text{stw}_{(i-1)} - \text{nax}_{(i-1)} - \text{rtx}_{(i-1)} - (\text{ltx}_i - \text{nax}_i) \geq \text{fs}$$

(debe haber por lo menos un `framingspace` entre las notas). ¿Qué pasa si no, si el espacio es menor? Ahí METAFONT entra en modo ‘cuidadoso’, que quiere decir que busca (por medio de otra variación de `solve`) cuál debe ser el desplazamiento horizontal de la nota actual para que no se cruce con la anterior. A ese le añade un `framingspace`. El `note_axis` de la nota actual será entonces ese desplazamiento menos el ancho de la cabecita sola.

Cuando las dos notas no son consecutivas en la partitura completa (es decir, sus índices no son consecutivos), se debe añadir el espacio ya elongado de todas las notas intermedias, lo cual hace menos probable que haya que mover la nota actual.

Entonces, ¿cuándo se debe ejecutar el cálculo de los extremos? Hay razones para tratar de evitarlo siempre que se pueda. Por ejemplo, si la nota consta sólo de una nota, no hay por qué calcular. Por definición, una plica no añade espacio horizontal, o sea que no hay que calcular. Entonces voy a implementar un condicional, activado por las cosas que potencialmente añaden espacio horizontal (como alteraciones, segundas, barras, claves, etc.)

La comparación misma, y el consiguiente hallazgo de `note_axis`, se llevan a cabo en `add_to_dos`, que incidentalmente podría ser parte de `new_char`, y que ahora voy a llamar `finalize_char`.

El valor de los extremos no tiene que guardarse en memoria: sólo se necesita el de la última nota del mismo instrumento. Es un arreglo que sólo tiene un valor por instrumento.

Bueno, eso parece que está funcionando como debe. Entonces voy a implementar el sostenido, que es un caso más sencillo que la barra.

Acabo de lograr que METAFONT ubique bien el sostenido, y alargue la nota si es necesario. Todavía no está implementado en $\text{T}_{\text{E}}\text{X}_{\text{M}}\text{S}_{\text{E}}$. Quiero primero terminar de implementar la barra de compás.

Por otro lado, ahora tengo que dedicarme a la reorquestación de mi *Ex Tenebris*, por lo que voy a dejar este proyecto por algunos días (después de eso tengo otras muchas cosas que hacer). Para cuando retome, las siguientes cosas deben hacerse pronto: las plicas de los dinteles todavía no están ajustadas a la cabecita; las plicas normales todavía no se tienen en cuenta para el cálculo de las alturas; las líneas del pentagrama tampoco (claro, por el momento estoy trabajando con 3 y -1, que son los valores del pentagrama estándar).

Claro que, volví (después de haber trabajado en otras cosas), y descubrí que hay un requerimiento nada cómodo para todo el mecanismo del espaciado de elementos exógenos. Los dibujos tienen que ser continuos en la horizontal,

para que el hallazgo de los extremos sea posible. Eso implica pintar al menos una línea del pentagrama. ¿Qué pasa cuando no hay pentagrama? Buena pregunta.

El hecho es que logré hacer el espaciado correcto de la barra. Pero, como se movieron todas las notas, los dinteles quedaron mal. Hay que hacer que los dinteles se calculen con `note_axis`, no el viejo `stem_axis`.

También descubro un problema de concepción en el modo ‘cuidadoso’: METAFONT está tratando de hallar el punto donde los dibujos no se cruzan. Pero está usando el dibujo de la nota anterior, *¡con pentagrama y todo!* Es decir: siempre se van a cruzar. Hay dos opciones: o posponer el trazado del pentagrama, o simplemente forzar el nuevo `note_axis` a partir del `avlbl_space`. La primera es la más profesional.

Lo más sensato es que cada caracter termina de trazar el caracter anterior, añadiéndole el pentagrama y los dinteles, *después* de haber decidido la posición propia. El pentagrama sale del `add_to_dos` de cada instrumento. Más bien, hay una orden `prev_staff` que lo traza, y cuyos argumentos son el índice del caracter (el anterior, usualmente), y las líneas que hay que trazar. Cuando el usuario cambia el pentagrama, esta última parte será redefinida en METAFONT.

Bueno, pues ya logré que separe cualquier cosa por dos `framingspace`, de ser necesario. Claro, las barras deben estar separadas más que eso. La respuesta al problema de la continuidad de los caracteres se resuelve con un control detallado de `cullit`. Se trazan las líneas del pentagrama con un ‘peso’ especial, y después se hace un `cull` adecuado para quietarlas. Eso habrá que hacerlo como parte de la implementación de diferentes tipos de pentagrama.

