Universidad del Valle de Guatemala

Facultad de Ciencias y Humanidades

Departamento de Matemática

DESARROLLO DE UN MODELO ALGORÍTMICO PARA LA TRANSFORMACIÓN DE PARTITURAS MUSICALES ESCRITAS EN NOTACIÓN MENSURAL BLANCA A NOTACIÓN CONTEMPORÁNEA

TRABAJO DE GRADUACIÓN PRESENTADO POR

Martha Eladia María Thomae Elías

PARA OPTAR AL GRADO ACADÉMICO DE

Licenciada en Matemática

GUATEMALA

2013

DESARROLLO DE UN MODELO ALGORÍTMICO PARA LA TRANSFORMACIÓN DE PARTITURAS MUSICALES ESCRITAS EN NOTACIÓN MENSURAL BLANCA A NOTACIÓN CONTEMPORÁNEA

Universidad del Valle de Guatemala

Facultad de Ciencias y Humanidades

Departamento de Matemática

DESARROLLO DE UN MODELO ALGORÍTMICO PARA LA TRANSFORMACIÓN DE PARTITURAS MUSICALES ESCRITAS EN NOTACIÓN MENSURAL BLANCA A NOTACIÓN CONTEMPORÁNEA

TRABAJO DE GRADUACIÓN PRESENTADO POR

Martha Eladia María Thomae Elías

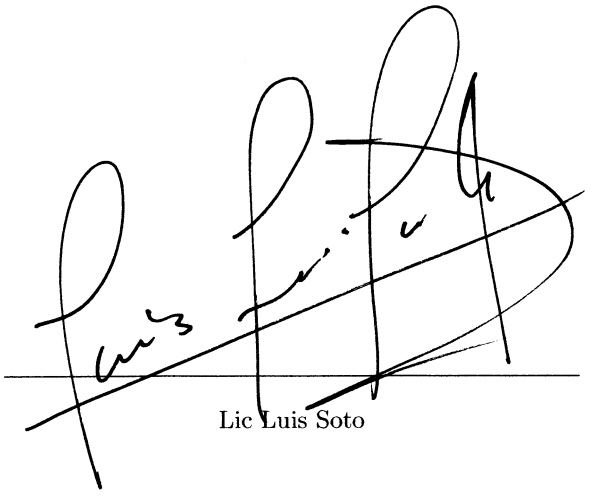
PARA OPTAR AL GRADO ACADÉMICO DE

Licenciada en Matemática

GUATEMALA

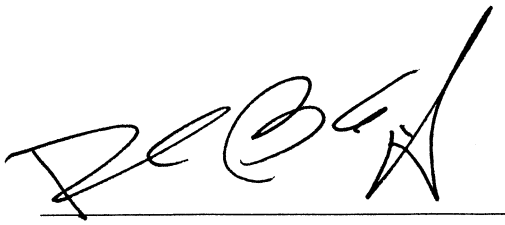
2013

Vo.Bo.:

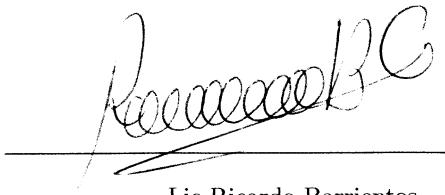


Lic Luis Soto

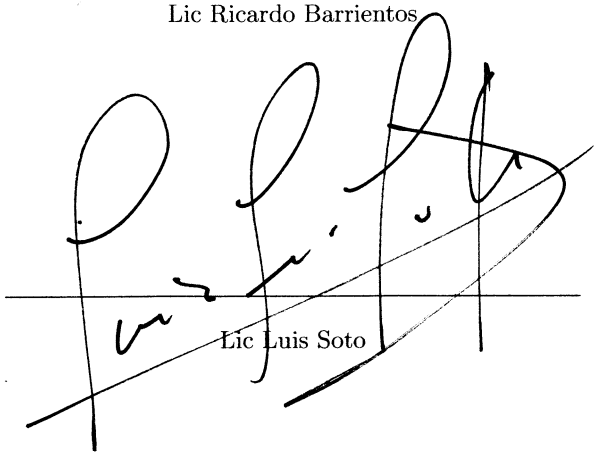
Tribunal Examinador:



PhD Raúl González



Lic Ricardo Barrientos



Lic Luis Soto

Fecha de aprobación: Guatemala, 26 de noviembre de 2013

Prefacio y agradecimientos

Desde pequeña siempre creí que mi carrera debía estar ligada a la ciencia. En mis años de estudio en matemática fui obligada a llevar un curso de arte, escogí piano, y esta simple elección fue la que me abrió un nuevo mundo. Jamás me había interesado el arte, no le veía utilidad (lo dice quien estudió matemáticas). En este curso conocí a Zoila Luz García-Salas. Ella no sólo fue mi maestra de piano, con ella comencé a aprender teoría musical. Debo admitirlo, no estaba en cero en lo que a formación musical se refiere, pero nunca hubo un esfuerzo consciente de mi parte por aprender música, no hasta ese momento. Cuando era pequeña había una sola cosa que me gustaba más que la ciencia: cantar. Siempre lo hacía, cuando estaba feliz o triste o enojada, siempre, incluso cuando estudiaba y me sentía cansada comenzaba a cantar, eso me daba energía para seguir trabajando. Estuve en el coro de mi colegio durante los 11 años que estudié en éste, no todos los años con igual dedicación debo admitir. Pero recuerdo claramente una sesión de coro donde sólo llegamos dos alumnas, el profesor nos sentó a la par de él en el piano y nos puso a cantar, ahí fue donde por primera vez vi de cerca una partitura. Le pregunté cómo la leía, en ese momento la clase dejó de ser de coro, me enseñó la distribución de las notas en el pentagrama y el valor de las notas, claro, esto no significaba que yo pudiera leer música, sólo que sabía qué nota era cuál (en el pentagrama con clave de sol) y cuánto valía (en tiempos de negra). Este profesor no estuvo los siguientes años pero dejó una semilla en mí, porque recuerdo que más grande compré un libro de teoría musical ya que me quedé interesada en el tema. El libro fue guardado en mi librera por muchos años.

Cuando tomé el curso de piano, Zoila Luz me enseñó a leer música y fue cuando comprendí mejor lo que mi profesor me había dicho, fue cuando recordé mi libro viejo de teoría musical. Empecé a leerlo y a aprender muchísimo, comencé a ganar interés y a meterme a cursos de música en la universidad. El primer curso que tomé fue *Introducción al jazz*, donde aprendí sobre la armonía, al momento completamente desconocida para mí; así fui aprendiendo poco a poco, con muchos agujeros supongo. Quería ingresar a los cursos de *Teoría musical* de la universidad, preguntaba que veían y observaba que estaba en desventaja. Fue así como inicié clases privadas con Zoila Luz, ella me enseñó teoría musical. Proseguí mis estudios con el Licenciado Luis Francisco Soto, que me niveló en teoría musical y armonía general. Es a estos dos personajes, la licenciada García-Salas y el licenciado Soto, a quienes les debo mi conocimiento teórico musical, conocimiento que me permitió hacer este trabajo.

Creo que me he desviado un poco de mi idea inicial. En mis años de estudio de matemáticas descubrí el mundo de la música, un mundo del que me enamoré. Al mismo tiempo, tuve unos problemas de salud mientras estudiaba mi carrera, aspecto que reconozco que me desmotivó respecto a la misma. Mi salud me forzó a pausar un semestre mi carrera, pero como no podía estar totalmente desocupada ingresé a un diplomado los sábados en una materia muy distinta. Mi amor por la música abrió en mí un interés por aprender sobre las otras artes, especialmente sobre una que anteriormente me disgustaba por no entenderla: las artes plásticas. El diplomado era de historia del arte, conocí nuevas personas y aprendí del arte plástico, mi mundo se expandió aún más. No digo que sea una experta en arte, un diplomado de un año no puede ofrecer esto, lo que digo es que mi panorama se abrió y dejé de ver el arte como algo inútil y más bien como un medio de placer estético, mi mente se abrió a un nuevo mundo. Un curso de música permitió todo esto, me introdujo al mundo del arte y me enamoré de él, de lo poco que conozco de él. Así que mientras me desenamoraba de la matemática, me enamoraba del arte (nunca en calidad de intérprete o artista, si no en calidad de estudioso del mismo). En determinado momento, consideré estudiar música y creí que una tesis de matemáticas enfocada a la música podría ayudarme a conseguir esto, que sería una buena carta de presentación para aplicar a algún ámbito teórico de la música como la musicología. Esto fue lo que motivó este trabajo, cuyo tema fue sugerido por Juan Pablo Pira (matemático y músico de esta universidad). Pero los resultados fueron inesperados incluso para mí. La música me dio tranquilidad cuando me desencante de la matemática por problemas de salud, y ahora con este trabajo me reenamoró de esta última. Los conocimientos aplicados en este trabajo se encuentran en la rama de la matemática discreta, la cuál no exploré mucho con anterioridad, y encontré gusto en ella. Hacer este trabajo fue un proceso agradable, sufrido, pero agradable; combina tres de mis pasiones: la música, la programación y la matemática.

Sólo quiero decir que el arte fue para mí un mundo donde encontré paz y en el cual pude desenchufarme un momento de mi cotidianidad, para tomar fuerzas y retornar. Este trabajo, aparte de ser una contribución importante para los estudiosos que se dedican a la transcripción de las partituras en notación mensural blanca por facilitar en cierta medida su trabajo, también es importante a nivel personal, ya que me permitió apreciar otra vez la matemática y a encontrarme en ella.

Muchas personas colaboraron en la elaboración del presente trabajo de graduación. Entre ellas agradezco principalmente a Juan Pablo Pira por la sugerencia del tema de tesis y por toda la ayuda brindada, realmente aprendí mucho; a la Lic. Zoyla Luz García-Salas y al Lic. Luis Francisco Soto, por toda la formación musical que ambos me han dado, sin ustedes no habría podido realizar este trabajo; a mis catedráticos, especialmente al Lic. Dorval Carías, Lic. Ricardo Barrientos y PhD. Raúl González, no sólo por su ayuda y sugerencias en la elaboración de este trabajo, si no también por todo lo que me han enseñado a lo largo de mi carrera; a la Lic. Nancy Zurita por todo el

apoyo que me ha dado, tanto en el sentido académico, al animarme a abordar este tema, como de forma más personal; y y al Lic. Michael Morales, por haber diseñado y autorizado el uso de la plantilla LaTex sobre la cuál he redactado el presente trabajo. También quiero agradecer a mi familia: German Thomae, Patricia Elías de Thomae y Anneliese Thomae, por toda su paciencia, apoyo y cariño; y a mis amigos, especialmente a: Andrés Lou, Milena Oliva y María José Prado, los tres me brindaron mucho apoyo en este tiempo.

Índice de contenido

Lista de figuras

Lista de tablas

Resumen

El objetivo del presente trabajo es desarrollar un modelo matemático para la transformación de partituras escritas en notación mensural blanca a notación actual. Hasta ahora existen reglas para la transcripción de estas partituras, pero los musicólogos tardan mucho tiempo en el proceso de transcripción. El fin de este trabajo es expresar estas reglas en un lenguaje formal para desarrollar un algoritmo que las modele y haga posible así la automatización del proceso de transcripción. El algoritmo toma como entrada un archivo de texto en formato LilyPond *.ly que describe la partitura mensural (sus voces, claves, mensuración y notas) y devuelve otro archivo de salida *.ly que contiene la información de la partitura actual. El modelo algorítmico se restringe a tres casos de mensuración: *tempus imperfectum cum prolatione imperfecta*, *tempus perfectum cum prolatione imperfecta* y *tempus imperfectum cum prolatione perfecta*. No toma en cuenta la coloración ni el uso de proporciones. Si la partitura producida aparentara ser errónea, se requiere de un análisis por parte del musicólogo pues en numerosos casos se ha detectado que los originales contienen errores cometidos por el copista.

1 Conceptos útiles de LilyPond

LilyPond es un programa de edición de partituras. Es un sistema compilado que toma como entrada un archivo de texto *.ly que describe la música, lo compila, y como salida genera un archivo pdf que contiene la música descrita en el texto. *LilyPond* se parece más a un lenguaje de programación que a los programas de edición gráfica de partituras, ya que la música no se escribe seleccionando las notas de una barra de herramientas gráfica y arrastrándolas a una partitura que se actualiza automáticamente, si no mediante el ingreso de un texto que la describe.(LilyPond.org, 2013)

En este capítulo se presentarán conceptos de *LilyPond* sobre notación mensural y actual que serán útiles en el presente trabajo. No se tratará información básica para el uso de *LilyPond*, ya que en ese caso puede consultarse el manual *Referencia de la notación* del equipo de desarrolladores de LilyPond. Los comandos relevantes para la notación mensural y actual son los siguientes:

- **Comando para agregar una nueva voz**

  Una partitura consta de varias voces sonando al unísono, cada una con una melodía independiente representada por su propio pentagrama. Debido a que *LilyPond* trabaja de manera predeterminada con notación contemporánea, es necesario utilizar la instrucción *\new MensuralVoice* en vez del usual comando *\new Voice* si se desea trabajar una determinada voz en notación mensural (Manual de notación en LilyPond, 2012, p. 416).

- **Comando *\time***

  El comando *\time* tiene dos significados diferentes dependiendo si la notación utilizada para determinada voz en la partitura es mensural o contemporánea (esto es fácil de identificar con la presencia del comando *\new MensuralVoice* o *\new Voice*, respectivamente). La palabra *\time* va seguida de una fracción, la cuál indica la mensuración o el tipo de compás. En el primer caso hay cuatro tipos de mensuración relevantes para el presente trabajo y son representados por las siguientes fracciones (Manual de notación en LilyPond, 2012, p. 418):

  1. *Tempus imperfectum cum prolatione imperfecta*, representado por la fracción *4/4*
  2. *Tempus perfectum cum prolatione imperfecta*, representado por la fracción *3/2*
  3. *Tempus imperfectum cum prolatione perfecta*, representado por la fracción *6/4*
  4. *Tempus perfectum cum prolatione perfecta*, representado por la fracción *9/4*

En el segundo caso a la palabra *\time* le sigue una fracción n/m, donde n y m son números naturales, esta fracción muestra el compás de la voz. También son cuatro los tipos de compás relevantes para este trabajo y corresponden a las transcripciones de cada una de las mensuraciones anteriores, respectivamente (Manual de notación en LilyPond, 2012, p. 59):

1. Compás 2/4
2. Compás 3/4
3. Compás 6/8
4. Compás 9/8

- **Comando *\clef***

El comando *\clef* indica la clave. El trazo de las claves mensurales y las contemporáneas varía mucho, en los siguientes dos cuadros se muestran las claves de ambas notaciones junto con el texto de entrada que las genera.

Cuadro 1.1: Código de LilyPond para claves en notación actual

Notación en LilyPond	Clave que representa
"G", G, treble	Clave de Sol 
"F", F, bass	Clave de Fa 
"G_8"	Clave de Sol una octava más abajo 

(Manual de notación en LilyPond, 2012, p. 16-17)

Cuadro 1.2: Código de LilyPond para claves en notación mensural

Clave			Notación en LilyPond
Sol			"mensural-g"
			"petrucci-g"
Fa			"mensural-f"
			"petrucci-f"
Do	en primera línea		"petrucci-c1"
	en segunda línea		"petrucci-c2"
	en tercera línea		"petrucci-c3"
	en cuarta línea		"petrucci-c4"
	en quinta línea		"petrucci-c5"

(Manual de notación en LilyPond, 2012, p. 417)

2 La noción de algoritmo

2.1 Problemas y sus soluciones algorítmicas

Un **problema** es una clase de preguntas, y cada una de estas preguntas es una **instancia** o **caso** del problema (Prather, 1986, p. 205). Dar solución a una instancia del problema, es resolver una pregunta específica, como cuál es la suma de 10 y 20. En cambio, resolver un problema implica describir un método general para solucionar cualquier instancia de dicho problema (Prather, 1986, p. 205). En el caso anterior, la solución consiste en describir el método para sumar dos números cualquiera; se imagina que se darán dos números arbitrarios a y b como entrada (que representan los sumandos), y la solución es describir el proceso general para obtener la salida c (que representa la suma).

Hay problemas de distinta naturaleza. A diferencia del problema anterior, cuya salida es numérica, hay algunos que tienen como salida un simple *verdadero* o *falso*, éstos son llamados problemas de decisión. Un ejemplo es determinar si un número es primo o no.(Prather, 1986, p. 205) Pero, sin importar la naturaleza del problema, resolverlo siempre requiere de un procedimiento general que, si cumple ciertas condiciones, es llamado un *algoritmo*. El presente capítulo trata sobre la resolución de problemas y el concepto de algoritmo, de cuándo un procedimiento puede ser llamado de esta manera, de los *procesos elementales* que componen un algoritmo, de los distintos tipos de *procesos compuestos* construidos a partir de los elementales, del lenguaje de los algoritmos (*pseudolenguaje*) y de la *metodología "top-down"* (*arriba-abajo*) para la composición de algoritmos complejos.

2.1.1. Principios para resolver un problema No existe una metodología universal que permita resolver cualquier problema, algunos matemáticos y filósofos trataron de encontrarla, y desarrollaron mucha herramienta útil en el camino, pero no lograron su objetivo. Al enfrentarnos con un problema, en lugar de buscar una metodología universal y aplicarla a dicho problema, se recurre a ciertos principios y sugerencias guía que no dan una solución inmediata al problema, pero ayudan al descubrimiento o invención de su solución. Estos principios fueron formulados por el matemático George Polya en su libro *How to Solve It* (*Cómo resolverlo*), y son el resultado de toda la herramienta aportada en investigaciones anteriores.(Prather, 1986, p. 206) No es parte del objetivo de este trabajo ilustrar todos estos principios, pero sí parafrasear algunos de los más útiles. Estos se muestran a continuación(Prather, 1986, p. 207):

1. Entender el problema

   Es muy importante entender el problema antes de proceder a resolverlo, ya que de lo contrario se pierde mucho tiempo. Es cierto que en casos donde el problema es grande, la situación puede cambiar, porque el mero acto de tratar de resolver el problema ayuda a entenderlo. Cabe señalar que es fundamental para la resolución de cualquier problema, entender qué tipo de variables conforman las entradas y las salidas deseadas. Algunas sugerencias para este paso son

   (a) Hacer una figura o dibujo que ilustre el problema

   (b) Identificar las *entradas* y *salidas* del problema, es decir, la información que se da a priori para resolver la pregunta y la respuesta de dicha pregunta, respectivamente

   (c) Introducir notación apropiada

2. Crear un plan

   (a) Encontrar conexiones entre la *entrada* y *salida*

   (b) Evaluar si hay algún otro problema, cuya solución ya se conozca, que se parezca, se relacione o sea una version más simple del problema en cuestión

   (c) Considerar introducir un problema auxiliar

   (d) Considerar exponer o plantear el problema de otra manera

3. Implementar el plan

   (a) Hacer un bosquejo del plan

   (b) Usar lenguaje preciso

   (c) Verificar cada paso

   (d) Hacer correcciones

4. Repasar

   (a) Probar la implementación

   (b) Mejorar el plan si se cree posible o adecuado

   (c) Evaluar si el resultado puede ser utilizado para la resolución de otro problema

   (d) Verificar los resultados

2.1.2. Definición de algoritmo Un **algoritmo** es una secuencia organizada de instrucciones para resolver cualquier pregunta que sea una instancia de un problema. Un *algoritmo* es escrito en un lenguaje preciso, pero esto no significa que sea un lenguaje de programación; el término algoritmo no presupone una conexión con ningún tipo de máquinas y se puede trabajar de forma independiente a estas nociones. La pregunta exacta (instancia) a responder por nuestro algoritmo, está completamente determinada por los valores de las *entradas* antes de la ejecución del algoritmo. La respuesta a la pregunta se obtiene luego de la ejecución y está dada por el valor de la *salida* del algoritmo. Como se señaló al principio de este capítulo, para que la solución a un problema sea un algoritmo, hay ciertos requisitos ha cumplir(Prather, 1986, p. 207):

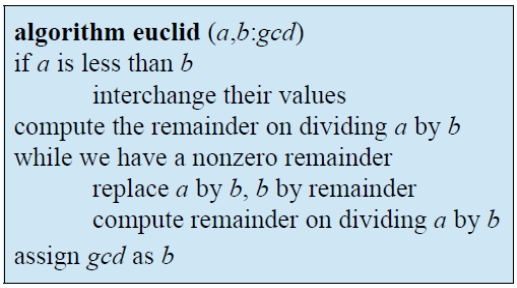
- La ejecución se lleva a cabo paso por paso, donde cada paso es un *proceso elemental* (ver definición en la siguiente sección).
- Las operaciones realizadas son deterministas, es decir, el azar no interviene en la ejecución del algoritmo.
- El número de pasos en la ejecución es finito, independientemente de los valores de las *entradas*.

2.1.3. Procesos elementales y cambios de estado Un **proceso elemental** es un proceso que puede ser realizado en un tiempo constante, independientemente del tamaño de la *entrada* del problema. Para aclarar esta idea, considérese el intercambiar dos valores, esto constituye un proceso elemental ya que puede realizarse en una unidad constante de tiempo, sin importar qué valores sean los que se ha de intercambiar. Por el contrario, el problema de hallar el máximo en una lista de números (la entrada es la lista), no toma un tiempo constante, depende de la longitud de la lista de números, por lo tanto, éste no es un proceso elemental. (Prather, 1986, p. 212)

Una solución algorítmica involucra variables: de entrada, de salida y algunas variables auxiliares. Durante la ejecución del algoritmo, se puede hablar del **estado** de éste, el *estado* se refiere a los valores que tienen todas las variables en ese momento(Prather, 1986, p. 212). Un ejemplo de esto se puede dar con el algoritmo euclideano, mediante el cuál se obtiene el máximo común divisor de dos números (ver *figura 2.1*).

Este algoritmo toma dos números a y b, los intercambia si $a < b$, luego obtiene el residuo de la división a/b, le asigna el valor de b a la variable a y el valor del residuo r a la variable b. El proceso anterior se repite hasta que el residuo es cero y el valor de b se le asigna a la variable del máximo común divisor *gcd*. En este caso se observan cuatro variables: las de entrada a y b, la variable auxiliar r y la variable de salida *gcd*. El *Cuadro 2.1* muestra un ejemplo de los estados de este algoritmo para un caso en particular con entradas $a = 180$ y $b = 252$, cada fila de la tabla representa un estado del algoritmo.

Figura 2.1: Algoritmo euclideano, solución del problema de hallar el máximo común divisor (gcd) de dos números



Cuadro 2.1: Ejemplo de los cambios de estado en el algoritmo euclideano, con entradas $a = 180$ y $b = 252$

a	b	r	gcd
180	252	0	0
252	180	0	0
252	180	72	0
180	72	72	0
180	72	36	0
72	36	36	0
72	36	0	0
72	36	0	36

(Prather, 1986, p. 213)

Como se puede ver con el ejemplo anterior, un proceso elemental causa una transformación en el estado de un conjunto de valores a otro. En contraste, un algoritmo también puede contener decisiones acerca de alguna **condición** de *estado*, es decir, interrogantes acerca de si se cumple o no cierta relación entre variables o de ellas con algún otro objeto. Por ejemplo, en el algoritmo euclideano las frases "si a es menor que b" y "hasta que el residuo es 0" son ejemplos de decisiones sobre condiciones de estado. Estas interrogantes sobre las condiciones de las variables, utilizan el estado para obtener alguna respuesta de tipo "verdadero" o "falso", pero no efectúan ningún cambio en el estado, a diferencia de los procesos elementales, los valores de las variables no cambian. (Prather, 1986, p. 213)

2.2 Herramienta básica para escribir un algoritmo

Durante todo este tiempo se ha utilizado el *pseudolenguaje* para la escritura de los algoritmos (ver *figura 2.1*). El **pseudolenguaje** no es un lenguaje de programación ni un lenguaje hablado; es más bien una mezcla de inglés con simbolismo matemático, pero su *estructura gramatical* es similar a la de un lenguaje de programación. Son estas estructuras gramaticales las que serán presentadas a continuación, esto establecerá directrices que ayudan a escribir algoritmos de una

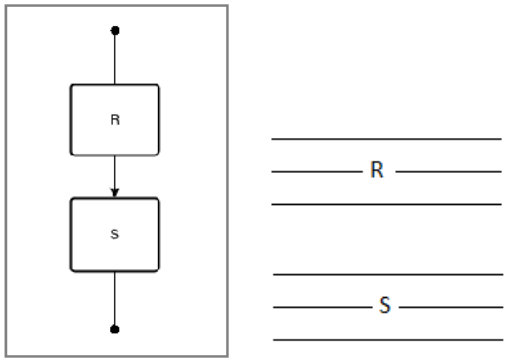
manera consistente, asegurando que todo aquél que lea el algoritmo comprenda el mismo mensaje, y que éste sea el que se quiere comunicar. Para hablar de dichas estructuras gramaticales se debe conocer lo que es un *proceso compuesto*. (Prather, 1986, p. 215)

2.2.1. Procesos compuestos Los **procesos compuestos** son uniones de procesos elementales mediante el uso de constructores gramaticales, estos últimos son de tres tipos: secuencia, selección y repetición. Al combinar de esta forma los procesos elementales se forman procesos compuestos con complejidad cada vez mayor. Todos estos procesos, elementales y complejos, combinados con el objetivo de dar solución a un problema, llegan a constituir un algoritmo. A continuación se describe cada uno de los constructores gramaticales señalados anteriormente y puede verse la Figura 2.8 que ejemplifica cada uno de estos:

- Secuencia

  Dos procesos elementales están en secuencia cuando se ejecuta uno luego del otro. Supóngase que R y S son dos procesos, la secuencia $R \circ S$ se representa en pseudocódigo tal y como lo muestra la Figura 2.2b, nótese que por ser una secuencia, ambos procesos poseen la misma sangría o tabulación (es decir, están alineados en el algoritmo). El diagrama de flujo de la Figura 2.2a muestra la forma de ejecutar la secuencia de instrucciones. (Prather, 1986, p. 217)

Figura 2.2: Secuencia



(a) Diagrama de flujo (b) Pseudocódigo

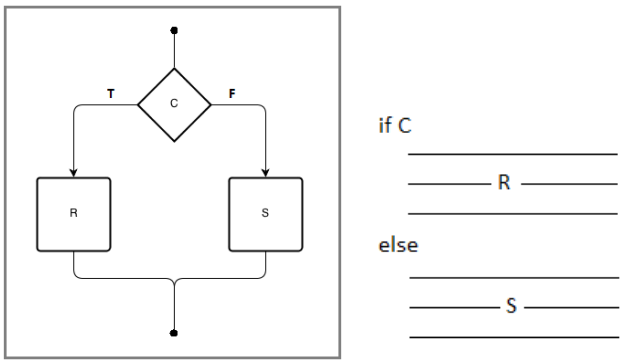
(Prather, 1986, p. 217)

- Selección

  Ahora, supóngase que R y S son dos procesos (elementales o compuestos) y que C es una condición que puede ser o no satisfecha en el *estado* actual del algoritmo. La selección "R if C else S" ("R si C, de no ser así S") se ejecuta de la manera indicada por el diagrama de flujo de la Figura 2.3a. En pseudocódigo esto se representa como se muestra en la Figura 2.3b,

nótese que el lugar ocupado por el proceso a ejecutarse si se cumple o no la condición posee una tabulación adicional al lugar ocupado por la condición (R y S no están alineados con C, si no que tienen una tabulación más). (Prather, 1986, p. 217)

Figura 2.3: Selección "R if C else S"

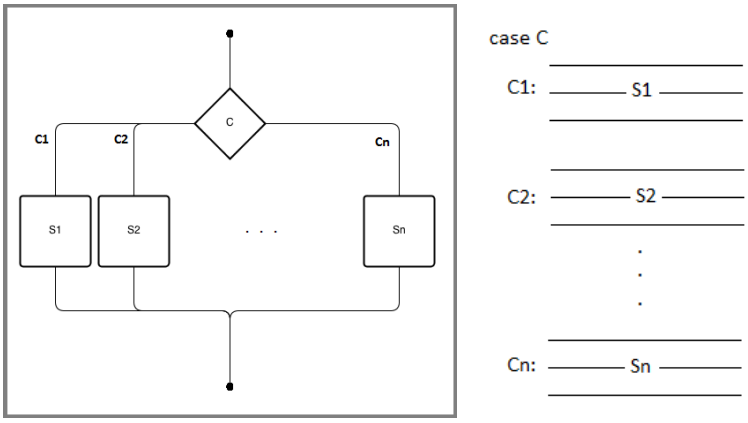


(a) Diagrama de flujo (b) Pseudocódigo

(Prather, 1986, p. 217)

Existe un caso especial de selección que, en lugar de tener sólo la capacidad de escoger entre dos opciones, permite elegir sobre un número n de casos C_1, C_2, ..., C_n de una condición general C. Puede ser que si C tiene la configuración C_1 se quiere llevar a cabo el proceso S_1, si C tiene la configuración C_2 el proceso adecuado sea el S_2, y así sucesivamente. La forma adecuada de escribir esto en pseudolenguaje y la ejecución de este grupo de instrucciones se muestra en la Figura 2.4. (Prather, 1986, p. 222)

Figura 2.4: Selección con múltiples opciones



(a) Diagrama de flujo (b) Pseudocódigo

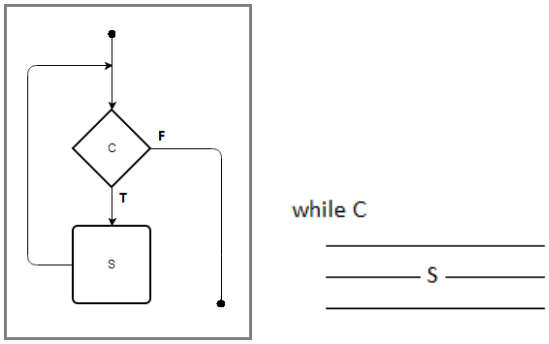
(Prather, 1986, p. 222)

- Repetición

  Hay tres tipos de repetición, para dos de ellos es necesaria una condición. Sea S es un proceso y C una condición, los dos ciclos (repeticiones) son:

  – **while**: el proceso S se repite mientras la condición C se cumpla y el ciclo se detendrá cuando la condición C sea falsa. Además, nótese que si la condición C es falsa desde el inicio, el proceso S jamás se ejecuta.

Figura 2.5: Ciclo *while* "S while C"

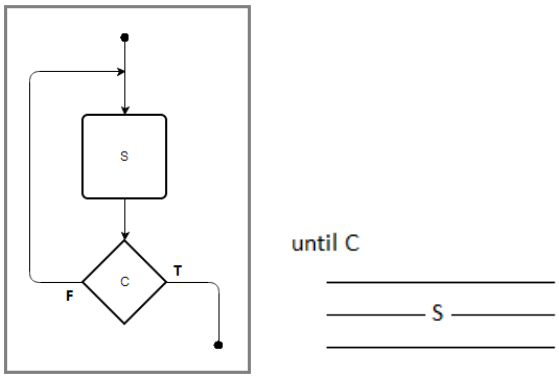


(a) Diagrama de flujo (b) Pseudocódigo

(Prather, 1986, p. 219)

  – **until**: el proceso S se repite hasta que ocurra la condición C, es decir, mientras la condición C no sea satisfecha el proceso S se llevará a cabo; el ciclo se detendrá cuando la condición C sea verdadera. Además, nótese que si la condición C es verdadera desde el inicio, el proceso S se ejecuta una sola vez, esto se debe a que el proceso es ejecutado antes de evaluar la condición, por lo que el proceso S es realizado al menos una vez.

Figura 2.6: Ciclo *until* "S until C"

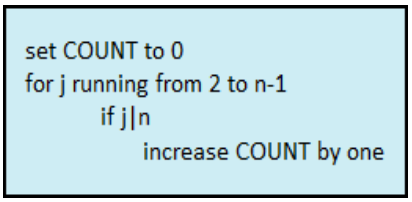


(a) Diagrama de flujo (b) Pseudocódigo

(Prather, 1986, p. 219)

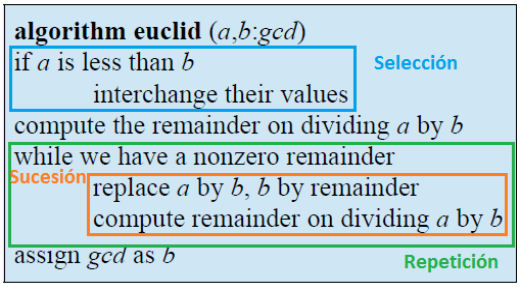
El tercer ciclo es **for**. Este ciclo tiene una variable auxiliar, llamada variable control, que determina cuántas veces exactas se efectúa determinado proceso (Prather, 1986, p. 224). Ejemplo de un algoritmo que usa este tipo de repetición es:

Figura 2.7: Algoritmo que encuentra el número de divisores diferentes de 1 y del mismo número



La siguiente figura muestra nuevamente el algoritmo euclideano con el fin de ejemplificar cada una de las estructuras gramaticales explicadas anteriormente (secuencia, selección y repetición).

Figura 2.8: Ejemplo de las tres estructuras gramaticales (secuencia, selección y repetición) en un algoritmo



2.2.2. Metodología "top-down"

Un algoritmo puede ser muy complejo, lo cual dificulta escribirlo en detalle desde un inicio. En vez de ello, se escribe un plan general que provee la estructura general de la solución, y más adelante se llenan los detalles. Esos subprocesos que luego se llenarán con detalle, pueden ser escritos en lenguaje natural; más adelante el diseñador vuelve a ellos, tratando cada subproceso como un problema en sí. De esta manera pasamos de un nivel superior a niveles con mayor detalle, bajando hasta el nivel donde todos los procesos son elementales y cada condición es una condición elemental. Ésta es la **metodología top-down** para el diseño de algoritmos. (Prather, 1986, p. 220)

2.2.3. Directrices para escribir algoritmos con pseudolenguaje

Como se mencionó al principio de la sección, el pseudolenguaje es una manera eficaz de escribir algoritmos, porque todo aquél que lee un algoritmo escrito de esta forma, entiende lo mismo, y comprende lo que el algoritmo hace. Pero esta comprensión universal se debe tanto a que las personas entienden el simbolismo matemático y las estructuras gramaticales (de secuencia, selección y repetición), como a ciertas pautas o directrices que sigue el pseudolenguaje, éstas son (Prather, 1986, p. 225-226):

1. No usar punto, especialmente al final de cada línea.
2. Los procesos que van en secuencia deben estar alineados verticalmente
3. Identificar las estructuras de selección y repetición tal y como se indicó en la *sección 2.2.1*
4. Escribir las condiciones (en selección o repetición) de forma clara, en inglés o en lenguaje matemático
5. Escribir los procesos en inglés o en lenguaje matemático, comenzando siempre con un verbo (ya que esto expresa la acción a tomar, y es la acción lo que diferencia a un proceso de una condición). Por ejemplo puede escribirse: "COUNT = 0" o "set COUNT to 0"; tanto la expresión matemática como el lenguaje natural es correcto, pero en el segundo caso la instrucción debe iniciar en verbo.
6. Usar cuidadosamente la sangría (tabulación).
7. El encabezado del algoritmo contiene un paréntesis que indica las entradas. En ocasiones este paréntesis también indica las salidas del algoritmo, pero esto opcional; dichas salidas se encuentran separadas de las entradas por dos puntos. Si hay más de una entrada, éstas se separan entre sí por comas (lo mismo aplica a las salidas cuando hay más de una).

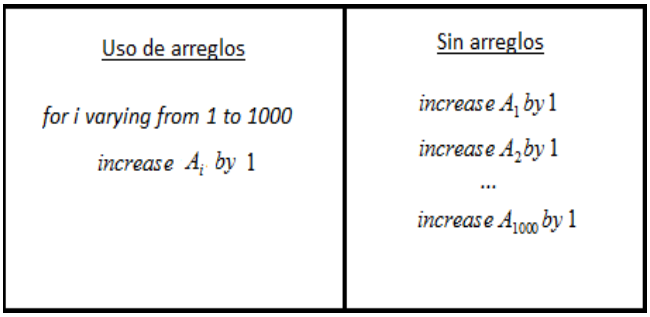
2.3 Herramienta adicional

Es claro que la noción de algoritmo no tiene por qué relacionarse con el aspecto computacional; sin embargo, sí toma muchas de sus estructuras gramaticales (la secuencia, la selección y repetición) y esto permite que el algoritmo efectúe procesos que pueden ser expresados en lenguajes de programación. El pseudolenguaje se extiende un poco más para expresar procesos que otros lenguajes de programación permiten realizar, de este modo se agranda el conjunto de procesos que el algoritmo puede llevar a cabo. En esta sección se introducen dos conceptos. El primero es el de *arreglos* o *variables con subíndices* (en inglés *arrays* o *subscripted variables*), esto extiende enormemente el marco de problemas que se pueden resolver. El segundo concepto es el de *módulos*, se necesita proveer mecanismos para tratar los algoritmos como módulos independientes, de modo que estos módulos puedan ser llamados por otros algoritmos y así efectuar de forma directa la acción de ese algoritmo (módulo) que fue llamado. (Prather, 1986, p. 232)

2.3.1. Arreglos Los arreglos sirven para realizar un proceso repetitivo en un gran número de variables. Por ejemplo, supóngase que se tienen 1000 números y a todos ellos se les quiere sumar una unidad. Este proceso implicaría tener una variable para cada uno de los mil números y agregarle una unidad a cada una de estas variables. Para no tener mil nombres de variables diferentes, se utiliza la idea de **arreglo** o **variables con subíndices** (*subscripted variables*). En lugar de tener n variables (en este caso 1000) se tiene una sola variable A, llamada arreglo, que

consiste en una sucesión de elementos. En este caso, el arreglo A consiste en una sucesión de mil números, para hacer referencia a cada uno de ellos se toma el nombre de la variable A y se le coloca un subíndice que representa una determinada posición en el arreglo, de este modo A_m es el m-ésimo elemento del arreglo. El uso de arreglos simplifica mucho la notación, como puede verse en el Figura 2.9 donde se presenta las dos versiones del algoritmo anterior (una que usa las 1000 variables y otra que usa un arreglo). Nótese que si se considera individualmente cada una de las mil variables, se requiere mil enunciados para lograr el mismo resultado que se obtiene con las dos líneas de código al usar un arreglo. (Prather, 1986, p. 233-234)

Figura 2.9: Comparación del pseudocódigo al usar arreglos y al no usarlos



2.3.2. Algoritmos tratados como módulos Ya se mencionó antes en la sección "Metodología top-down" que se puede dividir un problema en subproblemas, y realizar algoritmos secundarios para cada uno de esos subproblemas, de modo que luego un algoritmo general llame a esos algoritmos secundarios cuando sea apropiado, solucionando así el problema principal. Una importante variación de esta técnica, la llamada de un algoritmo dentro de otro algoritmo, es ver a ese módulo que se llamó como una función. La principal diferencia se da en el mecanismo de llamada. Al llamar a un algoritmo, se escribe sencillamente la palabra "call" y el nombre del algoritmo junto con sus argumentos de entrada. Esta frase ("call" y el nombre del algoritmo) es un proceso y se ubica en una sola línea del algoritmo que efectúa la llamada. Al contrario, al llamar a una función, ésta no se coloca en una línea aparte ni se utiliza la palabra "call", en vez de esto, el nombre de la función y sus argumentos de entrada aparecen en el contexto de algún proceso descrito en el algoritmo general. Para ejemplificar esta diferencia, se muestra a continuación el algoritmo *bisection* y el algoritmo *ratadd*. El primero, llama al algoritmo *selectionsort*, usando "call selectionsort" y el argumento de entrada A. El segundo, *ratadd*, tiene por objetivo sumar dos fracciones ($\frac{a_1}{a_2} + \frac{b_1}{b_2} = \frac{c_1}{c_2}$), para ello se necesita encontrar el máximo común divisor de diversas cantidades, por lo tanto el algoritmo *ratadd* llama a la función *gcd* (por *greatest common divisor*, máximo común divisor en inglés). Esta llamada la realiza en medio de diferentes procesos, como es hallar el denominador (c_2) de la fracción resultante ($c_2 = \frac{a_2 b_2}{gcd(a_2, b_2)}$) o el hallar el

máximo común divisor d del numerador c_1 y denominador c_2 de la fracción resultante. (Prather, 1986, p. 238, 240-241)

Figura 2.10: Llamada del **algoritmo** *selectionsort* por el algoritmo *bisection*

```
algorithm selectionsort (A)
for each i from 1 to n - 1
        set small at i
        for each j from i +1 to n
                if A_j < A_small
                        tak small from j
        interchange A_small with A_j
```

```
algorithm bisection (A, x: found, location)
call selectionsort(A)
set top and bottom to first and last respectively
initialize found as 'false'
while top ≤ bottom and ¬ found
        compute mid
        case comparison
                x < A_mid : change bottom to mid - 1
                x < A_mid : change top to mid + 1
                x = A_mid : set found to 'true'
assign location as mid
```

(Prather, 1986, p. 238, 240)

Figura 2.11: Llamada de la **función** *gcd* por el algoritmo *ratadd*

```
function gcd (a, b)
if a is less than b
        interchange their values
compute the reaminder on dividing a by b
while we have a nonzero remainder
        replace a by b, b by remainder
        compute remainder on dividing a by b
return gcd as b
```

```
algorithm ratadd (a, b: c)
set c2 = a2 b2 / gcd(a2 , b2 )
set c1 = ( c2 / a2 ) a1 + ( c2 / b2 ) b1
compute d = gcd( c1 , c2 )
divide c1 by d
divide c2 by d
```

(Prather, 1986, p. 241-242)

Nótese que al escribir una función hay ciertas características sintácticas que la diferencian de un algoritmo. La primera diferencia puede notarse en la línea de título del algoritmo o función. En el caso de un algoritmo se utiliza la palabra *algorithm* o la palabra *procedure* al inicio de dicha línea; mientras que en el caso de una función se utiliza la palabra *function*. En segundo lugar, la última línea de una función siempre comienza con la palabra *return*, que señala la salida de la función; el valor que adquiere dicha salida se convierte en el valor de la función misma. Tanto los algoritmos como las funciones realizan un proceso, pero la función obtiene un resultado del mismo (que puede ser numérico, booleano, etc.). Al llamar a una función se llama al resultado de la misma (no sólo al proceso); por eso se puede llamar a una función en medio de un proceso, porque el valor que toma sirve a los cálculos de éste último. (Prather, 1986, p. 240-241)

3 El grupo de las clases de congruencia módulo n

El presente capítulo contiene conceptos de teoría de números y álgebra moderna. La primera sección se centra en el concepto de *relación de congruencia* y cómo ésta, al ser una relación de equivalencia, genera un partición del conjunto de los números enteros Z. La segunda sección gira en torno a un tema más algebraico: el concepto de *grupo cociente*. Por último, la tercera sección muestra que la partición del conjunto de los números enteros generada por una relación de congruencia es un grupo, específicamente es un grupo cociente.