Por ahora, voy a implementar el sostenido, para ver si el espaciado está funcionando bien.

Y ya lo logré. Pero surgen varias cosas: primero, el sostenido mismo está horrible. Segundo, le vamos a dar un número a cada elemento musical. Parte de la tarea de la fuente es darle al programa METAFONT la información sobre el ancho de cada elemento, así el programa puede trazar el pentagrama (para la continuidad) con la distancia adecuada.

Otras cosas: hay que hacer que `make_line` trace el pentagrama anterior. Y hay que corregir el trazado de las plicas en los dinteles.

Bueno, acabo de arreglar los dinteles. Y ya puse el pentagrama del último caracter en `make_line`.

Escrito
Mayo 5, 2003

Escrito
Mayo 6, 2003

9.2 Vuelta a la barra

Entonces, la barra tiene que hacerse de otro modo. Va a ser un caracter en sí misma, con pentagrama hacia la izquierda, pero no hacia la derecha. Pero el trazado no modifica los `switches barline_next`, porque esos se usan, en la nota siguiente, para añadir más espacio. La barra no se traza inmediatamente, porque se necesita saber el alto. Pero se avanza el `index`, y si acaso se dibujan las claves y las métricas.

10 Aquí de nuevo

Más de tres meses después vuelvo. Había problemas con el espaciamiento de notas extrañas. Ya descubrí que eso se debía a que variables como `note_axis` y `extremes_to_find` se inicializaban en `add_notes`, o sea que una nota del pentagrama superior arruinaba lo que el inferior había decidido. Ahora se inicializan en `new_char`. Es más, en general `note_axis` no puede decrecer: si un instrumento lo fijó en un sitio, otro instrumento no lo puede hacer retroceder.

Escrito
Agosto 17, 2003

No sé en qué momento se dañó el mecanismo de las barras. Me toca repasar eso una vez más (y eso incluye entender cómo es que se hace). Por otro lado, todas las notas están buscando el extremo derecho, con la famosa variación de `solve`. Eso se podría evitar para la mayoría de notas, por lo menos las que son normales.

Ésto último ya lo hice, pero entonces me doy cuenta de que `ltx` es $1/2\text{regular_width}$, mientras que `rtx` es $1/2\text{horizontal_axis}$. Creo que el primero se puede modificar, porque el `framingspace` está incluido en el cálculo de `avlbl_space`. Efectivamente.

Bueno, entonces por la barra. Lo de trazar las barras como un caracter independiente (sección anterior) tiene un montón de problemas, porque afecta el número total de caracteres de la línea, el “space factor” total, etc. Más bien volvemos al esquema anterior: la barra es parte del caracter que le sigue. Lo único es que eso implica que debe haber siempre una nota después de la barra, en todos los instrumentos. Probablemente habrá que hacer que \TeX inserte una nota vacía si es necesario.

Problemas. Cada pentagrama se entera de que la próxima figura tiene barra en momentos distintos. Por ejemplo, la mano derecha se enteró para la invención en la última corchea, mientras la izquierda se enteró en la última *semicorchea*. Entonces, la mano derecha empezó a calcular siempre un `bar_axis`. La idea es que la posición de la barra se ve afectada sólo por aquellos pentagramas que tienen barra. En este caso, la derecha contaba, la izquierda no. Ese `bar_axis`, supuestamente, sólo tiene efecto cuando el caracter en cada voz tiene un peso mayor que 0, momento en el que se traza la barra realmente. Pero el problema es que cuando hubo que trazar la barra en la derecha, se tomó el valor que se había calculado antes, y que no se volvió a calcular... Mejor dicho.

Escrito
Agosto 18, 2003

10.1 ‘Insertos’

Entonces, nuevo enfoque. Supongamos que \TeX le pasa a METAFONT una lista de barras para cada instrumento, al principio del archivo (y esto significa volver a programar en \TeX por un ratito, que ya lo estaba extrañando). ¿Habrá momento en que \TeX ya sepa cuántas y cuáles barras habrá en la línea? En todo caso, las barras se convierten así en otro tipo de caracter especial: los ‘insertos’. Otros insertos son texto o cualquier caja \TeX que el usuario quiera insertar dentro del pentagrama. Habrá unos que son insertos a la derecha de un caracter (la barra, por ejemplo), otros a la izquierda (esto depende de por dónde se puede partir la línea). Seguramente se van a tener que definir dos clases de

barras, porque la final no añade espacio a la derecha. Pero por el momento voy a implementar todo esto sin espacio a la derecha de la barra (cada nota saldrá un `framingspace` después de la barra).