3.1 Relaciones y clases de congruencia módulo n

El concepto de *relación de congruencia* es fundamental en la teoría de números. Fue introducido por primera vez por el gran matemáticos Carl Friedrich Gauss a inicios del siglo XIX como la base de su nueva teoría, la teoría de congruencias, que marcó una nueva era en la teoría de números. Gauss formula la teoría de congruencias a partir de la siguiente definición: (Oystein, 1988, p. 210-211)

Definición. Se dice que dos enteros a y b son *congruentes módulo* n cuando su diferencia es divisible dentro de n. Esto se representa como $a \equiv b \pmod{n}$

La definición de relación de congruencia dada por Gauss puede reformularse de la siguiente manera:

Definición. a y b son *congruentes módulo* n cuando ambos tienen el mismo residuo al ser divididos dentro de n.

Para probar la equivalencia entre ambas definiciones es necesario hacer uso de la *propiedad de la división euclideana*, la cual establece que si m es dividido entre n, se obtiene un cociente q y un residuo r únicos, donde el residuo es un entero no negativo menor que n.

Teorema 1. ***Propiedad de la división euclideana*** *(Doerr y Levasseur, 1985, p. 270)*
Si m, $n \in Z$, $n > 0$, entonces existen dos enteros únicos q y r, tal que $m = nq + r$ con $0 \leq r < n$.

A continuación se muestra la prueba de la equivalencia de las dos definiciones de relación de congruencia.

Demostración. Sean a, $b \in Z$ y n entero positivo entonces, por el Teorema 1, $\exists!$ $q_a, r_a, q_b, r_b \in Z$ tal que $a = q_a n + r_a$ y $b = q_b n + r_b$ con $0 \leq r_a, r_b < n$.

De esto se tiene: $a - b = (q_a - q_b)n + (r_a - r_b)$
Por lo tanto: n divide a $a - b \iff r_a - r_b = 0 \iff r_a = r_b$ □

Nótese que la relación de congruencia para un módulo n fijo es una relación de equivalencia, por cumplir las 3 propiedades siguientes: (Niven, 1991, p. 128)

1. Reflexividad: $a \equiv a \pmod{n}$
2. Conmutatividad: Si $a \equiv b \pmod{n}$, entonces $b \equiv a \pmod{n}$
3. Transitividad: Si $a \equiv b \pmod{n}$ y $b \equiv c \pmod{n}$, entonces $a \equiv c \pmod{n}$

Una relación de equivalencia $\sim$ sobre un conjunto K puede denotarse como $(K, \sim)$. Esta relación define subconjuntos disjuntos en K, llamados clases de equivalencia. Dado un elemento $a \in K$, el conjunto de todos los elementos asociados a a por la relación $\sim$ define la clase $[a] = \{x \in K | x \sim a\}$, llamada clase de equivalencia asociada a a. Al elemento a se le llama el *representante de la clase de equivalencia*. Cualquier elemento de la clase de equivalencia puede ser llamado el representante de la clase, debido a la propiedad conmutativa de la relación de equivalencia.

Por lo tanto, una relación de congruencia módulo n, al ser una relación de equivalencia, define clases de equivalencia que particionan[1] el conjunto de los enteros Z en conjuntos disjuntos, no vacíos, cuya unión es Z. Estas clases son de la forma $[a] = \{x \in Z | x \equiv a \pmod{n}\}$, es decir, la clase de equivalencia asociada a a son todos los $x \in Z$ tal que x y a tienen el mismo residuo al ser divididos dentro de n. Por ello, y sin pérdida de generalidad, se puede tomar como representante de la clase de equivalencia al residuo r. Por lo tanto, la relación de congruencia módulo n define n clases de equivalencia representadas por los enteros r tales que $0 \leq r < n$, es decir, las n clases:

- $[0]_n = \{0, 0 \pm n, 0 \pm 2n, 0 \pm 3n, ...\}$

  Conjunto de enteros con residuo 0, todo $x \in [0]_n$ cumple con $x \equiv 0 \pmod{n}$

- $[1]_n = \{1, 1 \pm n, 1 \pm 2n, 1 \pm 3n, ...\}$

  Conjunto de enteros con residuo 1, todo $x \in [1]_n$ cumple con $x \equiv 1 \pmod{n}$

  ...

- $[n-1]_n = \{n-1, (n-1) \pm n, (n-1) \pm 2n, (n-1) \pm 3n, ...\}$

  Conjunto de enteros con residuo n-1

[1] Una partición de un conjunto A es una familia de subconjuntos $\{A_i, i \in N\}$ tal que:
$A_i \neq \emptyset, \forall i$
$\cup_{i=1}^{\infty} A_i = A$
y si $i \neq j$ entonces $A_i \cap A_j = \emptyset$.
Toda clase de equivalencia define una partición.

A estas clases de equivalencia generadas por la relación de congruencia se les denomina **clases de congruencia**. En el presente trabajo se hace uso del conjunto de clases de congruencia $\{[0]_n, [1]_n, ..., [n-1]_n\}$. Más adelante, en la sección 3.3 se demostrará que el conjunto de clases de congruencia es un grupo (al probar que es el grupo cociente Z/nZ). Antes de esto es necesario aclarar algunos conceptos para entender qué es un grupo cociente y por qué Z/nZ es un grupo cociente, éste es el propósito de la siguiente sección.

3.2 Grupo cociente

El objetivo de esta sección es probar, por medio de los siguientes teoremas y ejemplos, que el conjunto Z/nZ es un grupo. Para esto es necesario definir algunos conceptos básicos, empezando por el término *grupo*.

Definición. Se dice que un conjunto G es un **grupo** bajo una operación binaria $\cdot$ si se cumplen las siguientes 4 propiedades:

1. Cerradura: Dados $a, b \in G$, entonces $a \cdot b \in G$

2. Asociatividad: Dados $a, b, c \in G$ se cumple $(a \cdot b) \cdot c = a \cdot (b \cdot c)$

3. Neutro (o identidad): Existe $e \in G$ tal que para cualquier $a \in G, a \cdot e = e \cdot a = a$. Al elemento e en G que cumple con esta propiedad se le llama *neutro* o *identidad*

4. Inversos: Para cualquier $a \in G$, existe un elemento $a^{-1} \in G$ tal que $a \cdot a^{-1} = a^{-1} \cdot a = e$. Al elemento a^{-1} de G se le llama inverso de a.

Para entender mejor la noción de grupo se presentan los siguientes dos ejemplos:

Ejemplo 1. El conjunto de los números enteros $Z = \{..., -3, -2, -1, 0, 1, 2, 3, ...\}$ es un grupo bajo la adición $(+)$. Nótese que la suma de dos enteros siempre da un entero (cerradura) y que la suma en los enteros es asociativa. Además $0 \in Z$ se comporta como la identidad $(0+a = a+0 = a, \forall a \in Z)$. Por último, todos los enteros tienen un inverso aditivo en Z (para cualquier $a \in Z$ existe $-a$ también en Z que cumple con que $a + (-a) = (-a) + a = 0$).

Ejemplo 2. El conjunto de los números enteros $Z = \{..., -3, -2, -1, 0, 1, 2, 3, ...\}$ no forma un grupo bajo la multiplicación.

Nótese que las primeras 3 propiedades de un grupo se cumplen para Z bajo la multiplicación:

- El producto de enteros es un entero (cerradura)

- El producto en los enteros es asociativo

- Y existe un elemento en los enteros que se comporta como la identidad: 1

Pero hay enteros que no poseen inverso multiplicativo en el conjunto Z. Se sabe que para cualquier entero a su inverso multiplicativo es $\frac{1}{a}$, ya que $a(\frac{1}{a}) = (\frac{1}{a})a = 1$. Pero, con excepción de $a = 1$ y $a = -1$, el inverso multiplicativo $\frac{1}{a}$ no está en Z (es una fracción no entera) y como queda claro del inciso 4 de la definición de grupo, para que un conjunto sea considerado un grupo el inverso de cada uno de sus elementos debe estar también dentro del conjunto ("Para cualquier $a \in G$, existe un elemento $a^{-1} \in G$ tal que...").

El Ejemplo 1 y 2 claramente muestran que tanto el conjunto como la operación definida en el mismo son importantes para la noción de *grupo*. Algunos grupos cumplen con una propiedad adicional: la conmutatividad; estos grupos reciben un nombre especial.

Definición. Se dice que G es un **grupo abeliano** bajo la operación $\cdot$ si G es un grupo bajo dicha operación y cumple un quinta propiedad adicional:

5 Conmutatividad: Dados cualquiera $a, b \in G$, $a \cdot b = b \cdot a$

Ejemplo 3. El conjunto de los enteros Z es un grupo abeliano bajo la suma.

Hasta el momento se tiene claro que un grupo es un conjunto con una operación definida en él y que cumple ciertas propiedades. Como cualquier conjunto, un grupo posee subconjuntos; algunos de estos subconjuntos también son grupos y reciben un nombre especial.

Definición. Un subconjunto H de un grupo G se dice que es **subgrupo** de G, si H es a su vez un grupo bajo la operación definida en G.

Ejemplo 4. El conjunto $nZ = \{nk | k \in Z\} = \{..., -3n, -2n, -n, 0, n, 2n, 3n, ...\}$ para un $n \in Z$ es un subgrupo de Z. Es obvio que nZ es subconjunto de Z. Para probar que nZ es subgrupo de Z es necesario establecer que nZ es un grupo bajo la misma operación que hace a Z un grupo (la adición):

- Cerradura: Sean $nj, nk \in nZ$, entonces $j, k \in Z$. Ahora examinemos la suma de los dos elementos de nZ, $nj + nk = n(j + k)$ con $j + k \in Z$ por la cerradura de Z con respecto a la adición. Por lo tanto, $nj + nk = n(j + k) \in nZ$.

- Asociatividad: Sean $a, b, c \in nZ$, entonces $a, b, c \in Z$ por ser $nZ \subset Z$. Como a, b y c son elementos de Z, y Z es grupo, se cumple la propiedad asociativa: $(a + b) + c = a + (b + c)$.

- Neutro: El elemento neutro es 0 ($0 \in nZ$ y $0 + a = a + 0 = a$ para todo $a \in nZ \subset Z$).

- Inversos: Para cualquier $nk \in nZ$, $k \in Z$ por lo tanto su inverso $-k \in Z$, de esto se tiene que $n(-k) \in nZ$. La suma de estos dos elementos de nZ está dada por $nk + n(-k) = n(k + (-k)) = n0 = 0$ y de igual manera $n(-k) + nk = 0$. Por lo tanto, para cualquier $nk \in nZ$, existe un $n(-k) \in nZ$ que actúa como su inverso aditivo.

Por lo tanto, nZ es subgrupo de Z.

Como se mencionó al inicio de esta sección, el objetivo de la misma es probar que Z/nZ es grupo, específicamente Z/nZ es un *grupo cociente*. Para poder entender el concepto de *grupo cociente*, es necesario introducir otros dos conceptos: *clase lateral* y *subgrupo normal*.

Definición. Sea G un grupo y H un subconjunto de G. Dado un $a \in G$, se definen los conjuntos $aH = \{ah | h \in H\}$ y $Ha = \{ha | h \in H\}$. Si H es subgrupo de G entonces aH es llamado **clase lateral izquierda de H en G**, Ha se denomina **clase lateral derecha de H en G** y al elemento a se le conoce como el **representante de la clase lateral**.

El siguiente lema muestra una de las propiedades que cumple toda clase lateral, dicha propiedad es útil en la demostración del Teorema 2.

Lema 1. *Sea H subgrupo de G y $a \in G$. Entonces se cumple que $aH = H$ si y solo si $a \in H$*

Demostración. Sea H un subgrupo de G y $a \in G$.

($\Rightarrow$) Supóngase que $aH = H$. Como H es un grupo en sí, existe un elemento neutro $e \in H$, por lo tanto $a = ae \in aH$, pero como $aH = H$ entonces $a \in H$ también.

($\Leftarrow$) Ahora suponga que $a \in H$, entonces se cumple que:

($\subseteq$) Sea $ah \in aH$, entonces $h \in H$. Como también $a \in H$ y H es grupo, entonces, por cerradura, el producto $ah \in H$. Por lo tanto $aH \subseteq H$.

($\supseteq$) Sea $h \in H$. Como $a \in H$, también su inverso $a^{-1} \in H$. Por cerradura, el elemento $a^{-1}h \in H$, entonces $h = (aa^{-1})h = a(a^{-1}h) \in aH$. Por lo tanto $H \subseteq aH$

Por lo tanto, $aH = H$.

□

Ahora que se ha dejado claro el concepto de *clase lateral*, es momento de definir lo que es un *subgrupo normal*.

Definición. Un subgrupo H de un grupo G es denominado un **subgrupo normal** de G si $aH = Ha$ para todo $a \in G$. Esto se denota mediante $H \triangleleft G$.

Nótese que cuando G es grupo abeliano, cualquier subgrupo H de G es un subgrupo normal. Esto es cierto ya que, dado un $a \in G$, si se toma un $h \in H \subset G$, se tiene que ambos elementos $a, h \in G$. Por ser G un grupo abeliano $ah = ha$, de lo cual se obtiene $aH = \{ah | h \in H\} = \{ha | h \in H\} = Ha$, comprobando así que $H \triangleleft G$.

Ejemplo 5. nZ es subgrupo normal de Z, por ser nZ subgrupo de Z (ejemplo 4) y Z un grupo abeliano (ejemplo 3).

Los subgrupos normales son de gran importancia, ya que si el subgrupo H de G es normal, el conjunto de las clases laterales izquierdas (o derechas) de H en G también es un grupo, denominado *grupo cociente de G por H*, como se demuestra a continuación.

Teorema 2. *Cuando H es un subgrupo normal de G, el conjunto definido como $G/H = \{aH | a \in G\}$ es también un grupo bajo la operación $(aH)(bH) = (ab)H$.*

Demostración. Sea G un grupo y $H \triangleleft G$. Defínase $G/H = \{aH | a \in G\}$.
A probar: 1. la operación $(aH)(bH) = (ab)H$ está bien definida; 2. El conjunto G/H es grupo.

1. Para probar que la operación esta bien definida es necesario demostrar que la correspondencia definida arriba y que va de $G/H \times G/H$ a G/H es realmente una función, es decir: dados aH, $a'H$, bH , $b'H \in G/H$ tales que $aH = a'H$ y $bH = b'H$, entonces $aH \cdot bH = a'H \cdot b'H$.

   Debido a que $aH = a'H$ y $bH = b'H$ tenemos que $a' = ah_1$ y $b' = ah_2$ para alguna h_1 y h_2 en H y por tanto $a'b'H = ah_1bh_2H = ah_1bH = ah_1Hb = aHb = abH$ (la primera igualdad consiste en una simple sustitución, mientras que la segunda y cuarta se deben al Lema 1, y la tercera y quinta al hecho de que H es subgrupo normal).

2. Para probar que G/H es un grupo se debe demostrar que se cumplen las 4 propiedades: cerradura, asociatividad, exixtencia del elemento neutro (o identidad) y existencia de inversos.

   (a) Cerradura:

   Si aH, $bH \in G/H$ entonces a, $b \in G$ y, como G es cerrado, esto implica que $ab \in G$, por lo que $(ab)H \in G/H$.

   (b) Asociatividad:

   Sean aH, bH, $cH \in G/H$ entonces a, b, $c \in G$ y se cumple:

$$\begin{aligned}
(aH \cdot bH) \cdot cH &= (ab)H \cdot cH && \text{por definición de la operación en G/H} \\
&= ((ab)c)H && \text{por definición de la operación en G/H} \\
&= (a(bc))H && \text{por asociatividad en G} \\
&= aH \cdot (bc)H && \text{por definición de la operación en G/H} \\
&= aH \cdot (bH \cdot cH) && \text{por definición de la operación en G/H}
\end{aligned}$$

   (c) Identidad:

   Como G es grupo, $\exists e \in G$ tal que $\forall a \in G$, $ae = ea = a$. Como $e \in G$, $eH \in G/H$. Y para todo $aH \in G/H$ se cumple que:

$aH \cdot eH = (ae)H = aH$

y $eH \cdot aH = (ea)H = aH$

Por lo tanto, $\exists eH \in G/H$ tal que $\forall aH \in G/H$, $aH \cdot eH = eH \cdot aH = aH$

(d) Inversos:

$\forall aH \in G/H$, $a \in G$. Entonces, como G es grupo, $\exists a^{-1} \in G$ tal que $aa^{-1} = a^{-1}a = e$. Por lo tanto, $a^{-1}H \in G/H$ y cumple: $aH \cdot a^{-1}H = (aa^{-1})H = eH$ y $a^{-1}H \cdot aH = (a^{-1}a)H = eH$

□

Gracias al resultado enunciado en el teorema anterior se puede definir el grupo cociente de la siguiente manera y bajo las siguientes condiciones:

Definición. Si H es un subgrupo normal de G, el conjunto de todas las clases laterales izquierdas (o derechas) de H en G, $G/H = \{aH | a \in G\}$, es un grupo y se denomina **grupo cociente** de G por H.

Ejemplo 6. Del ejemplo 5 se tiene que $nZ \triangleleft Z$, por lo tanto $Z/nZ = \{a + nZ | a \in Z\}$ es grupo.

3.3 El conjunto de las clases de congruencia es el grupo cociente Z/nZ

Sea C el conjunto de clases de congruencia módulo n, es decir, $C = \{[0]_n, [1]_n, ..., [n-1]_n\}$. De la Sección 3.1 se deduce que $C \subset Z/nZ$ ya que todos los conjuntos que son elementos de C pueden reescribirse como clases laterales:

- $[0]_n = \{0, 0 \pm n, 0 \pm 2n, 0 \pm 3n, ...\} = 0 + nZ$
- $[1]_n = \{1, 1 \pm n, 1 \pm 2n, 1 \pm 3n, ...\} = 1 + nZ$

  ...

- $[n-1]_n = \{n-1, (n-1) \pm n, (n-1) \pm 2n, (n-1) \pm 3n, ...\} = (n-1) + nZ$

Por ello, lo único que hace falta para probar que los conjuntos C y Z/nZ son el mismo conjunto, es demostrar que $0 + nZ$, $1 + nZ$, ..., $(n-1) + nZ$ son todos los elementos de Z/nZ, es decir, cualquier conjunto $a + nZ$ elemento de Z/nZ se puede reescribir como $r + nZ$ con r un entero $0 \leq r < n$. Dicho enunciado es válido según el siguiente lema.

Lema 2. *Sea $n \in Z$, entonces para cualquier $a \in Z$ se cumple $a + nZ = a \pmod{n} + nZ$*

Demostración. Sea $n \in Z$, considérese $a \in Z$. Para probar la igualdad es necesario probar las dos contenciones siguientes:

- $a + nZ \subseteq a \pmod{n} + nZ$
- y $a + nZ \supseteq a \pmod{n} + nZ$

($\subseteq$) Supóngase que $x \in a + nZ$, entonces $x = a + ny$ para algún $y \in Z$. Para a y n elementos de Z existen $q, r \in Z$ tal que $a = nq + r$ con $0 \leq r < n$, según el Teorema 1. Por lo tanto, $x = nq + r + ny = r + n(q + y)$. Como $r \equiv a \pmod{n}$ y $q + y \in Z$, entonces $x \in a \pmod{n} + nZ$.

($\supseteq$) Suponga ahora que $x \in a \pmod{n} + nZ$, entonces $x = a \pmod{n} + ny$ para algún $y \in Z$. Para a y n elementos de Z existen $q, r \in Z$ tal que $a = nq + r$ con $0 \leq r < n$, según el Teorema 1; de esto se tiene que $a \pmod{n} = r$, por lo tanto $a \pmod{n} = a - qn$, entonces $x = (a - qn) + ny = a + n(y - q)$ y como $y - 1 \in Z$ se tiene que $x \in a + nZ$.

□

De lo anterior se concluye que los elementos de Z/nZ son las *clases laterales*: $0 + nZ$, $1 + nZ$, ..., $(n - 1) + nZ$; pero estas clases laterales son las clases de congruencia módulo n, de modo que $Z/nZ = C$. Como Z/nZ es el grupo cociente, se tiene el siguiente corolario:

Corolario 1. *El conjunto de las clases de congruencia módulo n* $\{0 + nZ, 1 + nZ, ..., (n - 1) + nZ\}$ *es un grupo bajo la operación* $(a + nZ) + (b + nZ) = (a + b) + nZ$*, donde el representante de la clase resultante es el entero* $(a + b) \pmod{n}$.

En la sección 4.5 del presente trabajo se hará uso del grupo de clases de congruencia módulo 3: $Z/3Z = \{3Z, 3Z + 1, 3Z + 2\}$. Dicha sección trata sobre la mensuración *tempus perfectum cum prolatione imperfecta*; las reglas de transcripción de esta mensuración dejan claro que es necesario observar el número de semibreves que anteceden o siguen a una breve. Este número siempre es un entero y, por lo tanto, pertenece a una de las tres clases de congruencia módulo 3: $3Z$, $3Z + 1$ y $3Z + 2$. Después de un estudio cuidadoso de las reglas, es evidente que la transcripción de una sucesión de notas depende de a qué clase de congruencia pertenezca el número de semibreves en cuestión.

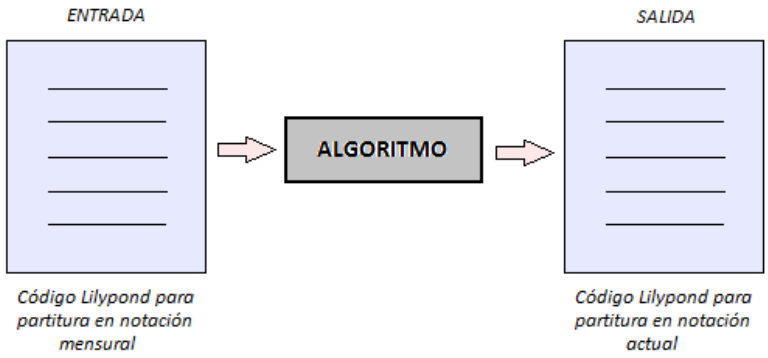
4 Desarrollo del algoritmo

El problema a resolver es la transcripción de una partitura mensural cualquiera a la notación actual. Este problema es tan extenso que debe ser dividido en subproblemas, tratando a los algoritmos de cada uno de estos subproblemas como módulos que, al ensamblarse de la forma apropiada, resuelven el problema principal. Para lograr esto, se inicia estudiando el problema general, para determinar los subproblemas en los que éste se puede dividir.

4.1 Problema general

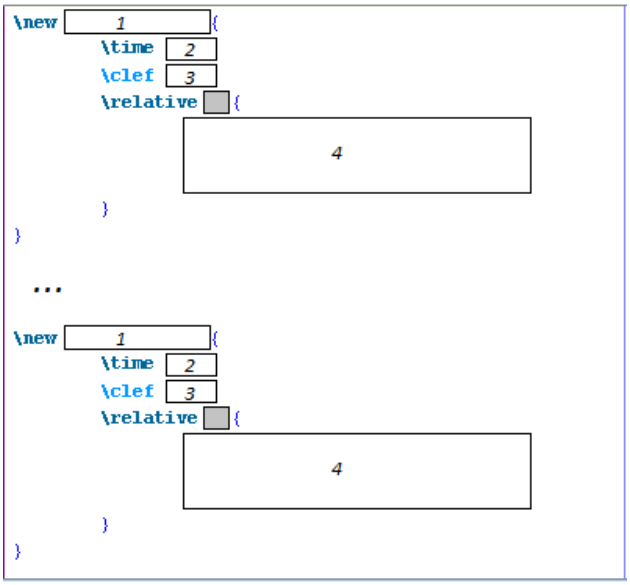
El objetivo del presente trabajo es desarrollar un algoritmo para resolver el problema de transcripción de partituras en notación mensural a notación actual. Para ello se seguirán los pasos señalados en la *sección 2.1.1*. Se iniciará analizando el problema (haciendo un dibujo de éste y distinguiendo sus entradas y salidas), para luego proceder a realizar un plan y posteriormente implementarlo como un algoritmo.

Figura 4.1: Dibujo del problema a resolver, con sus entradas y salidas



4.1.1. Estudio del problema Es claro que tanto la entrada como la salida son archivos de *Liylpond*, pero ¿cuál es la diferencia entre el código contenido en el archivo de entrada y en el archivo de salida? La estructura general del código de *LilyPond* para generar una partitura de más de una voz es:

Figura 4.2: Código LilyPond para partituras de muchas voces



Cuadro 4.1: Información que distingue el código de LilyPond para una partitura en notación mensural y una partitura en notación actual

Campo	**Tipo de información**	**Valores**	
		Entrada	**Salida**
1	VOICE (voz)	MensuralVoice	Voice
2	TIME (tiempo)	4/4	2/4
		3/2	3/4
		6/4	6/8
3	CLEF (clave)	“mensural-f” “petrucci-f” “petrucci-c5”	F
		“mensural-g” “petrucci-g” “petrucci-c1” “petrucci-c2”	G
		“petrucci-c3” “petrucci-c4”	“G_8”
4	NOTES (notas)	Caso Especial: Los cambios entre entrada y salida dependen de la mensuración (TIME de notación mensural)	

La Figura 4.2 es la estructura general del código en LilyPond para generar una partitura, los campos sin llenar que están identificados por los números del 1 al 4, contienen información que especifica: el tipo de voz (mensural o no), el tiempo, la clave y las notas. El campo señalado por la caja de fondo gris, contiene un parámetro que indica a qué altura inicia la voz (generalmente posee los valores de c’ para el do central y c” para el do agudo). Todas las palabras del código de entrada se conservan en el código de salida, exceptuando las palabras contenidas en los campos *1*, *2*, *3* y *4*. En el Cuadro 4.1 se indica qué tipo de información contiene cada uno de estos campos

y cuáles son sus posibles valores en el código de la notación mensural (entrada) y en código de la notación actual (salida).

4.1.2. Desarrollo de un plan Luego de estudiar el problema, se tiene claro que la entrada y salida del programa es un archivo LilyPond y lo que diferencia a ambos archivos es la información contenida en los 4 campos señalados en la Figura 4.2. El campo *4*, de ahora en adelante llamado *NOTES*, es un caso diferente de los otros tres campos, como muestra el Cuadro 4.1. El campo *NOTES* depende de la información del campo *TIME* para efectuar sus cambios, por el contrario, los otros tres campos, sólo dependen del valor de ellos mismos. Por lo tanto, es conveniente dividir el problema principal en dos subproblemas. En lugar de trabajar con todo el código de entrada del archivo LilyPond, éste se dividirá en dos: la parte del código que contiene exclusivamente notas, y el resto del código del archivo.

4.2 Subproblema 1: Transformación de la parte del código, del archivo de entrada, que no contiene notas

4.2.1. Estudio del problema La entrada de este subproblema consiste en una sucesión (arreglo) de oraciones del archivo *.ly de entrada que no contengan notas. La salida consiste en las oraciones transformadas según el Cuadro 4.1. La única diferencia entre entrada y salida, en este subproblema, son las palabras de los campos *VOICE*, *TIME* y *CLEF* del Cuadro 4.1.

4.2.2. Desarrollo de un plan Se debe evaluar si cada palabra de las oraciones pertenecientes al arreglo de entrada contiene alguna expresión de la columna *Entrada* del Cuadro 4.1; en caso afirmativo, se debe cambiar esta expresión por su correspondiente en notación actual (columna *Salida* del mismo Cuadro 4.1). Ese proceso de evaluar si la frase mensural está en una palabra, se repite con cada uno de los valores de la columna *Entrada* del Cuadro 4.1. Para simplificar la notación se puede usar una variable universal *MPHRASE* para designar un arreglo tal que $MPHRASE_i$ sea el i-ésimo elemento de dicha columna. Por lo tanto, el plan para abordar este subproblema es el siguiente:

1. Hacer un arreglo con todas las palabras de notación mensural que deben cambiar ($MPHRASE$) y hacer otro arreglo con las palabras correspondientes a las que deben cambiar para estar en notación actual ($APHRASE$).

2. Dividir el arreglo de entrada ($TEXT$) en líneas (*line*) y dividir estas líneas en palabras (*word*).

3. Evaluar si las palabras (*word*) contienen alguna expresión del arreglo *MPHRASE*, es decir si contiene algunas de las expresiones mensurales que deben cambiar; si sí, cambiar esa expresión por su correspondiente actual que se encuentra en el arreglo *APHRASE*.

4.2.3. Implementación del plan El signo "\" es un caracter de escape[1].

Algorithm 1 Cambio del código sin notas

```
procedure GeneralModifications(TEXT, new_file)
    MPHRASE = ["MensuralVoice", "4/4", "3/2", "6/4", ""mensural–f"", ""petrucci–f"",
""petrucci–c5"", ""mensural–g"", ""petrucci–g"", ""petrucci–c1"", ""petrucci–c2"",
""petrucci–c3"", ""petrucci–c4""]
    APHRASE = ["Voice", "2/4", "3/4", "6/8", "F", "F", "F", "G", "G", "G", "G", ""G_8"",
""G_8""]
    for line in TEXT do
        new_line = ""
        words_in_line ← list of the elements in line
        for word in words_in_line do
            for i from 0 until the length of the array MPHRASE do
                if MPHRASE_i in word then
                    Change MPHRASE_i in word to APHRASE_i
                end if
            end for
            Add word and a blank space to new_line
        end for
        Write new_line and "\n" in new_file
    end for                                          ▷ write the text in the output file
end procedure
```

4.3 Subproblema 2: Transformación de la parte del código, del archivo de entrada, que contiene únicamente las notas

4.3.1. Estudio del problema La entrada de este subproblema consiste en la sucesión de notas de una voz perteneciente al archivo *.ly de entrada. La salida consiste en las notas transformadas a notación actual.

4.3.1.1. ¿Cómo se representa una nota en LilyPond? Una nota en LilyPond está compuesta de dos partes fundamentales:

1. **Altura**

   Representada por las letras: *c*, *d*, *e*, *f*, *g*, *a* y *b* (*do*, *re*, *mi*, *fa*, *sol*, *la* y *si*, respectivamente). También puede ser un silencio representado por la letra *r*.

   Las alteraciones cromáticas de una nota también son consideradas parte del altura de la nota (los bemoles, representados por *is*, y los sostenidos, representados por *es*). Algunos ejemplos del código LilyPond para una nota con alteración cromática son: *ais* (la bemol), *ces* (do sostenido).

[1] Un caracter de escape ocasiona que el caracter siguiente posea una interpretación alternativa. Ejemplo: '\n' simboliza un cambio de línea.

2. **Duración o valor rítmico**

Es un texto o número que indica cuánto dura la nota, la siguiente tabla ilustra su uso.

Cuadro 4.2: Código LilyPond para las distintas figuras mensurales y actuales

Texto en LilyPond	Figura en notación				Texto en LilyPond	Figura en notación	
	mensural		actual			mensural	actual
\maxima		máxima	–	–	\maxima.		–
\longa		longa	–	–	\longa.		–
\breve		breve	–	–	\breve.		–
1		semibreve		redonda	1.		
2		mínima		blanca	2.		
4		semimínima		negra	4.		
8		fusa		corchea	8.		
16		semifusa		semicorchea	16.		
32	–	–		fusa	32.	–	
64	–	–		semifusa	64.	–	

Aparte, si se trata de una partitura mensural, puede haber ligaduras mensurales que unan algunas notas. LilyPond dispone de ciertos caracteres para representar dichas ligaduras. Los signos "\[" y "\]" agrupan un conjunto de notas en ligadura mensural. El "\[" se coloca antes de una nota e indica que con esta nota inicia la ligadura. En cambio el "\]" se coloca después de la nota e indica que con esa nota termina la ligadura. Un ejemplo de esto se muestra en la siguiente figura

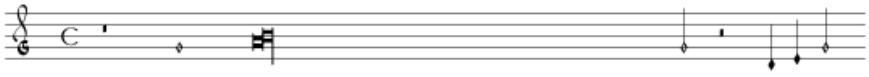
Figura 4.3: Ejemplo del código para generar ligaduras mensurales

ligadura_mensural.ly (%USERPROFILE%\Dropbox\TESIS\Tesis\)

```
1 \version "2.16.2"
2
3 \new MensuralVoice{
4         \clef "mensural-g"
5         \time 4/4
6         \relative c''{
7                 rl gl \[a\breve b\longa\] g2 r2 d4 e4 g2
8         }
9 }
```

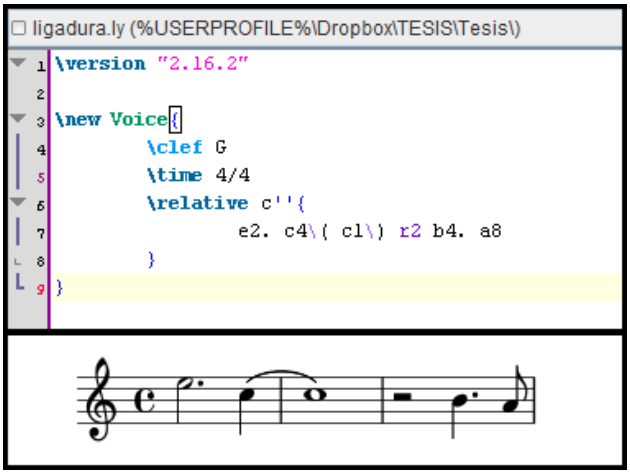
La ligadura empieza en *la* y termina en *si*, como se trata de una *breve* seguida de una *longa*, es una ligadura *cum propietate et cum perfectione* ascendente (ver Sección 8.2.3 del Anexo), por lo que tiene la forma:

Figura 4.4: Partitura mensural generada a partir del código mostrado en la Figura 4.3



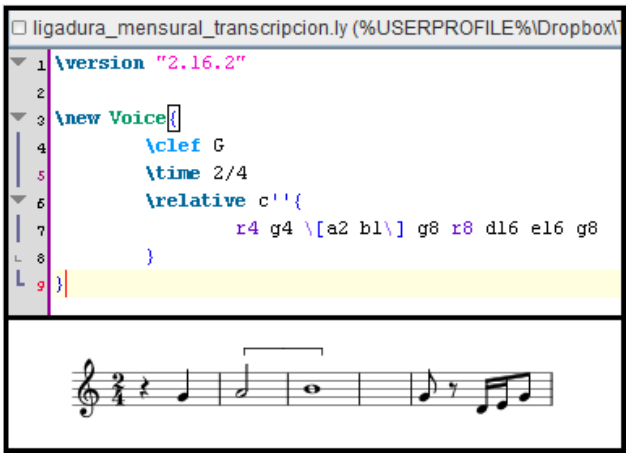
En la notación contemporánea, también existen ligaduras de otro tipo, éstas se representan por los símbolos "\(" y "\)". El símbolo "\(" se utiliza al final de una nota para indicar que es la primera nota de la ligadura. El símbolo "\)" se coloca también al final de una nota pero indica que ésta es la última nota de la ligadura. Un ejemplo de esto se muestra en la siguiente figura.

Figura 4.5: Partitura actual con ligadura



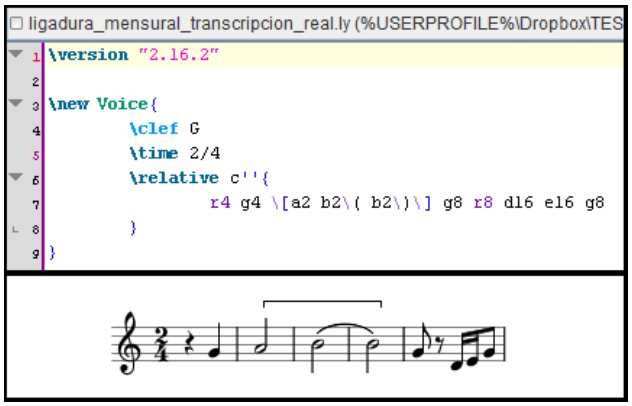
Este subproblema, el transformar las notas mensurales en actuales, se resume en cambiar la duración o valor rítmico de las notas, ya que su altura no se ve modificada (incluso las alteraciones cromáticas se conservan en la transcripción). También la ligadura mensural puede conservarse en el código de la partitura actual, ya que indica, mediante un corchete ubicado encima de un grupo de notas (ver Figura 4.6 y 4.7), cuáles notas solían estar unidas por una ligadura mensural en la partitura original. En resumen, lo que cambia en la transcripción es la duración de las notas, el altura y las ligaduras se conservan; para ejemplificar esto se utiliza el código de la Figura 4.3 (que genera la partitura de la Figura 4.4) y se transcribe en lo siguiente:

Figura 4.6: Transcripción de la partitura mensural mostrada en las figuras 4.3 y 4.4



Note el compás en blanco, la redonda debe ocupar dos compases, porque el compás es de 2 tiempos de negra (2/4) y la redonda ocupa 4 negras. LilyPond no puede subdividir de forma automática las figuras y ligarlas a través de la barra de compás; en cambio, pone toda la figura en el compás que inicia y deja el siguiente en blanco (o parcialmente en blanco). La transcripción real debería ser:

Figura 4.7: Transcripción adecuada de la partitura en las figuras 4.3 y 4.4



Como se dijo antes, LilyPond no puede subdividir figuras y ligarlas de forma automática al ver que éstas se “salen” del compás, sin embargo, este aspecto no es fundamental para el presente trabajo; éste se centra en cambiar el valor de las figuras mensurales al valor actual apropiado, se deja a un lado la subdivisión de las figuras a través de las barras de compás.

4.3.1.2. Proceso de transcripción según la mensuración Como ya se ha indicado, la transcripción de las notas de una partitura mensural depende del tipo de mensuración, los tres tipos pertinentes al trabajo son:

- *Tempus Imperfectum cum Prolatione Imperfecta*: todas las figuras poseen relación binaria entre sí, lo que hace que este caso sea parecido a la notación actual y es el caso más fácil de trabajar.
- *Tempus Perfectum cum Prolatione Imperfecta*: todas las notas poseen valor binario, son imperfectas; la única excepción es la breve, que posee un valor ternario, es decir, es perfecta. Sin embargo, hay condiciones que "imperfeccionan" a la breve.
- *Tempus Imperfectum cum Prolatione Perfecta*: todas las notas poseen valor binario, excepto la semibreve que es perfecta, posee un valor ternario; sin embargo, hay condiciones que la "imperfeccionan".

4.3.2. Desarrollo de un plan

El plan es simple, consiste en dividir el *subproblema 2* en tres subproblemas, uno por cada caso de mensuración a estudiar. Para cada subproblema se estudiará el método de transcripción actual y, a partir de éste, se deducirán reglas generales y se formularán éstas en un lenguaje formal adecuado para elaborar un algoritmo que permita la transcripción automática de las notas para cada uno de los casos de mensuración. Pero en todos los casos, como lo único que sufre cambio en la transcripción es la duración de la nota, se debe aislar ésta del resto de la descripción de la nota (es decir, aislarla de la altura y ligaduras mensurales) y utilizarla para realizar los cambios apropiados según la mensuración. Por lo tanto, aislar la duración del resto del texto descriptivo de la nota es un proceso que debe realizarse sin importar el tipo de mensuración.

4.4 Subproblema 3: transcripción para el caso *Tempus Imperfectum cum Prolatione Imperfecta*

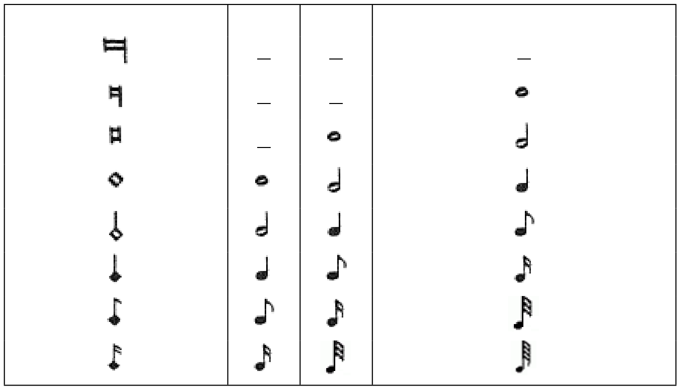
4.4.1. Estudio del problema

Este caso se parece a la notación actual debido a que todas las notas mantienen una relación binaria entre ellas. Por lo tanto, es posible transcribir una semibreve como una redonda, una mínima como una blanca, una semimínima como una negra, y así sucesivamente. Sin embargo, la transcripción depende del *factor de conversión*. El caso anterior usa el factor *1:1*, ya que cada figura mensural se transcribe a la figura actual que se le parece en su forma o trazo. Otros factores de conversión utilizados mucho por los musicólogos son el factor *1:2* y el factor *1:4*. Al usarse el factor *1:2* la semibreve, en lugar de transcribirse en una redonda, se transcribe en una blanca (la mitad de una redonda); y en el caso del factor *1:4* la semibreve se transcribe en una negra (la cuarta parte de una redonda). El Cuadro 4.3

incluye las figuras mensurales y sus transcripciones bajo los distintos factores de conversión para la mensuración *Tempus Imperfectum cum Prolatione Imperfecta*.

Cuadro 4.3: Transcripción de figuras para el caso *Tempus Imperfectum cum Prolatione Imperfecta* según los diferentes factores de conversión

Figura mensural	Transcripción actual según la razón		
	1:1	1:2	1:4



Con la información del Cuadro 4.3 y el Cuadro 4.2 es fácil deducir una forma de cambiar el valor rítmico de las notas del código LilyPond de la partitura mensural para generar el código LilyPond para la partitura actual. Los cambios apropiados para el código se muestran en el Cuadro 4.4.