En principio, debe ser posible que T_EX ya sepa de las barras antes de empezar a escribir el archivo. La instrucción `barline` es el resultado de `\@bar`, que a su vez es el de `\end@block` (para música normal). Ahora, en vez de expandirse como `\@bar` en el archivo auxiliar del instrumento, esto añade el número del carácter a un registro especial, separado por comas (porque es una variable de texto de METAFONT). El registro es creado por `\newinstrument`, y se llama `#1@bars`.

Y, sin embargo, hay un problema, muy tonto: T_EX va llevando el cálculo del número de carácter sólo cuando está escribiendo el archivo METAFONT. Es decir, en la cuantización (que sería el momento de averiguar dónde hay barras) no sabe la nota en que va. También hay otros problemas: la barra no puede ser un agregado a la derecha, porque de pronto no hay nota a la cual agregar, etc.

10.2 Factores primos

Pero se me ocurrió otro camino, mucho más elegante: el argumento de `new_char`, que hasta el momento no se ha usado (1 es nota, 0 es nada). Ahí podemos dar información a METAFONT sobre características generales del carácter en el pentagrama. Sería por medio de los factores primos del número que se le da. Si el número es múltiplo de 2, significa que *hay* nota; si es múltiplo de 3, hay barra. También se pueden usar el 5, el 7 y el 11 (porque 13 llevaría consigo el problema de que si lo hay todo el número es 30030, más que lo que METAFONT puede manejar; pero si se define que no puede haber 11 y 13 al mismo tiempo no hay problema, porque da 2730).

Elaboremos: METAFONT tiene así una manera de saber si tiene que preparar (¿y pintar?) la barra para cada nota. Como sería la nota misma que tiene la barra a su izquierda (incluso si no hay nota en sí misma), todo el problema de cómo hacer que se ejecute realmente la barra está solucionado (el problema era que el dato se daba en la nota *anterior* a la barra, de la que no sabemos cuán lejos está).

Voy entonces a implementar el cálculo del número que debe hacer T_EX. Debe ser trivial, una modificación en la cuantización. Y sí, más o menos lo fue. ‘1’ ahora significa ‘nada’ (de manera que puede haber una nada con barra). El 0 no se usa más que para el 9999 final.

Bueno, ya está implementado lo del factor primo 3 para la barra. La barra sigue saliendo mal, porque se cruza con el sostenido. Eso quiere decir que el cálculo de `bar_axis` está viciado.

Finalmente pude. El problema era que estaba contando `note_axis` dentro de la traslación de la barra, y no se tenía que contar (porque en todo momento antes de realmente dibujar el carácter, 0 es el eje de la nota). Pero después también había el siguiente problema: la barra se debe trazar después de encontrar los extremos izquierdos de todos los caracteres; pero al trazarla el extremo

Escrito
Agosto 22, 2003

se mueve con ella. Entonces hay que insertar el trazado de la barra dentro de `finalize_char` mismo, antes de calcular el espacio disponible.

Por otro lado, en este momento la barra se hace con el simple `draw`. Pero se deberán definir variables `picture` que contengan el tipo de barra de cada instrumento.

Sigue habiendo una cosa mal: cuando la barra no es para todos, debería simplemente añadirse a la izquierda de aquel(los) caracteres que la tiene(n). Pero en este momento la barra en el pentagrama superior sigue influyendo sobre el espaciado de la nota en el superior. Tendré que mirar eso en detalle.

¡Al fin! Después explico cuáles fueron los múltiples problemas muy sutiles.

A saber: las rutinas que encuentran el ancho del caracter (`find_ltx` y `find_rtx`) requieren un manejo muy cuidadoso del ‘peso’ del caracter (con `cull`). Parece que ya está solucionado, y entonces ya tenemos un prototipo para las barras, incluso aquellas que no están coordinadas.

Paso ahora al tercer compás, debería ser posible hacerlo inmediatamente. En efecto, sale bien. Queda por hacer:

- Borrar las líneas auxiliares de los elementos exógenos.
- Implementar el espacio *antes* de la barra.
- Dibujar la barra final de línea y los conectores de barras.
- Inclinar los dinteles.
- Pintar el silencio de semicorchea.

Me voy a dedicar al espacio ahora. Voy a hacer un nuevo instrumento para hacer pruebas.