Cuadro 4.4: Cambio del código LilyPond para la transcripción de las figuras mensurales a notación actual para el caso *Tempus Imperfectum cum Prolatione Imperfecta*

Código LilyPond para la figura mensural	Código LilyPond para la figura actual según el factor		
	1:1	1:2	1:4
\maxima	\maxima	\longa	\breve
\longa	\longa	\breve	1
\breve	\breve	1	2
1	1	2	4
2	2	4	8
4	4	8	16
8	8	16	32
16	16	32	64
\maxima.	\maxima.	\longa.	\breve.
\longa.	\longa.	\breve.	1.
\breve.	\breve.	1.	2.
1.	1.	2.	4.
2.	2.	4.	8.
4.	4.	8.	16.
8.	8.	16.	32.
16.	16.	32.	64.

El plan consiste en aislar la duración de la nota (igual que en todos los tipos de mensuración) y realizar los cambios señalados en el Cuadro 4.4.

4.4.2. Plan e implementación Lo primero que se debe hacer es solicitar el factor de conversión (*ratio*). Luego, se debe cambiar la figura de cada nota a su valor actual, para esto se debe llevar a cabo los siguientes pasos para cada nota:

1. Hallar la figura mensural de la nota
2. Identificar su figura contemporánea, según el Cuadro 4.4 con el factor de conversión correspondiente al ingresado.
3. Cambiar la figura mensural por la actual y escribirla en el código LilyPond de las notas.

Los pasos 1 y 2 no son procesos elementales, no pueden ser realizados por cualquiera, a menos que conozca las figuras mensurales y la semántica de LilyPond. Por lo tanto, es necesario definir un listado con la representación en LilyPond de cada una de las figuras mensurales, para que la persona que ejecuta el algoritmo evalúe si alguna de éstas se encuentra en la palabra que contiene la información de la nota. De la misma forma, se debe definir un listado con las correspondientes figuras actuales (recordar que este listado depende del factor de conversión), para que el ejecutante pueda realizar los pasos 1 y 2 sin problema. Los algoritmos 2 y 3 llevan a cabo estos procesos.

Algorithm 2 Solicitud del factor de conversión (*ratio*) y listado de figuras mensurales y contemporáneas según el factor

```
MVALUE = ["16.", "8.", "4.", "2.", "1.", "\\breve.", "\\longa.", "\\maxima.", "16", "8", "4", "2", "1", "\\breve", "\\longa", "\\maxima"]
AVALUE12 = ["32.", "16.", "8.", "4.", "2.", "1.", "\\breve.", "\\longa.", "32", "16", "8", "4", "2", "1", "\\breve", "\\longa"]
AVALUE14 = ["64.", "32.", "16.", "8.", "4.", "2.", "1.", "\\breve.", "64", "32", "16", "8", "4", "2", "1", "\\breve"]
repeatAsk for ratio
until ratio = "1:1" or ratio = "1:2" or ratio = "1:4"
if ratio = "1:1" then
    AVALUE ← MVALUE
else if ratio = "1:2" then
    AVALUE ← AVALUE12
else if ratio = "1:4" then
    AVALUE ← AVALUE14
end if
```

Y luego para cada nota (*mnote*)

Algorithm 3 Encontrar la figura mensural y su correspondiente actual de cada nota *mnote* de la sucesión de notas de entrada

```
for i running from 0 until the length of the list MVALUE do
    if MVALUE_i in mnote then
        mvalue ← MVALUE_i
        avalue ← AVALUE_i
    end if
end for
```

Así se hallan los valores mensurales y actuales de las notas (pasos 1 y 2). El Algoritmo 2 utiliza un *ciclo until* que permite solicitar el factor de conversión *ratio* y repetir esta solicitud hasta que el ingreso sea válido. Finalmente, toca el último paso, cambiar el valor de la figura mensural a su valor actual. Se ubica el valor de la nueva figura (actual) en el espacio ocupado por el valor de la antigua figura (mensural). Por lo tanto, el algoritmo para llevar a cabo el paso 3 es:

Algorithm 4 Cambia el valor rítmico de una nota

function ChangeNoteValue(*note*, *mvalue*, *avalue*)
  $first \leftarrow$ characters of *note* before *mvalue*
  last $\leftarrow$ characters of *note* after the *mvalue*
  new_note $\leftarrow$ join the strings $first$, *avalue* and *last*, in that order
  return *new_note*
end function

Como el proceso para cambiar la figura de una nota de mensural a actual sera utilizado constantemente para los diferentes tipos de mensuración (sin importar de qué valor mensural a qué valor actual se esté cambiando), entonces este algoritmo es más bien una función que tomará como argumentos a la nota *mnote*, al valor mensural de la nota *mvalue* y al valor actual correspondiente *avalue*, y devolverá la nota transformada en actual.

Al unir el código de los algoritmos 2 y 3 para todas las notas, y hacer uso de la función *ChangeNoteValues* del Algoritmo 4, se obtiene el Algoritmo 5, que toma una sucesión de notas mensurales ($MNOTES$) y transcribe cada una de ellas en notación actual, según la mensuración *tempus imperfectum cum prolatione imperfecta*.

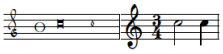
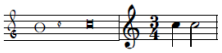
Algorithm 5 Cambio de la secuencia de notas mensurales *MNOTES* a la sucesión de notas actuales para el caso *tempus imperfectum cum prolatione imperfecta*

procedure TranscripteII(MNOTES, *new_file*)
  MVALUE = [“16.”, “8.”, “4.”, “2.”, “1.”, “\\breve.”, “\\longa.”, “\\maxima.”, “16”, “8”, “4”, “2”, “1”, “\\breve”, “\\longa”, “\\maxima”]
  AVALUE12 = [“32.”, “16.”, “8.”, “4.”, “2.”, “1.”, “\\breve.”, “\\longa.”, “32”, “16”, “8”, “4”, “2”, “1”, “\\breve”, “\\longa”]
  AVALUE14 = [“64.”, “32.”, “16.”, “8.”, “4.”, “2.”, “1.”, “\\breve.”, “64”, “32”, “16”, “8”, “4”, “2”, “1”, “\\breve”]
  repeatAsk for ratio
  until ratio = “1:1” or ratio = “1:2” or ratio = “1:4”
  if ratio = “1:1” **then**
    AVALUE $\leftarrow$ MVALUE
  else if ratio = “1:2” **then**
    AVALUE $\leftarrow$ AVALUE12
  else if ratio = “1:4” **then**
    AVALUE $\leftarrow$ AVALUE14
  end if

```
    for mnote in MNOTES do
        for i running from 0 until the length of the list MVALUE do
            if MVALUE_i in mnote then
                mvalue ← MVALUE_i
                avalue ← AVALUE_i
            end if
        end for
        anote ← ChangeNoteValue(mnote, mvalue, avalue)
        write anote in new_file
    end for                                          ▷ write the notes in the output file
end procedure
```

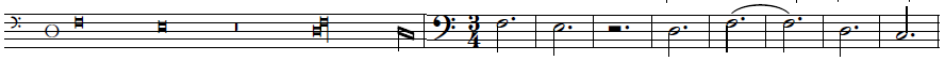
4.5 Subproblema 4: transcripción para el caso *Tempus Perfectum cum Prolatione Imperfecta*

4.5.1. Estudio del problema En este tipo de mensuración todas las notas mantienen una relación binaria entre sí, es decir, son *imperfectas*; con excepción de la breve. La breve es *perfecta*, equivale a tres semibreves. Sin embargo, hay condiciones que pueden "imperfeccionar" a la breve, depende de qué figuras la rodean. Hay dos métodos principales de imperfección: *imperfectio a parte post* (*a.p.p.*), la imperfección causada por la siguiente nota, e *imperfectio a parte ante* (*a.p.a.*), la imperfección causada por la nota anterior; como se muestra a continuación:

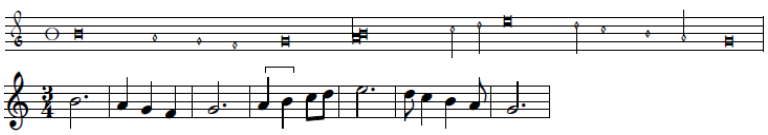
- Una imperfección a.p.p. se ejemplifica como 
- Una imperfección a.p.a. se da cuando 

Según Apel (2010) las siguientes reglas determinan si una breve es perfecta o imperfecta según su contexto. Es necesario señalar que cuando Apel se refiere a una cantidad n de semibreves que siguen o preceden a una breve, no quiere decir que haya exactamente n figuras de semibreve, si no que hay cualquier grupo de figuras, cada una menor o igual en valor a una semibreve (semibreves, mínimas, semimínimas, fusas o semifusas), cuya duración conjunta es equivalente a n semibreves. Esto es fácil de notar al observar los ejemplos de las reglas.

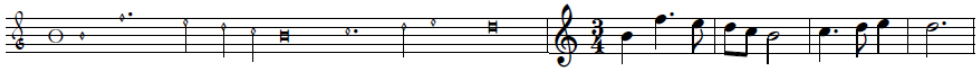
1. Una breve es perfecta si es seguida por otra breve o por un silencio de breve.



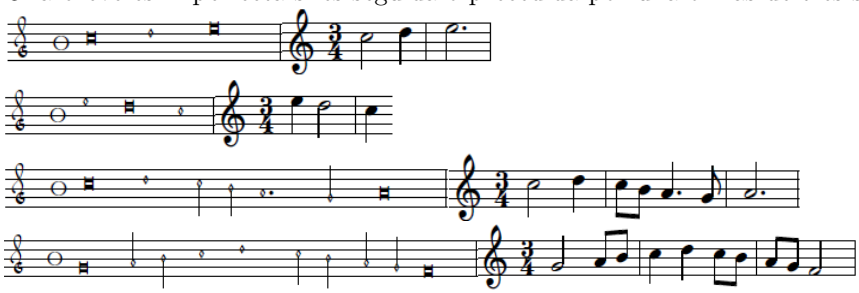
2. Una breve es perfecta si es seguida por dos (a veces hay excepciones, ver la primera *Nota* de la *Regla 6*) o tres semibreves.



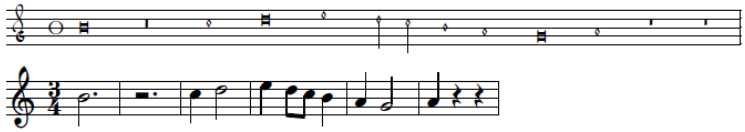
Nota: en ocasiones una breve seguida por tres semibreves puede ser imperfecta, por *imperfección a.p.a.*



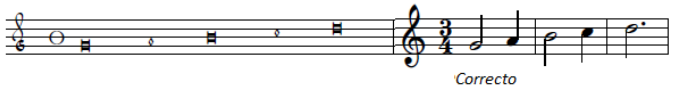
3. Una breve es imperfecta si es seguida o precedida por una o más de tres semibreves.

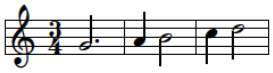


Nota: Aunque los ejemplos mostrados para esta regla muestran un agrupamiento normal para 1, 4 y 5 semibreves entre dos breves, la agrupación puede variar ocasionalmente dependiendo del contexto, es decir, cuando la primera breve es imperfeccionada por *a.p.a.*. En el ejemplo siguiente el grupo de 4 semibreves se divide en 3 + 1 (en lugar de 1 + 3).

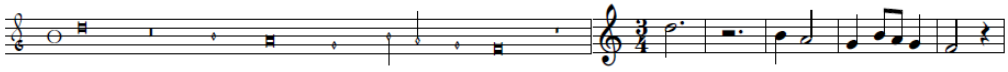


4. Si ambas imperfecciones *a.p.p.* y *a.p.a.* son admisibles, la primera toma precedencia.

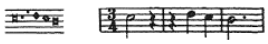




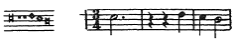
5. Un silencio de breve no puede ser imperfeccionado jamás, pero un silencio de semibreve es capaz de imperfeccionar una nota.



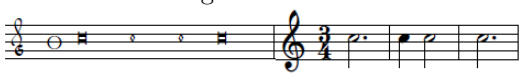
Nota: Si una breve es seguida por dos silencios de semibreve, se hace una importante distinción. Cuando dos silencios se encuentran en diferentes líneas del pentagrama, el primero imperfecciona a la primera breve, mientras que el segundo pertenece a la siguiente perfección.

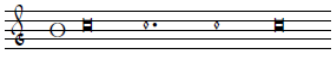


Si, por el contrario, ambos silencios aparecen en la misma línea del pentagrama, ambos pertenecen a la misma perfección, así que la primera breve permanece perfecta.

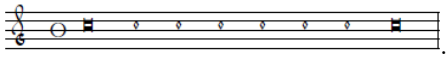
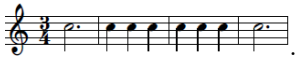
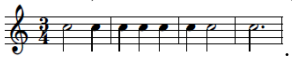


6. *Principio de alteración*: si dos semibreves se encuentran entre dos breves, se duplica la duración de la segunda semibreve.



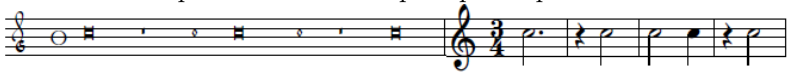
Nota: Sin embargo, en ocasiones la combinación "*breve, semibreve, semibreve, breve*" requiere de una interpretación distinta de la antes mencionada, llamada *imperfección*. En ésta última, ambas breves son imperfectas. Según la teoría, esta interpretación debía ser denotada por el *punctus divisionis* de la siguiente manera  , separando así ambas partes como grupos perfectos (de 3 tiempos). Sin embargo, no son inusuales los ejemplos que necesitan de imperfección y que no contienen *punctus divisionis*.

Nota: La imperfección es imposible en combinaciones "*breve, semibreve, semibreve, breve, breve*" por contradecir la regla 1. Y la alteración es imposible en combinaciones donde la segunda semibreve sea remplazada por notas de menor valor que juntas equivalgan a la semibreve, por ejemplo: "*breve, semibreve, mínima, mínima, breve*".

Nota: El principio de alteración también se utiliza cuando hay seis semibreves entre dos breves  . Al principio esta combinación podría sugerir formar dos grupos de tres semibreves cada uno y, aplicando la regla 2, mantener la breve inicial perfecta  . Sin embargo, al aplicar la regla 3, la primera semibreve imperfecciona a la breve inicial, y así el grupo de 5 semibreves restantes se divide en un grupo de 3 y otro de 2, éste último exige la alteración de la última semibreve; de este modo se obtiene  . Sin embargo, aunque la última versión es la que prevalece, no descarta a la primera, ya que hay casos donde la alteración es imposible, como cuando se remplaza la última semibreve por valores menores o por un silencio de semibreve, o que la primera breve haya sufrido una *imperfección a.p.a.*

7. Es seguro que el principio de alteración actúa cuando las dos semibreves (entre las breves) están escritas con *ligadura c.o.p.* (ver Sección 8.2.3 del Anexo).

8. Un silencio no puede ser alterado pero puede producir la alteración de una nota.

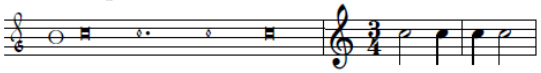


Nota: La imposibilidad de alterar un silencio implica que la combinación "*breve, semibreve, silencio de semibreve, breve*" debe obedecer al *principio de imperfección*.

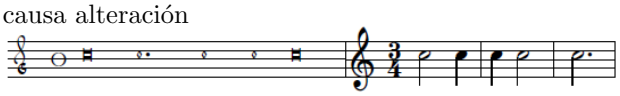
9. El *punctus divisionis* divide conjuntos de notas en subconjuntos perfectos (equivalentes en duración a tres semibreves) y puede así tener diferentes efectos, produciendo imperfecciones, alteraciones y perfecciones, adquiriendo así otros nombres como *punctus imperfectionis*, *punc-*

tus alterationis y *punctus perfectionis*, respectivamente. Ejemplos de estos distintos efectos provocados por el mismo *puctus divisionis* son:

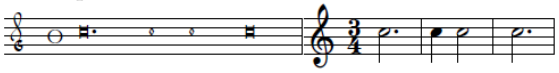
- causa imperfección



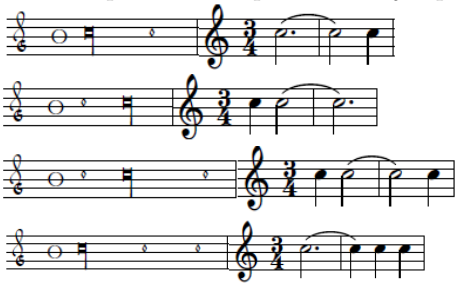
- causa alteración



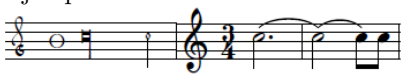
- causa perfección



10. La imperfección manejada hasta ahorita es *imperfectio ad totum*, imperfección de toda la nota, también existe la *imperfectio ad partem*, la imperfección de una parte de la nota. Esto ocurre cuando se imperfecciona una longa, ya que ésta está constituida por dos breves y puede que alguna de ellas (alguna de sus partes) sea imperfeccionada. Por lo tanto, una longa puede equivaler a 4, 5 o 6 semibreves, dependiendo de si se imperfecciona ambas breves, sólo una, o ambas permanecen perfectas. Ejemplos de esto son:



Los ejemplos anteriores presentan *imperfectio ad partem propinquam*, es decir, imperfección causada por una nota dos grados abajo de la nota imperfeccionada. También existe la *imperfectio ad partem remotam*, causada por una nota tres grados bajo la imperfeccionada. Ejemplo



Sin embargo, la *imperfectio ad partem remotam* es tan poco usual que no será tratada en el presente trabajo.

4.5.2. Estudio de las reglas de transformación Previo al análisis de estas reglas, hay algunas aclaraciones sobre la terminología a utilizar. De aquí en adelante se hará uso de las expresiones:

- *formar grupos perfectos* o *formar perfecciones*, esto quiere decir formar grupos de notas cuya duración sea equivalente a tres semibreves.

- *observar la cantidad de semibreves*, esta expresión se utilizará generalmente en el siguiente contexto "observar la cantidad de semibreves entre dos breves". Se refiere a tomar el conjunto de todas las figuras que se encuentran entre dos breves y determinar la cantidad de semibreves cuya duración es equivalente a este grupo de figuras; en otras palabras, es determinar la cantidad de tiempos de semibreve que caben en esa sucesión de figuras, cada una menor o igual en valor a una semibreve.

- *se forman grupos de tres semibreves*, esto significa que las figuras se agrupan de tal forma que la duración de cada uno de esos grupos sea equivalente a tres semibreves. Esto se realiza con las figuras que se encuentran entre breves, es decir, con figuras cuyo valor individual sea menor o igual a una semibreve. Después de formar las agrupaciones de tres semibreves, se utiliza la expresión *sobran n semibreves*, esto significa que las figuras restantes son equivalentes en duración a n semibreves.

Después de haber definido claramente estas expresiones, se procede a analizar las reglas de la Sección 4.5.1. Todas las reglas contemplan la cantidad de semibreves que preceden o siguen a una breve, ya que dicha cantidad ejerce un efecto sobre las breves que están alrededor con el fin de formar perfecciones. Dado un conjunto de notas el proceso de transcripción se realiza de izquierda a derecha. Es lógico que la transcripción de este tipo de mensuración no se realice nota por nota (como en el caso del tempus imperfectum cum prolatione imperfecta) si no mediante sucesiones de notas, ya que el valor de una nota (específicamente de la breve) no depende únicamente de ella misma si no también de su contexto, de las notas que la rodean (específicamente de las semibreves que la preceden o siguen).

De las reglas de esta mensuración se infiere que hay sólo dos tipos de sucesiones de transcripción: aquéllas que inician y terminan en breve, y aquéllas que inician en semibreve (o una nota de menor duración) y terminan en breve. Es evidente que toda sucesión de notas a transcribir debe tener como referencia una breve, la cuál debe estar al inicio o al final debido a que su valor depende de la cantidad de semibreves que la siguen o la preceden, respectivamente. Si se toma una breve como el punto de inicio de una sucesión de notas a transcribir (como es el caso del inicio de muchas partituras) se debe contar cuántas semibreves la siguen, hasta llegar a la siguiente breve donde se detiene la cuenta. De este modo la sucesión inicia y termina en una breve. Nótese que el número de semibreves entre breves afecta el valor de la primera breve (la mantiene perfecta o la imperfecciona de la forma a.p.p.) y puede o no afectar el valor de la última (ya que puede no influir en su duración o puede imperfeccionarla de la forma a.p.a.). Si la última breve sufre una imperfección a.p.a. la siguiente sucesión de notas a transcribir inicia con la nota que le sigue a esta breve, la cuál debe ser forzosamente una semibreve (o una nota de menor duración) ya que de otro modo se entraría en contradicción con la *Regla 1.* Por lo tanto, también hay sucesiones de notas a transcribir que inician con una semibreve (o una nota de menor valor), ya sea por que

la breve anterior sufrió una imperfección a.p.a. o porque la primera nota de la partitura es una semibreve; en cualquier caso, se debe encontrar la primera breve que se encuentre después de este punto de inicio ya que el número de semibreves que la anteceden determinan su valor (la breve de referencia se encuentra al final). Así se justifican los dos tipos de sucesiones que se utilizarán en el presente trabajo para la transcripción partituras en *tempus perfectum cum prolatione imperfecta*. Dicha transcripción inicia con la sucesión que parte de la primera nota de la partitura y termina en la primera breve que le sigue, y luego avanza progresivamente transcribiendo sucesiones de notas cuyo punto de inicio es la nota siguiente a la última nota transcrita de la sucesión anterior.

Así queda explicado el proceso de transcripción de una partitura en *tempus perfectum cum prolatione imperfecta*. En resumen, dicho proceso se realiza de izquierda a derecha a partir de sucesiones de notas que pueden ser de dos tipos: aquéllas que inician y terminan en breve, o aquéllas que inician en una semibreve (o nota de menor valor) y terminan en una breve. Otro factor que se debe tomar en cuenta es la presencia de algún punto de división dentro de la sucesión de notas a transcribir, dicho punto divide la sucesión en dos conjuntos perfectos (cada uno equivalente a agrupaciones de 3 semibreves). Las reglas se centran en estos dos tipos de sucesiones, uno de los cuáles cuenta con un caso especial (la presencia del punto de división). En las siguientes tres subsecciones se analizan estos tres casos individualmente.

Como ya se ha enfatizado muchísimas veces, las reglas anteriores observan la cantidad de semibreves que siguen o preceden a una breve y lo que pretenden es agruparlas formando grupos perfectos, grupos equivalentes a tres semibreves. Si esto no es posible, se realizan modificaciones en las figuras mediante los procesos de: *alteración*, *imperfección a.p.p.* e *imperfección a.p.a.*; con el fin de obtener agrupaciones perfectas. El tipo de modificación apropiada depende del contexto: de la cantidad de semibreves involucradas y, para ser más específicos, de a cuál de las 3 clases de congruencia del grupo $Z/3Z = \{3Z, 3Z+1, 3Z+2\}$ pertenece dicha cantidad.

4.5.2.1. Sucesión de notas que inicia y termina en breves, sin punctus divisionis Sea m el número de semibreves equivalentes en duración a las figuras encerradas entre las dos breves. Hay tres condiciones que m puede cumplir: $m \in 3Z+1$, $m \in 3Z+2$ y $m \in 3Z$; la modificación a realizar en la transcripción (*alteración*, *imperfección a.p.p.* e *imperfección a.p.a.*) depende de cuál de estas tres condiciones sea satisfecha por m.

1. Si $m \in 3Z+1$ entonces *imperfección a.p.p.*

   Cuando las figuras entre dos breves se reúnen en grupos equivalentes en duración a 3 semibreves y sobra una semibreve, ésta ocasiona una *imperfección a.p.p.* según la *Regla 3*; de este modo, la primera breve equivale a 2 semibreves, y junto a la semibreve restante forma una perfección. (Nótese que es la primera breve la imperfeccionada y no la última, ya que de otra manera se contradice la *Regla 4*.)

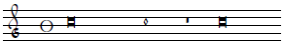
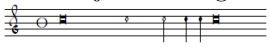
2. Si $m \in 3Z + 2$

   Cuando, luego de reunir las figuras en grupos de 3 semibreves, sobran 2 semibreves se pueden realizar dos tipos de modificación:

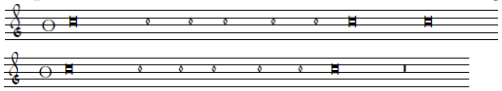
   (a) *Alteración.* Consiste en duplicar el valor de la última semibreve de la sucesión. La otra semibreve restante y la alterada forman juntas una perfección.

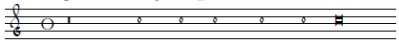
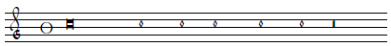
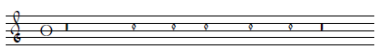
   (b) *Imperfección a.p.p* e *imperfección a.p.a.* Cada una de las 2 semibreves restantes imperfecciona a una breve; de esta manera la primera breve (equivalente ahora a 2 semibreves) y una de las semibreves restantes, forman una perfección; lo mismo ocurre con la última breve y la otra semibreve.

   Por lo general, la opción *(a)* se prefiere cuando hay exactamente 2 semibreves (o un grupo de figuras equivalentes en duración) entre dos breves, como lo indica la *Regla 2* y la *Regla 6*. Y la opción *(b)*, es la predilecta cuando hay más de 3 semibreves (cuando son 5, 8, 11, etc.) como indica la *Regla 3*. Sin embargo, aunque hay casos donde se prefiere una modificación a la otra, nunca se descarta la menos predilecta ya que hay circunstancias donde la predilecta no puede ser llevada a cabo. Las circunstancias que eliminan la posibilidad de aplicar una *alteración* son:

   - Que la última semibreve sea un silencio. En este caso la alteración contradice la *Regla 8*. Ejemplo: 
   - Que la última semibreve sea sustituida por un conjunto de figuras de menor valor (ver la segunda *Nota* de la *Regla 6*). Ejemplo: 

   Las circunstancias que eliminan la posibilidad de imperfección son:

   - Que la última breve de la sucesión anteceda a otra breve, o a un silencio de breve. La imperfección en este caso contradice la *Regla 1*. Ejemplos:
     

   - Que la primera o la última breve sean silencios. En este caso la imperfección contradice la *Regla 5*. Ejemplos:
     
     
     

3. Si $m \in 3Z$

   Cuando las figuras entre dos breves se juntan en grupos de tres semibreves y no sobra ninguna, hay dos escenarios posibles:

(a) Que el número de semibreves entre las dos breves sea menor o igual a 3, es decir, que no haya ninguan figura entre las dos breves o que hayan exactamente 3 semibreves (o su equivalente en otras figuras); en este caso la breve se mantiene perfecta, como indican la *Regla 1* y la *Regla 2*, respectivamente.

(b) Que el número de semibreves sea mayor a 3 (cuando son 6, 9, 12, etc.). En este caso hay dos posibilidades:

i. Mantener la breve perfecta como en 3a.

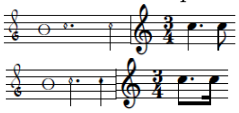
ii. Realizar *imperfección a.p.p.* y *alteración*. Esto permite también formar agrupaciones perfectas.

La modificación *(ii)* es la predilecta en este caso, como lo indica la *Regla 3*, pero no descarta el uso de la modificación *(i)*, ya que hay condiciones donde la *imperfección a.p.p.* y la *alteración* no pueden ser llevadas a cabo:

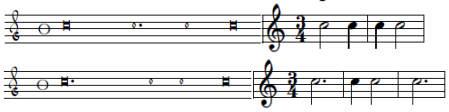
- cuando la primera breve es un silencio (imposibilita la imperfección a.p.p.)
- cuando la última semibreve es un silencio (imposibilita la alteración)
- o cuando la última semibreve está sustituida por valores más pequeños (imposibilita la alteración)

Finalmente, hay una circunstancia que demanda el uso de la *alteración*: cuando las dos últimas semibreves de la sucesión se encuentran ligadas (esta ligadura es conocida como ligadura c.o.p., como se muestra en la Sección 8.2.3).

4.5.2.2. Sucesión de notas que inicia y termina en breves, con punctus divisionis Todo lo anterior sucede cuando no hay *punctus divisionis* entre las dos breves. El trazo del *punctus divisionis* no se distingue del trazo del *puntillo*, la distinción entre estos dos radica en el contexto. El *puntillo* le agrega a una figura la mitad de su valor; por lo tanto, cuando una nota tiene *puntillo* va seguida de una figura que completa la otra mitad. Por ejemplo:



En cambio, el *punctus divisionis* separa el conjunto de semibreves para formar dos conjuntos perfectos. En esta separación, los dos conjuntos poseen un valor entero de semibreves; de lo contrario, si la cantidad de semibreves entre las breves posee un valor fraccionario, no hay manera de formar perfecciones (aún imperfeccionando las breves o alterando la última semibreve). Ejemplos de sucesiones de notas con *punctus divisionis* son:



En LilyPond el *punctus divisionis* tampoco se diferencia del puntillo. Si todos los puntos del código LilyPond fuesen considerados como *puntillos*, la suma de los valores (en tiempos de semibreve) de las figuras entre las dos breves daría un resultado que, si no es entero, implica que uno de los *"puntillos"* es en realidad un *punctus divisionis*, ya que no hay una figura que "complete" la fracción generada por el puntillo.

Para que los conjuntos a la derecha e izquierda del punto de división sean perfectos, se utiliza la misma idea de formar grupos ternarios de semibreves e identificar cuántas quedan fuera de estos grupos (pueden ser 0, 1 o 2). Esto se hace para cada uno de los dos conjuntos separados por el punto de división. Este proceso se realiza también en la Sección 4.5.2.1, la diferencia con respecto a lo discutido en esa sección es que no hay una breve a cada extremo de la sucesión, sólo hay una al principio (para el conjunto anterior al punto de división) o una al final (para el conjunto posterior al punto de división). Las modificaciones a realizar dependen del número de semibreves que queden fuera de los grupos ternarios: se mantiene la breve perfecta (si el número es 0), o se imperfecciona (si es 1), o se altera la última semibreve (si es 2). Estas modificaciones son obvias al considerarse que hay una única breve en la sucesión (al inicio o al final).

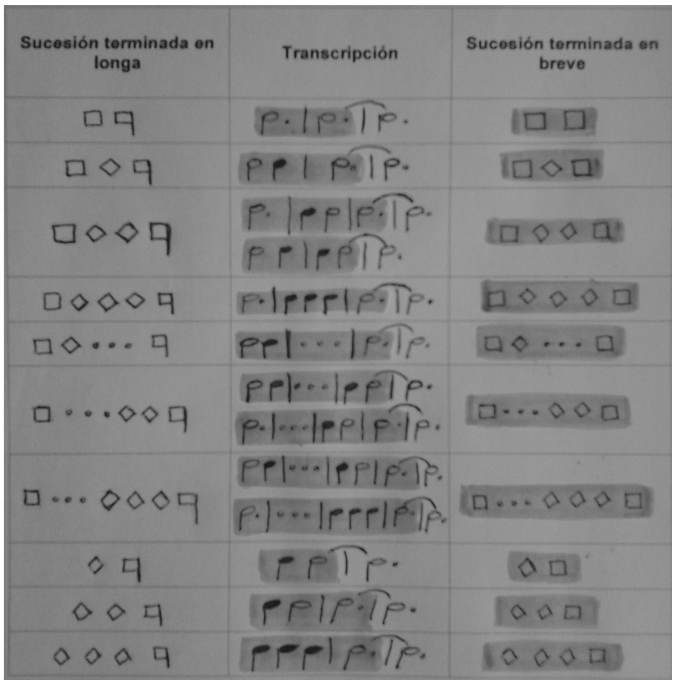
4.5.2.3. Sucesión de notas que inicia en una semibreve y termina con una breve Hay dos motivos por los cuáles es importante tomar en cuenta este tipo de sucesiones. El primero es porque muchas partituras inician con una nota de menor duración que una breve. Otro es porque al tomar sucesiones de notas que inician y terminan con breve (como se hace en la Sección 4.5.2.1) hay casos donde ocurre una *imperfección a.p.a.*, por lo que la última nota transcrita es la última breve de la sucesión, de modo que la siguiente sucesión a transcribir no empieza en esa breve si no en la siguiente nota, y ésta debe ser menor en valor a una breve (ya que de no ser así, se contradice la *Regla 1*). Por lo tanto, sí existen sucesiones de notas a transcribir que inician con una semibreve (o un grupo de figuras equivalente). Dichas sucesiones deben tomar todas las notas siguientes hasta la primera breve que aparece después de la nota inicial, porque las semibreves son importantes en la transcripción por su efecto sobre las breves a su alrededor. Las modificaciones en este caso son obvias, ya que sólo hay una breve en la sucesión, la final. Luego de agrupar las semibreves anteriores a la breve en conjuntos ternarios, el número de semibreves restantes determina la modificación a realizar: la breve se mantiene perfecta (si el número es 0), o se imperfecciona (si es 1), o se altera la última semibreve (si el número es 2).

4.5.2.4. Casos no considerados aún El comportamiento de estos tres tipos de sucesiones engloba la mayor parte de las reglas, con excepción de la *Regla 10* y la *Nota* de la *Regla 5*. Esta última indica que si dos silencios de semibreve que siguen a una breve están ubicados en diferente línea del pentagrama, entonces cada uno pertenece a distinta perfección, y si están en la misma línea pertenecen a la misma perfección. El modelo algorítmico no pondrá atención a la ubicación de los silencios en las líneas del pentagrama, pero si una persona quiere

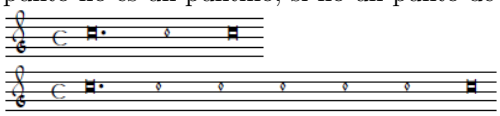
indicar que dos silencios pertenecen a distintas perfecciones puede realizarlo colocando un *punctus divisionis* entre ambos (en lugar de ubicarlos en diferentes líneas). De este modo se evita complicar el modelo innecesariamente, ya que se puede obtener el mismo resultado utilizando aspectos ya contemplados en el modelo (como el *punto de división*).

Con respecto a la *Regla 10*, luego del estudio de muchas partituras con mensuración *tempus perfectum cum prolatione imperfecta*, se observó que en todas ellas la *longa* se utiliza siempre al final de cada una de las voces de la partitura, y jamás aparece al inicio o en medio de la melodía. Por lo tanto, y para mantener la simplicidad del modelo algorítmico, éste se restringirá a partituras donde las longas sean utilizadas al final de cada voz, y que no se encuentren en ninguna otra ubicación. Esto mantendrá el modelo simple y aún así abarcará la mayoría de los casos, que es lo que un modelo matemático pretende. Por consiguiente, falta considerar un tipo más de sucesión de notas para transcribir: el que termina en longa, empezando o no en una breve. Cualquiera que sea el caso, también debe tomarse en cuenta cuántas agrupaciones de 3 semibreves caben en el conjunto de figuras menores a una breve, para determinar cuántas semibreves quedan fuera de estas agrupaciones (0, 1 o 2) y así tomar una decisión acerca del tipo de modificación que debe sufrir la longa (y la breve, si en caso la sucesión inicia con una). Por la *Regla 10* se sabe que una longa puede poseer un valor equivalente a 4, 5 o 6 semibreves, dependiendo del contexto; véase la Figura 4.8 para ver los distintos casos que pueden darse en esta sucesión y su correspondiente transcripción.

Figura 4.8: Sucesiones terminadas en longa, su transcripción y la comparación de ésta última con la transcripción de la misma sucesión terminada en breve. (Los puntos suspensivos denotan agrupaciones equivalentes a 3 semibreves.)



Finalmente, aunque en la Sección 4.5.2.2 ya se ha tomado en cuenta las sucesiones que poseen *punto de división*, falta un caso importante. En dicha sección se considera al *punctus divisionis* cuando éste se encuentra dentro de las semibreves de la sucesión, donde actúa como un punto de imperfección o de alteración. Pero falta considerar cuando este punto actúa como *punctus perfectionis* al colocarse después de la primera breve de la sucesión. En este tipo de circunstancias, por el tipo de mensuración de la breve (perfecta, ya posee un valor ternario), es obvio que ese punto no es un puntillo, si no un punto de división. Ejemplos de esto son:



(o cualquier otro caso donde, sin el uso del punto de división, se presenta una *imperfección a.p.p.*). Es fácil notar que este caso no ha sido tomado en cuenta en la Sección 4.5.2.2 ya que el punto está ubicado en la breve y ademas éste no genera un valor no entero; en dicha sección la manera de determinar la presencia de un *punto de división* es mediante el valor no entero de la duración, en tiempos de semibreve, de las figuras entre las breves.

4.5.3. Formalización de las reglas de transcripción De acuerdo a la discusión anterior, donde se analizaron las reglas de transcripción, se puede proceder a formalizar dichas reglas para su posterior implementación algorítmica. Es fácil notar lo siguiente:

- la transcripción se lleva a cabo por sucesiones de notas y la última nota transcrita en la sucesión determina la siguiente sucesión de notas a transcribir, iniciando después de esa última nota transcrita
- estas sucesiones empiezan en una determinada nota y terminan con la primera breve que aparece luego de dicha nota
- se observa cuántas semibreves hay en la sucesión, sin tomar en cuenta a las breves.
- se agrupan estas semibreves en conjuntos de 3, luego se evalúa cuántas semibreves quedaron fuera de estos conjuntos (0, 1 o 2). Esto es lo mismo que tomar el número de tiempos de semibreve en la sucesión (sin tomar en cuenta a las breves) y obtener el residuo de dividir el número entre 3. Este residuo (0, 1 o 2) se utiliza para tomar una decisión acerca de la modificación que debe sufrir la breve o la última semibreve de la sucesión (modificaciones como la imperfección o la alteración)
- el tipo de modificación a seguir también depende de si el primer elemento de la sucesión a transcribir es una breve o una nota de menor duración, y de si dentro de la sucesión hay puntos de división
- las notas menores en duración que las breves, se transcriben como lo indica el Cuadro 4.5

Cuadro 4.5: Cambio del código LilyPond para la transcripción de las figuras mensurales a notación actual para el caso *Tempus Perfectum cum Prolatione Imperfecta*, cuando la figura es menor en duración a una breve

Código LilyPond para la figura mensural	Código LilyPond para la figura actual
1	4
2	8
4	16
8	32
16	64
1.	4.
2.	8.
4.	16.
8.	32.
16.	64.

Con base a este resumen del análisis de las reglas de transcripción para el *tempus perfectum cum prolatione imperfecta* se procede a la formalización de estas reglas, iniciando con la definición de las sucesiones de transcripción ν.

Definición. Sea n_j la j–ésima nota de una partitura, entonces se define la sucesión de transcripción $\nu = (n_j, n_{j+1}, \ldots, n_{j+k})$ como la sucesión de notas consecutivas de una partitura que inicia en n_j y termina en n_{j+k}, primera nota después de n_j cuyo valor rítmico es igual a una breve.

Notación. Los elementos de una sucesión de transcripción ν de $k+1$ notas, serán denotados como $\nu_0, \nu_1, \ldots, \nu_k$, con ν_0 el primer elemento de la sucesión ν y a partir del cual esta se define, y ν_k el último elemento de la sucesión ν, es decir, la primera nota breve después de ν_0.

Notación. El signo $\leftarrow$ se utiliza en expresiones como $a \leftarrow b$ para indicar que a toma el valor de b. En este contexto se utilizará con la sucesión de notas ν a transcribir de la forma $\nu_0 \leftarrow \nu_i$, esta expresión tiene un significado más preciso, como lo indica la siguiente definición.

Definición. $\nu_0 \leftarrow \nu_i$ si y solo si la última nota transcrita de la sucesión ν es ν_{i-1}, entonces la nueva sucesión de notas a transcribir se define a partir de ν_i, es decir, ν_i es el primer elemento de la nueva sucesión de notas ν a transcribir (es el elemento ν_0).

Las sucesiones ν servirán para dividir la partitura en unidades de transcripción. Se inicia con ν_0 como la primera nota de la partitura y se define una sucesión ν a partir de este ν_0, sea $\nu = (\nu_0, \ldots, \nu_k)$. Para transcribir esta sucesión y generar la siguiente sucesión ν, que representa el siguiente segmento de la partitura por transcribir, se evalúan las 3 siguientes condiciones. Estas condiciones son mutuamente excluyentes y tratan sobre el valor de ν_0.

0 Si ν_0 es una breve con punto (código: *breve.*), entonces:

- ν_0 se transcribe como una blanca con puntillo (código: *2.*)
- $\nu_0 \leftarrow \nu_1$

1 Si ν_0 es un silencio de duración mayor o igual a una breve, entonces:

1.1 Si ν_0 es silencio de breve:

- ν_0 se transcribe como un silencio de blanca con puntillo.

1.2 Si ν_0 es silencio de longa:

- ν_0 se transcribe como dos silencios de blanca con puntillo.

1.3 Si ν_0 es silencio de máxima:

- ν_0 se transcribe como cuatro silencios de blanca con puntillo.

- Y ahora $\nu_0 \leftarrow \nu_1$, es decir, ν_1 es la nota de inicio que define la siguiente sucesión ν de notas por transcribir.

2 Si ν_0 es un sonido de breve, entonces sean $t_1, \ldots, t_{k-1}$ el valor en tiempos de semibreve de las notas $\nu_1 \ldots, \nu_{k-1}$ respectivamente, y sea $t = \sum_{i=1}^{k-1} t_i$. Hay dos opciones: $t \in Z$ y $t \notin Z$.

2.1 Si $t \in Z$:

Por la propiedad de división euclideana (ver Teorema 1, de la Sección 3), $\exists! q, r \in Z$ tal que $t = 3q + r$ con $0 \leq r < 3$. La relación de congruencia módulo 3, $r = t \pmod 3$, define 3 clases de equivalencia para los diferentes residuos (0, 1 y 2).

2.1.1 Si $r = 1$:

- *Imperfección.* ν_0 se transcribe como una blanca (código en LilyPond de la figura: *2*)
- Las notas $(\nu_1, \ldots, \nu_{k-1})$ se transcriben según el Cuadro 4.5
- $\nu_0 \leftarrow \nu_k$, es decir, ν_k es la nota inicial que define la siguiente sucesión ν por transcribir, ya que al momento se ha transcrito hasta la nota representada por ν_{k-1}.