Una pequeña modificación: hasta el momento, TEX calculaba el número de cuanta de una nota a la siguiente; lo dividía por 16; y lo multiplicaba por 10. Era la única forma de hacer que pudiera pasarle a $\text{M}\text{E}\text{T}\text{A}\text{F}\text{O}\text{N}\text{T}$ un número entero que reflejara el `\sfcode`. Pero crea la posibilidad de desborde aritmético. Entonces ahora TEX simplemente le pasará a $\text{M}\text{E}\text{T}\text{A}\text{F}\text{O}\text{N}\text{T}$ el número de cuanta. Los cálculos se llevarán a cabo en $\text{M}\text{E}\text{T}\text{A}\text{F}\text{O}\text{N}\text{T}$, en un orden 100 menor. No hay porqué dividir por 16, además. El cambio implica también manejar el cálculo total de `\to@stretch`, que no estoy seguro de entender por completo, y más bien sospecho que está mal. Esto es lo que se “explica” en 8.2. Hay muchas cosas que no están claras. Pero en todo caso, en principio, la elongación que recibe una nota de q cuanta es

$$\phi(q) = 1.618^{\log_2(q/16)-1},$$

siempre y cuando $q \geq 16$ (porque figuras menores que la semicorchea funcionan igual a ésta).

Lo que alcanzo a entender es que esto da la longitud “ideal”. No estoy muy seguro si esta longitud ideal es relevante o simplemente un resultado de los algoritmos. Por ejemplo: ¿cuál es su relación con la longitud “deseada”

Escrito
Agosto 23, 2003

Escrito
Agosto 25, 2003

para la línea? Si ésta última está definida (por las márgenes, por el argumento de `\musicparbox`, o por lo que sea), me parece, la longitud “ideal” ni siquiera será calculada. Bueno, no sé... En todo caso parece que la suma total de elongaciones ($\sum \phi(q)$) debe ser calculada por `TEX`, porque no siempre es función directa del número total de cuanta.

O mejor: lo que `TEX` calcula es la suma total de “cuanta efectivos”, es decir, el número total de cuanta, pero convirtiendo cualquier $q < 16$ en 16.

Encuentro que la implementación en este momento no incorpora la noción de longitud “deseada”. Es `\musicbox`, que simplemente dibuja la línea con su longitud “real”. Ésta última es la que se calcula con la fórmula de arriba, la que allí llamé “ideal”. El algoritmo se basa en la fórmula:

Escrito
Agosto 26, 2003

$$s_i = l_i + f_i \frac{r \sum \phi(i)}{\sum_{j=1}^z f_j},$$

donde r es el `regular_width` de un caracter normal, y s_i el `stretched_width` del caracter i . Nótese que no se incluyen los anchos de las notas especiales: una línea con tres sostenidos sería igual de ancha que una sin los tres sostenidos. Así mismo, las barras no se tienen en cuenta.

Volviendo al problema de la barra: ¿cómo reestablecer el espacio *detrás* de la barra? Si, por ejemplo, se intenta que la barra caiga justamente en posición horizontal 0 del caracter (y así todo el pentagrama a la derecha de la nota anterior estaría vacío), hay interferencia indeseada cuando las barras son individuales. Más bien, se trata de que la barra incluya en sí misma una línea hacia la izquierda de la longitud que la nota anterior se elongó.

Pero todo esto me hace darme cuenta de que he abandonado la variable `width`, que debería actualizarse cada vez que se encuentran los extremos. Lo que me temo es que ese cambio también arruine el trazado de los dinteles.

11 Por hacer

Entonces, hay varias cosas por hacer. Hoy les doy el siguiente orden:

1. Re-implementar la fórmula de espaciado en `METAFONT`.
2. Re-acomodar el espacio antes de la barra.
3. Incorporar `width` en los cálculos.
4. Hacer que los elementos exógenos borren las líneas de control.
5. Pensar en un mecanismo de dos pasos de `METAFONT`, que podría hacer el cálculo de espacio, líneas, etc. más preciso.

12 El espacio...

A ver, entendí que hasta aquí he venido confundiendo la ‘elongación’ con el

Escrito
Diciembre
15/2003

procedimiento para encontrar el espacio después de una nota. Es decir, siempre se ha calculado la longitud ‘ideal’ de cada caracter; la longitud ‘deseada’ no ha ni aparecido. No estoy seguro de cómo voy a incorporarlo al producto final, pero por ahora me voy a concentrar en en la longitud ideal (un `\musicbox`).

Entonces hay un cambio de nomenclatura:

Antes	Ahora	Descripción
<code>regular_width</code>	<code>head_width</code>	Ancho normal de una cabecita
<code>stretched_width</code>	<code>real_width</code>	Ancho de una nota, incluyendo lo que se añade a su derecha

Entonces, `finalize_char(n)` calcula