2.1.2 Si $r = 2$, hay dos opciones:

2.1.2.1 Si $t = 2$:

Generalmente se optará por realizar una *alteración*; sin embargo, hay circunstancias que no la permiten y forzan a una *imperfección.*

- Si se cumple: ν_{k-1} es silencio o ν_{k-1} es menor a una semibreve; entonces:
  - *Imperfección a.p.p.* ν_0 se transcribe como una blanca (código LilyPond: *2*)

- *Imperfección a.p.a.* ν_k se transcribe como una blanca (código: *2*)
- Las notas $(\nu_1, \ldots, \nu_{k-1})$ se transcriben según el Cuadro 4.5
- Si ν_k representa a la j−ésima nota n_j de la melodía, entonces $\nu_0 \leftarrow n_{j+1}$, es decir, n_{j+1} es la nota inicial que define la siguiente sucesión ν por transcribir, ya que se ha transcrito hasta la nota n_j.

◦ De lo contrario:

- ν_0 se transcribe como una blanca con puntillo (código: *2.*)
- *Alteración.* ν_{k-1} se transcribe como una blanca (código: *2*)
- Las notas $(\nu_1, \ldots, \nu_{k-2})$ se transcriben según el Cuadro 4.5
- $\nu_0 \leftarrow \nu_k$

2.1.2.2 Si $t > 3$:

Generalmente se optará por realizar una *imperfección a.p.p. y a.p.a.*; sin embargo, hay circunstancias que no la permiten y forzan a una *alteración.*

◦ Si se cumple: ν_0 es silencio de breve o ν_k es silencio de breve o ν_{k+1} es una breve (silencio o sonido, es indiferente) o ν_{k-2} y ν_{k-1} son sonidos semibreves ligados mensuralmente; entonces

- ν_0 se transcribe como una blanca con puntillo (código: *2.*)
- *Alteración.* ν_{k-1} se transcribe como una blanca (código: *2*)
- Las notas $(\nu_1, \ldots, \nu_{k-2})$ se transcriben según el Cuadro 4.5
- $\nu_0 \leftarrow \nu_k$

◦ De lo contrario:

- *Imperfección a.p.p.* ν_0 se transcribe como una blanca (código LilyPond: *2*)
- *Imperfección a.p.a.* ν_k se transcribe como una blanca (código: *2*)
- Las notas $(\nu_1, \ldots, \nu_{k-1})$ se transcriben según el Cuadro 4.5
- Si ν_k representa a la j−ésima nota n_j de la melodía, entonces $\nu_0 \leftarrow n_{j+1}$.

2.1.3 Si $r = 0$, hay dos opciones:

2.1.3.1 Si $t = 0$:

- ν_0 se transcribe como una blanca con puntillo (código: *2.*)
- $\nu_0 \leftarrow \nu_k$

2.1.3.2 Si $t = 3$:

- ν_0 se transcribe como una blanca con puntillo (código: *2.*)
- Las notas $(\nu_1, \ldots, \nu_{k-1})$ se transcriben según el Cuadro 4.5
- $\nu_0 \leftarrow \nu_k$

2.1.3.3 Si $t > 3$:

Generalmente se optará por realizar una *imperfección a.p.p* y una *alteración*; sin embargo, hay circunstancias que no permiten realizar la una o la otra.

- Si se cumple: ν_{k-1} es un silencio o ν_{k-1} es un sonido menor a una semibreve o ν_0 es un silencio de breve; entonces
  - ν_0 se transcribe como una blanca con puntillo (código: *2.*)
  - Las notas $(\nu_1, \ldots, \nu_{k-1})$ se transcriben según el Cuadro 4.5
  - $\nu_0 \leftarrow \nu_k$
- De lo contrario:
  - *Imperfección a.p.p.* ν_0 se transcribe como una blanca (código: *2*)
  - *Alteración* ν_{k-1} se transcribe como una blanca (código: *2*)
  - Las notas $(\nu_1, \ldots, \nu_{k-2})$ se transcriben según el Cuadro 4.5
  - $\nu_0 \leftarrow \nu_k$

2.2 Si $t \notin Z$:

En este caso, uno de los "puntillos" de las notas, es realmente un *punctus divisionis*. Por lo tanto, se ubica una nota de la sucesión ν que posea un "puntillo", supóngase que el índice de esta nota es i, entonces sea ν_i la nota con el "puntillo". Para comprobar si dicho "puntillo" es realmente un *punto de división* se realiza el siguiente procedimiento:

(a) Sea ν_i' la misma nota que ν_i pero con 2/3 de su duración (es decir, se le quita el valor agregado por el puntillo) y defínase las sucesiones de notas anteriores y posteriores al punto, $s_1 = (\nu_0, ..., \nu_{i-1}, \nu_i')$ y $s_2 = (\nu_{i+1}, ..., \nu_k)$, respectivamente.

(b) Sea $t_{s_1} = \sum_{j=1}^{i-1} t_j + \frac{2}{3} t_i$ y $t_{s_2} = \sum_{j=i+1}^{k-1} t_j$ la duración en tiempos de semibreve del cada una de estas sucesiones de notas (sin tomar en cuenta las breves).

(c) Si $t_{s_1}, t_{s_2} \in Z$, entonces el "puntillo" de ν_i es realmente un *punctus divisionis*.

(d) De lo contrario, se debe buscar otra nota de la sucesión que posea un puntillo, supóngase que el índice de esta nota es un número entero i, entonces se repite el procedimiento anterior con ν_i hasta que la condición *(c)* sea verdadera.

Al determinar la nota ν_i de la sucesión ν que contiene el *punctus divisionis*, se utilizan los valores t_{s_1} y t_{s_2} y se definen $r_1 = t_{s_1} \pmod 3$ y $r_2 = t_{s_2} \pmod 3$. Los valores de r_1 y r_2 determinan la transcripción de las notas de la sucesiones s_1 y s_2, al evaluar las siguientes condiciones:

2.2.1 Si $r_1 = 1$:

- *Imperfección a.p.p.* ν_0 se transcribe como una blanca (código: *2*)
- Las notas $(\nu_1, \ldots, \nu_i')$ se transcriben según el Cuadro 4.5

2.2.2 Si $r_1 = 2$:

- ν_0 se transcribe como una blanca con puntillo (código: *2.*)
- *Alteración.* ν_i' se transcribe como una blanca (código: *2*)
- Las notas $(\nu_1, \ldots, \nu_{i-1})$ se transcriben según el Cuadro 4.5

2.2.3 Si $r_1 = 0$:

- ν_0 se transcribe como una blanca con puntillo (código: *2.*)
- Las notas $(\nu_1, \ldots, \nu_i')$ se transcriben según el Cuadro 4.5

2.2.4 Si $r_2 = 1$:

- *Imperfección a.p.a.* ν_k se transcribe como una blanca (código: *2*)
- Las notas $(\nu_{i+1}, \ldots, \nu_{k-1})$ se transcriben según el Cuadro 4.5
- Si ν_k representa a la j−ésima nota n_j de la melodía, entonces $\nu_0 \leftarrow n_{j+1}$

2.2.5 Si $r_2 = 2$:

- *Alteración.* ν_{k-1} se transcribe como una blanca (código: *2*)
- Las notas $(\nu_{i+1}, \ldots, \nu_{k-2})$ se transcriben según el Cuadro 4.5
- $\nu_0 \leftarrow \nu_k$

2.2.6 Si $r_2 = 0$:

- Las notas $(\nu_{i+1}, \ldots, \nu_{k-1})$ se transcriben según el Cuadro 4.5
- $\nu_0 \leftarrow \nu_k$

3 Si ν_0 es menor a una breve (tanto sonido como silencio), entonces sean $t_0, \ldots, t_{k-1}$ el valor en tiempos de semibreve de las notas $\nu_0 \ldots, \nu_{k-1}$ respectivamente, y sea $t = \sum_{i=0}^{k-1} t_i$. Luego del estudio de muchas partituras de esta mensuración, se deduce que $t \in Z$; por lo tanto, se puede definir $r = t \pmod 3$.

3.1 Si $r = 1$:

- *Imperfección a.p.a.* ν_k se transcribe como una blanca (código: *2*)
- Las notas $(\nu_0, \ldots, \nu_{k-1})$ se transcriben según el Cuadro 4.5
- Si ν_k representa a la j−ésima nota n_j de la melodía, entonces $\nu_0 \leftarrow n_{j+1}$

3.2 Si $r = 2$:

- *Alteración.* ν_{k-1} se transcribe como una blanca (código: *2*)
- Las notas $(\nu_0, \ldots, \nu_{k-2})$ se transcriben según el Cuadro 4.5
- $\nu_0 \leftarrow \nu_k$

3.3 Si $r = 0$:

- Las notas $(\nu_0, \ldots, \nu_{k-1})$ se transcriben según el Cuadro 4.5

- $\nu_0 \leftarrow \nu_k$

A partir de la primera sucesión ν se comienza la transcripción de la melodía y se determina la siguiente sucesión de notas a transcribir, este proceso se repite una y otra vez; de este manera se transcribe poco a poco la melodía (por bloques sucesivos de notas). La repetición finaliza cuando se alcanza la última breve de la voz (ya que no se puede definir otra sucesión ν de transcripción a partir de este punto, por la misma definición de *sucesión* ν). Como se indicó en la Sección 4.5.2, las partituras mensurales en *tempus perfectum cum prolatione imperfecta* terminan en una longa. Por lo tanto, al llegar a la última sucesión ν de transcripción (con ν_k la última breve de la melodía), se procede a definir una última sucesión x de notas consecutivas. Dicha sucesión inicia con la nota que le sigue a la última nota transcrita por el proceso anterior (esta sería ν_0), que de ahora en adelante será llamada x_0, y termina con la última nota de la voz (la longa). Se vuelven a evaluar las condiciones *0*, *1*, *2* y *3*. Como se utilizan las mismas condiciones, la transcripción es la misma que para las sucesiones ν pero, como se observa en la Figura 4.8, luego de la última nota transcrita x_k se agrega la misma nota pero con el valor de una blanca con puntillo (código: *2.*), y éstas dos (la transcripción de x_k y la nota agregada) se ligan añadiendo al final de ellas los símbolos "\\(" y "\\)", respectivamente. Así finaliza el proceso de transcripción de una voz (melodía) de la partitura mensural.

4.5.4. Implementación

A continuación se muestra la solución para este tipo de mensuración. El algoritmo posee el procedimiento *TranscriptePI* que realiza la transcripción, pero para esto usa las funciones que lo preceden y también de la función *ChangeNoteValue* que se muestra en el Algoritmo 4

Algorithm 6 Cambio de la secuencia de notas mensurales *NOTE* a la sucesión de notas actuales para el caso *tempus perfectum cum prolatione imperfecta*.

```
1:  function GetNoteValue(note)
2:      MVALUE = ["16.", "8.", "4.", "2.","1.", "\\breve.", "\\longa.", "\\maxima.", "16", "8",
    "4", "2","1", "\\breve", "\\longa", "\\maxima"]
3:      for i running from 0 until the length of the array MVALUE do
4:          if MVALUE_i in note then
5:              mvalue = MVALUE_i
6:              break
7:          end if
8:      end for
9:      return mvalue
10: end function
11:
12: function GetNote(note)
13:     mvalue = GetNoteValue(note)
14:     just_note ← note without the mvalue
15:     return just_note
16: end function
17:
```

```
18: function TimeSemibreve(note)
19:     MVLAUE = [“16.”, “8.”, “4.”, “2.”,“1.”, “16”, “8”, “4”, “2”,“1”]
20:     TIMESEMIB = [3/32, 3/16, 3/8, 3/4, 3/2, 1/16, 1/8, 1/4, 1/2, 1]
21:     for i running from 0 until the length of the array MVALUE do
22:         if MVALUE_i in mnote then
23:             time_semib = TIMESEMIB_i
24:             break
25:         end if
26:     end for
27:     return time_semib
28: end function
29:
30: function GetActualValue(mnote)
31:     MVALUE = [“16.”, “8.”, “4.”, “2.”,“1.”, “16”, “8”, “4”, “2”,“1”]
32:     AVALUE = [“64.”, “32.”, “16.”, “8.”, “4.”, “64”,“32”, “16”, “8”, “4”]
33:     for i running from 0 until the length of the array MVALUE do
34:         if MVALUE_i in mnote then
35:             avalue = AVALUE_i
36:             break
37:         end if
38:     end for
39:     return avalue
40: end function
41:
42: function AddLigature(note, ligature)
43:     if “\\]” in note then
44:         Insert the ligature before the sign \]
45:     else
46:         Insert the ligature at the end of note
47:     end if
48:     return note
49: end function
50:
51: function Remove(word, char)
52:     first ← characters of word before char
53:     last ← characters of word after the char
54:     new_word ← join the strings first, char and last, in that order
55:     return new_word
56: end function
57:
58: procedure TranscriptePI(NOTE, new_file)
59:     start = 0
60:     NEW_NOTE = [ ]
61:     INDEXBREVES = [ ] ▷ Arreglo de los índices de las breves y de la última longa para las
    notas de una voz
62:     for i running from 0 until the length of the array NOTE do
63:         if “\\breve” in NOTE_i then
64:             Add i at the end of the array INDEXBREVES
65:         end if
66:     end for
67:     L ← length of the array NOTE
68:     Add the value L-1 at the end of the array INDEXBREVES
```

```
    for j running fro 0 until the length of the array INDEXBREVES do
        if INDEXBREVES_j > start then
            end = INDEXBREVES_j
            break
        end if
    end for
    while start < end do
        ν = [ ]
        for j running from start to end do
            Add NOTE_j at the end of the array ν
        end for
        NEW_NU = [ ]

        if GetNoteValue(ν_0) = "\\breve." then                    ▷ 0: punctus perfectionis
            new_note = ChangeNoteValue(ν_0, "\\breve.", "2.")
            Add new_note at the end of the array NEW_NU
            start = start + 1

        else if "r" in GetNote(ν_0) and GetNoteValue(ν_0) is "\\breve" or "\\longa" or
    "\\maxima" then                                               ▷ 1: ν_0 ≥ silencio de breve
            new_note = ChangeNoteValue(ν_0, "\\breve", "2.")
            if GetNoteValue(ν_0) = "\\breve" then
                Add new_note at the end of the array NEW_NU
            else if GetNoteValue(ν_0) = "\\longa" then
                Add new_note at the end of the array NEW_NU
                Add new_note at the end of the array NEW_NU
            else
                Add new_note at the end of the array NEW_NU
                Add new_note at the end of the array NEW_NU
                Add new_note at the end of the array NEW_NU
                Add new_note at the end of the array NEW_NU
            end if
            start = start + 1

        else if "r" not in GetNote(ν_0) and GetNoteValue(ν_0) = "\\breve" then    ▷ 2: ν_0
    sonido de breve
            k = end - start
            t = 0
            if k > 1 then
                for j running from 1 until k do
                    t = t + TimeSemibreve(ν_j)
                end for
            end if

            if t is an integer then                               ▷ 2.1: No hay punctus divisionis
                r = t (mod 3)

                if r = 1 then                                     ▷ 2.1.1: Imperfeccion a.p.p.
                    new_note = ChangeNoteValue(ν_0, "\\breve", "2")
                    Add new_note at the end of the array NEW_NU
                    for i running from 1 to k-1 do
                        new_note = ChangeNoteValue(ν_i,GetNoteValue(ν_i),GetActualValue(ν_i))
                        new_note at the end of the array NEW_NU
                    end for
                    start = end
```

```
122:
123:            else if r = 2 then                                                  ▷ 2.1.2
124:                if t = 2 then          ▷ 2.1.2.1: Exactamente dos semibreves entre breves
125:                                        ▷ La opción predilecta es la alteración, excepto cuando:
126:                    if GetNoteValue(ν_{k-1}) is not "1" or 'r' is in GetNote(ν_{k-1}) then  ▷
    Imperfección .ap.p. y a.p.a.
127:                        new_note = ChangeNoteValue(ν_0, "\\breve", "2")
128:                        Add new_note at the end of the array NEW_NU
129:                        for i from 1 to k-1 do
130:                            new_note = ChangeNoteValue(ν_i,GetNoteValue(ν_i),GetActualValue(ν_i))
131:                            Add new_note at the end of the array NEW_NU
132:                        end for
133:                        new_note = ChangeNoteValue(ν_k, GetNoteValue(ν_k), "2")
134:                        Add new_note at the end of the array NEW_NU
135:                        start = end + 1
136:                    else                                                        ▷ Alteración
137:                        new_note = ChangeNoteValue(ν_0, "\\breve", "2.")
138:                        Add new_note at the end of the array NEW_NU
139:                        for i from 1 to k-2 do
140:                            new_note = ChangeNoteValue(ν_i,GetNoteValue(ν_i),GetActualValue(ν_i))
141:                            Add new_note at the end of the array NEW_NU
142:                        end for
143:                        new_note = ChangeNoteValue(ν_{k-1}, "1", "2")
144:                        Add new_note at the end of the array NEW_NU
145:                        start = end
146:                    end if
147:                else                    ▷ 2.1.2.2: 5, 8, 11, ... semibreves entre breves
148:                                        ▷ La opción predilecta es la imperfección, excepto cuando:
149:                    if "r" in GetNote(ν_k) or "]" in ν_{k-1} then               ▷ Alteración
150:                        new_note = ChangeNoteValue(ν_0, "\\breve", "2.")
151:                        Add new_note at the end of the array NEW_NU
152:                        for i from 1 to k-2 do
153:                            new_note = ChangeNoteValue(ν_i,GetNoteValue(ν_i),GetActualValue(ν_i))
154:                            Add new_note at the end of the array NEW_NU
155:                        end for
156:                        new_note = ChangeNoteValue(ν_{k-1}, "1", "2")
157:                        Add new_note at the end of the array NEW_NU
158:                        start = end
159:                    else if the length of the array NOTE > end + 1 then
160:                        if "\\breve" in NOTE_{end+1} then                       ▷ Alteración
161:                            new_note = ChangeNoteValue(ν_0, "\\breve", "2.")
162:                            Add new_note at the end of the array NEW_NU
163:                            for i from 1 to k-2 do
164:                                new_note = ChangeNoteValue(ν_i,GetNoteValue(ν_i),GetActualValue(ν_i))
165:                                Add new_note at the end of the array NEW_NU
166:                            end for
167:                            new_note = ChangeNoteValue(ν_{k-1}, "1", "2")
168:                            Add new_note at the end of the array NEW_NU
169:                            start = end
170:                        else                                    ▷ Imperfección a.p.p. y a.p.a.
171:                            new_note = ChangeNoteValue(ν_0, "\\breve", "2")
172:                            Add new_note at the end of the array NEW_NU
173:                            for i from 1 to k-1 do
174:                                new_note = ChangeNoteValue(ν_i,GetNoteValue(ν_i),GetActualValue(ν_i))
175:                                Add new_note at the end of the array NEW_NU
176:                            end for
```

```
                        new_note = ChangeNoteValue(ν_k, GetNoteValue(ν_k), "2")
                        Add new_note at the end of the array NEW_NU
                        start = end + 1
                    end if
                else                                          ▷ Imperfeccion a.p.p. y a.p.a.
                    new_note = ChangeNoteValue(ν_0, "\\breve", "2")
                    Add new_note at the end of the array NEW_NU
                    for i from 1 to k-1 do
                        new_note = ChangeNoteValue(ν_i,GetNoteValue(ν_i),GetActualValue(ν_i))
                        Add new_note at the end of the array NEW_NU
                    end for
                    new_note = ChangeNoteValue(ν_k, GetNoteValue(ν_k), "2")
                    Add new_note at the end of the array NEW_NU
                    start = end + 1
                end if
            end if

        else if r = 0 then                                                    ▷ 2.1.3
            if t = 0 then                          ▷ 2.1.3.1: breve seguida de breve
                new_note = ChangeNoteValue(ν_0, "\\breve", "2.")
                Add new_note at the end of the array NEW_NU
                start = end
            else if t = 3 then          ▷ 2.1.3.2: Exactamente 3 semibreves entre breves
                new_note = ChangeNoteValue(ν_0, "\\breve", "2.")
                Add new_note at the end of the array NEW_NU
                for i from 1 to k-1 do
                    new_note = ChangeNoteValue(ν_i,GetNoteValue(ν_i),GetActualValue(ν_i))
                    Add new_note at the end of the array NEW_NU
                end for
                start = end
            else                                ▷ 2.1.3.3: 6, 9, 12, ... semibreves entre breves
        ▷ La opción predilecta es la imperfección a.p.p. y la alteración, excepto cuando:
                if GetNoteValue(ν_{k-1}) is not "1" or "r" is in GetNote(ν_{k-1}) then
                    new_note = ChangeNoteValue(ν_0, "\\breve", "2.")
                    Add new_note at the end of the array NEW_NU
                    for i from 1 to k-1 do
                        new_note = ChangeNoteValue(ν_i,GetNoteValue(ν_i),GetActualValue(ν_i))
                        Add new_note at the end of the array NEW_NU
                    end for
                    start = end
                else                                    ▷ Imperfección a.p.p. y alteración
                    new_note = ChangeNoteValue(ν_0, "\\breve", "2")
                    Add new_note at the end of the array NEW_NU
                    for i from 1 to k-2 do
                        new_note = ChangeNoteValue(ν_i,GetNoteValue(ν_i),GetActualValue(ν_i))
                        Add new_note at the end of the array NEW_NU
                    end for
                    new_note = ChangeNoteValue(ν_{k-1},GetNoteValue(ν_{k-1}),"2")
                    Add new_note at the end of the array NEW_NU
                    start = end
                end if
            end if
        end if
```

```
231:        else                                   ▷ 2.2: Si t no es entero, hay punctus divisionis
232:            k = end - start
233:            t_{s_1} = 0
234:            t_{s_2} = 0
235:            ind_punto = 0
236:            for i running from 0 until k do
237:                if "."in ν_i then
238:                    t_{s_1} = 0
239:                    t_{s_2} = 0
240:                    ind_punto = i
241:                    if ind_punto > 1 then
242:                        for j running from 1 to ind_punto - 1 do
243:                            t_{s_1} = t_{s_1} + TimeSemibreve(ν_j)
244:                        end for
245:                    end if
246:                    t_{s_1} = t_{s_1} + 2/3 * TimeSemibreve(ν_{ind_punto})
247:                    if ind_punto < k-1 then
248:                        for j running from ind_punto +1 to k-1 do
249:                            t_{s_2} = t_{s_2} + TimeSemibreve(ν_j)
250:                        end for
251:                    end if
252:                    if t_{s_1}, t_{s_2} ∈ Z then
253:                        ν_{ind_punto} = Remove(ν_{ind_punto}, ".")
254:                        break
255:                    end if
256:                end if
257:            end for
258:            r_1 = t_{s_1} (mod 3)
259:            r_2 = t_{s_2} (mod 3)
260:
261:                                          ▷ Sucesión antes del punto de división:
262:            if r_1 = 1 then                ▷ 2.2.1: Imperfección a.p.p.
263:                new_note = ChangeNoteValue(ν_0, "\\breve", "2")
264:                Add new_note at the end of the array NEW_NU
265:                for i running from 1 to ind_punto do
266:                    new_note = ChangeNoteValue(ν_i,GetNoteValue(ν_i),GetActualValue(ν_i))
267:                    Add new_note at the end of the array NEW_NU
268:                end for
269:
270:            else if r_1 = 2 then                                ▷ 2.2.2: Alteración
271:                new_note = ChangeNoteValue(ν_0, "\\breve", "2.")
272:                Add new_note at the end of the array NEW_NU
273:                for i running from 1 to ind_punto -1 do
274:                    new_note = ChangeNoteValue(ν_i,GetNoteValue(ν_i),GetActualValue(ν_i))
275:                    Add new_note at the end of the array NEW_NU
276:                end for
277:                new_note = ChangeNoteValue(ν_{ind_punto},GetNoteValue(ν_{ind_punto}),"2"))
278:                Add new_note at the end of the array NEW_NU
279:
280:            else if r_1 = 0 then                ▷ 2.2.3: Se mantiene perfecta
281:                new_note = ChangeNoteValue(ν_0, "\\breve", "2.")
282:                Add new_note at the end of the array NEW_NU
283:                for i running from 1 to ind_punto do
284:                    new_note = ChangeNoteValue(ν_i,GetNoteValue(ν_i),GetActualValue(ν_i))
285:                    Add new_note at the end of the array NEW_NU
286:                end for
```

```
                                                          ▷ Sucesión después del punto de división:
        else if r_2 = 1 then                                          ▷ 2.2.4: Imperfección a.p.a.
            for i running from ind_punto +1 to k-1 do
                new_note = ChangeNoteValue(ν_i,GetNoteValue(ν_i),GetActualValue(ν_i))
                Add new_note at the end of the array NEW_NU
            end for
            new_note = ChangeNoteValue(ν_k, GetNoteValue(ν_k), "2")
            Add new_note at the end of the array NEW_NU
            start = end + 1

        else if r_2 = 2 then                                          ▷ 2.2.5: Alteración
            for i running from ind_punto +1 to k-2 do
                new_note = ChangeNoteValue(ν_i,GetNoteValue(ν_i),GetActualValue(ν_i))
                Add new_note at the end of the array NEW_NU
            end for
            new_note = ChangeNoteValue(ν_{k-1}, GetNoteValue(ν_{k-1}), "2")
            Add new_note at the end of the array NEW_NU
            start = end

        else if r_2 = 0 then                                                  ▷ 2.2.6
            for i running from ind_punto +1 to k-1 do
                new_note = ChangeNoteValue(ν_i,GetNoteValue(ν_i),GetActualValue(ν_i))
                Add new_note at the end of the array NEW_NU
            end for
            start = end
        end if
    end if

    else if not ("\\breve" or "\\longa" or "\\maxima" in GetNoteValue(ν_0)) then   ▷ 3:
ν_0 < breve
        k = end - start
        t = 0
        for j running from 0 until k do
            t = t + TimeSemibreve(ν_j)
        end for
        r = t (mod 3)

        if r = 1 then                                                 ▷ 3.1: Imperfeccion a.p.a.
            for i running from 0 to k-1 do
                new_note = ChangeNoteValue(ν_i,GetNoteValue(ν_i),GetActualValue(ν_i))
                Add new_note at the end of the array NEW_NU
            end for
            new_note = ChangeNoteValue(ν_k, GetNoteValue(ν_k), "2")
            Add new_note at the end of the array NEW_NU
            start = end + 1

        else if r = 2 then                                            ▷ 3.2: Alteración
            for i running from 0 to k-2 do
                new_note = ChangeNoteValue(ν_i,GetNoteValue(ν_i),GetActualValue(ν_i))
                Add new_note at the end of the array NEW_NU
            end for
            new_note = ChangeNoteValue(ν_{k-1}, GetNoteValue(ν_{k-1}), "2")
            Add new_note at the end of the array NEW_NU
            start = end
```

```
340:
341:            else if r = 0 then                                                    ▷ 3.3
342:                for i running from 0 to k-1 do
343:                    new_note = ChangeNoteValue(ν_i,GetNoteValue(ν_i),GetActualValue(ν_i))
344:                    Add new_note at the end of the array NEW_NU
345:                end for
346:                start = end
347:            end if
348:        end if
349:        for j running from 0 until the length of the array INDEXBREVES do
350:            if  INDEXBREVES_j > start then
351:                end = INDEXBREVES_j
352:                break
353:            end if
354:        end for
355:        NEW_NOTE = NEW_NOTE + NEW_NU
356:    end while
357:
358:    if start = end then                                  ▷ Si la última breve no ha sido transcrita
359:        new_note = ChangeNoteValue(NOTE_end,GetNoteValue(NOTE_end),"2."))
360:        Add new_note at the end of the array NEW_NOTE
361:        start = start + 1
362:    end if
363:    if start > end then ▷ Se le agrega la segunda breve perfecta a la última breve transcrita,
    para componer la longa
364:        L ← length of the array NEW_NOTE
365:        final_note = NEW_NOTE_{L-1}
366:        NEW_NOTE_{L-1} = AddLigature(final_note, "\\(")
367:        new_note = AddLigature(ChangeNoteValue(final_note,GetNoteValue(final_note),"2."),"\\)")
368:        Add new_note at the end of the array NEW_NOTE
369:    end if
370:
371:    for note in NEW_NOTE do
372:        Write note in new_file
373:    end for                                          ▷ write the notes in the output file
374: end procedure
```

4.6 Subproblema 5: transcripción para el caso *Tempus Imperfectum cum Prolatione Perfecta*

En este tipo de mensuración todas las figuras poseen una relación binaria entre sí, excepto la semibreve, ésta es perfecta, es decir, equivalente a tres mínimas. Es similar al caso *tempus perfectum cum prolatione imperfecta*, sólo que aquí la figura perfecta es la semibreve en lugar de la breve. Por lo tanto, las reglas que aplican para este tipo de mensuración son las mismas que las de la Sección 4.5.1, pero centran su atención en la *semibreve* en lugar de la *breve*. En consecuencia, la solución de la presente mensuración es la misma que para el caso *tempus perfectum cum prolatione imperfecta*, la única diferencia es que las sucesiones ν toman como referencia a las semibreves en lugar de las breves.

4.7 Solución del problema general

El Algoritmo 1 de la Sección 4.2 toma cualquier grupo de oraciones del archivo *.ly de entrada que no simbolicen notas, y realiza en éste los cambios señalados por el Cuadro 4.1, y así escribe un nuevo grupo de oraciones en el archivo *.ly de salida. Los algoritmos 5 (con la ayuda del 3) y 6 toman cualquier sucesión de notas del archivo *.ly de entrada, genera su transcripción y la escribe en el archivo *.ly de salida. La idea inicial para abordar el problema consiste en la separación del código del archivo *.ly de entrada en dos partes: la parte que no posee notas y la que contiene únicamente notas; y luego trabajar la transformación de ambas secciones individualmente. Es necesario un procedimiento para separar el código de entrada en estas dos secciones. La solución a este problema se muestra en el Algoritmo 7, donde el archivo de entrada es representado por la variable *source*. Este algoritmo secciona el archivo de entrada en dos arreglos:

- *TEXTS*. Cada elemento de este arreglo es un arreglo de oraciones consecutivas en el archivo de entrada que no representan notas. Las oraciones que no poseen notas en un archivo de entrada son fácilmente reconocibles, porque poseen algún tipo de instrucción (\clef, \time, \new Voice, etc.), las cuáles son indicadas en LilyPond por una diagonal invertida colocada al inicio de la oración (\).

- *NOTES*. Cada elemento de este arreglo es un arreglo de todas las notas pertenecientes a una voz. Las oraciones de LilyPond que contienen notas son fácilmente distinguibles, éstas, a diferencia de los comandos, no inician con una diagonal invertida, excepto si la primera nota de la oración simboliza a una nota que se encuentra al inicio de una ligadura mensural (en este caso la nota será precedida por el signo "\[").

Por ejemplo, para la Figura 4.9, los dos arreglos obtenidos por el Algoritmo 7 son:

TEXTS = [[“\\version “2.16.2””, “”, “\\new MensuralVoice{”, “\\clef “petrucci g””, “\\time 3/2”, “\\relative c”{”], [“\\new MensuralVoice{”, “\\clef “petrucci f””, “\\time 4/4”, “\\relative c”{”], [“}”, “}”]]

NOTES = [[“c\\breve”, “d1”, “e1”, “f\\longa”], [“a2”, “b2”, “c\\breve”]]

Figura 4.9: Código LilyPond para una partitura mensural de dos voces

ejemplo_arrays_texts_notes.ly (%USERPROFILE%\Desktop\)

```
1  \version "2.16.2"
2
3  \new MensuralVoice{
4          \clef "petrucci-g"
5          \time 3/2
6          \relative c''{
7                  c\breve d1
8                  e1 f\longa
9          }
10 }
11 \new MensuralVoice{
12         \clef "petrucci-f"
13         \time 4/4
14         \relative c''{
15                 a2 b2 c\breve
16         }
17 }
```

Algorithm 7 Secciona del archivo de entrada en arreglos de texto y de notas

```
count = 0
COUNT_TEXT_LINE = [ ]
COUNT_NOTES_LINE = [ ]
TEXT = [ ]
TEXTS = [ ]
NOTES_VOICE = [ ]
NOTES = [ ]
for line in source do
    count = count + 1
    words_in_line ← list of elements in line
    if words_in_line = [ ] then                    ▷ line is empty, then is part of the text
        Add line at the end of the array TEXT
        Add count at the end of the array COUNT_TEXT_LINE
    else
        first_word = words_in_line_0
        if ("\\" is at the beggining of first_word and "\\[" is not in first_word) or ("}" in
first_word) then                                   ▷ this means the line has no notes
            if the length of the array COUNT_TEXT_LINE is 0 then
                Add line at the end of the array TEXT
            else
                l ← length of the array COUNT_TEXT_LINE
                d = count - COUNT_TEXT_LINE_{l-1}
                if d > 1 then
                    Add the array TEXT at the end of the array TEXTS
                    TEXT = [ ]
                end if
                Add line at the end of the TEXT
            end if
            Add count at the end of the array COUNT_TEXT_LINE
        else                                       ▷ this means the line representes a group of notes
            if the length of the array COUNT_NOTES_LINE is 0 then
                for note in words_in_line do
                    Add note at the end of the array NOTES_VOICE
                end for
            else
                l ← length of the array COUNT_NOTES_LINE
                d = count - COUNT_NOTES_LINE_{l-1}
                if d > 1 then
                    Add the array NOTES_VOICE at the end of the array NOTES
                    NOTES_VOICE = [ ]
                end if
                for note in words_in_line do
                    Add note at the end of the array NOTES_VOICE
                end for
            end if
            Add count at the end of the array COUNT_NOTES_LINE
        end if
    end if
end for
Add the array TEXT at the end of the array TEXTS
Add the array NOTES_VOICE at the end of the array NOTES
```

Es necesario recordar que el proceso de transcripción de las notas depende del tipo de mensuración de la partitura, como se indicó en la Sección 4.1. Pero el tipo de mensuración en un archivo *.ly se indica en la parte del código que no posee notas. Por lo tanto, es pertinente modificar el Algoritmo 1 para que devuelva un valor que indique la mensuración de la melodía; por este motivo, dicho algoritmo se convierte en la siguiente función:

Algorithm 8 Cambio del código sin notas y obtención del tipo de mensuración de una voz determinada

```
function GeneralModifications(TEXT, new_file)
    mensuration = "4/4"
    MPHRASE = ["MensuralVoice", "4/4", "3/2", "6/4", ""mensural–f"", ""petrucci–f"",
""petrucci–c5"", ""mensural–g"", ""petrucci–g"", ""petrucci–c1"", ""petrucci–c2"",
""petrucci–c3"", ""petrucci–c4""]
    APHRASE = ["Voice", "2/4", "3/4", "6/8", "F", "F", "F", "G", "G", "G", "G", ""G_8"",
""G_8""]
    for line in TEXT do
        new_line = ""
        words_in_line ← list of elements in line
        for word in words_in_line do
            for i from 0 until the length of the array MPHRASE do
                if MPHRASE_i in word then
                    Change MPHRASE_i in word to APHRASE_i
                    if MPHRASE_i is "4/4" or "3/2" or "6/4" then
                        mensuration = MPHRASE_i
                    end if
                end if
            end for
            Add word and a blank space to new_line
        end for
        Write new_line and "\\n" in new_file
    end for
    return mensuration
end function
```

Nótese que si no esta especificado el tipo de mensuración de una voz, el valor retornado es "4/4". LilyPond se comporta de esta manera; al no identificar ningún tipo de mensuración o compás, le asigna inmediatamente el valor de "4/4".

Gracias al valor retornado por la función *GeneralModifications*, se puede definir un método general de transcripción (*Transcripte*) que, con el valor de la mensuración, haga llamar al método de transcripción apropiado:

- **TranscripteII** para el caso *tempus imperfectum cum prolatione imperfecta*.
- **TranscriptePI** para el caso *tempus perfectum cum prolatione imperfecta*.
- **TranscripteIP** para el caso *tempus imperfectum cum prolatione perfecta*.

Véase el Algoritmo 9.

Algorithm 9 Método que llama al algoritmo de transcripción adecuado al tipo de mensuración de una voz

```
procedure TRANSCRIPTE(ARRAY_NOTES, mensuration_type, new_file)
    if mensuration_type = "4/4" then
        Call TranscripteII(ARRAY_NOTES , new_file)
    else if mensuration_type = "3/2" then
        Call TranscriptePI(ARRAY_NOTES , new_file)
    else if mensuration_type = "6/4" then
        Call TranscripteIP(ARRAY_NOTES , new_file)
    end if
end procedure
```

Al momento, se puede seccionar el archivo de entrada en dos arreglos, *TEXTS* y *NOTES*. Los elementos de *TEXTS* pueden ser modificados por la función *GeneralModifications*, y los elementos de *NOTES* por el algoritmo *Transcripte*, sólo hace falta un procedimiento que llame a estos algoritmos cuando sea pertinente. El Algoritmo 10 realiza este proceso, este procedimiento debe seguir al señalado por el Algoritmo 7.

Algorithm 10 Método que llama a los algoritmos GeneralModifications y Transcripte para efectuar los cambios necesarios en el archivo de entrada

```
L ← length of NOTES
for i running from 0 until L do
    mensuration = GeneralModifications(TEXTS_i)
    Call Transcripte(NOTES_i, mensuration)
end for
GeneralModifications(TEXTS_L)
```

Finalmente, uniendo el código de los algoritmos 7 y 10, se obtiene el procedimiento *TextNotes* que divide el código del archivo de entrada (*source*) en secciones de texto y de notas, luego hace llamar a la función *GeneralModifications* para cambiar los textos y al proceso *Transcripte* para cambiar las notas, e imprime el nuevo código en el archivo de salida (*new_file*).

Algorithm 11 Toma el archivo *.ly de entrada *source* y genera el archivo *.ly de salida *new_file* con los textos y notas modificados para generar la partitura en notación actual

```
procedure TEXTNOTES(source, new_file)
    count = 0
    COUNT_TEXT_LINE = [ ]
    COUNT_NOTES_LINE = [ ]
    TEXT = [ ]
    TEXTS = [ ]
    NOTES_VOICE = [ ]
    NOTES = [ ]
    for line in source do
        count = count + 1
        words_in_line ← list of elements in line
```

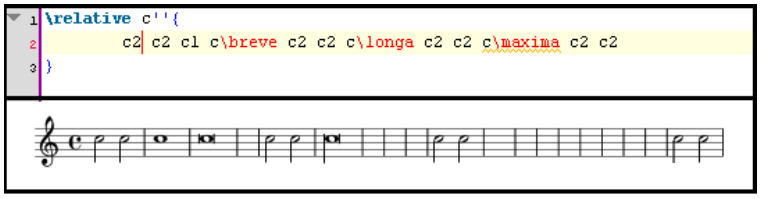
```
        if words_in_line = [ ] then                    ▷ line is empty, then is part of the text
            Add line at the end of the array TEXT
            Add count at the end of the array COUNT_TEXT_LINE
        else
            first_word = words_in_line_0
            if ("\\" is at the beggining of first_word and "\\[" is not in first_word) or ("}" in
first_word) then                                       ▷ this means the line has no notes
                if the length of the array COUNT_TEXT_LINE is 0 then
                    Add line at the end of the array TEXT
                else
                    l ← length of the array COUNT_TEXT_LINE
                    d = count - COUNT_TEXT_LINE_{l-1}
                    if d > 1 then
                        Add the array TEXT at the end of the array TEXTS
                        TEXT = [ ]
                    end if
                    Add line at the end of the TEXT
                end if
                Add count at the end of the array COUNT_TEXT_LINE
            else                                       ▷ this means the line representes a group of notes
                if the length of the array COUNT_NOTES_LINE is 0 then
                    for note in words_in_line do
                        Add note at the end of the array NOTES_VOICE
                    end for
                else
                    l ← length of the array COUNT_NOTES_LINE
                    d = count - COUNT_NOTES_LINE_{l-1}
                    if d > 1 then
                        Add the array NOTES_VOICE at the end of the array NOTES
                        NOTES_VOICE = [ ]
                    end if
                    for note in words_in_line do
                        Add note at the end of the array NOTES_VOICE
                    end for
                end if
                Add count at the end of the array COUNT_NOTES_LINE
            end if
        end if
    end for
    Add the array TEXT at the end of the array TEXTS
    Add the array NOTES_VOICE at the end of the array NOTES
    L ← length of NOTES
    for i running from 0 until L do
        mensuration = GeneralModifications(TEXTS_i, new_file)
        Call Transcripte(NOTES_i, mensuration, new_file)
    end for
    GeneralModifications(TEXTS_L, new_file)
end procedure
```

4.7.1. Modelo algorítmico Finalmente, con toda la herramienta desarrollada a lo largo de este capítulo, se muestra a continuación el modelo algorítmico que resuelve el problema de transcripción de partituras mensurales. El algoritmo utiliza el proceso *TextNotes* (ver *Algoritmo 11*) desarrollado anteriormente que, a su vez, utiliza la función *GeneralModifications* (*Algoritmo 8*) y al proceso *Transcripte* (*Algoritmo 9*), éste último llama al método de transcripción adecuado: *TranscripteII* (*Algoritmo 5*), *TrasncriptePI* (*Algoritmo 6*) o *TranscripteIP*; y estos hacen uso de otras funciones, entre ellas *ChangeNoteValue* (*Algoritmo 4*).

Se enfatizan las restricciones del algoritmo:

- El algoritmo se restringe a los casos de mensuración:
  - *tempus imperfectum cum prolatione imperfecta*
  - *tempus perfectum cum prolatione imperfecta*
  - *tempus imperfectum cum prolatione perfecta*
- No se toma en cuenta coloración ni proporciones.
- **Todas las notas** del archivo *.ly de entrada **deben tener indicada su duración** mediante un número o texto correspondiente.
- No deben haber líneas en blanco entre las notas pertenecientes a una voz en el archivo *.ly de entrada.
- En caso de que alguna voz de la partitura se encuentre en *tempus perfectum cum prolatione imperfecta*, es necesario que dicha voz **termine en una longa** y que **no haya otra longa en cualquier otro sitio de la voz mensural**.
- **Si se quiere indicar que 2 silencios que siguen a una breve pertenecen a distinta perfección**, no los coloque en líneas diferentes del pentagrama, **debe utilizar un punctus divisionis entre ellos**.
- El algoritmo no subdivide las figuras y las liga a través de las barras de compás. Puede ocurrir que la transcripción de una partitura, especialmente en el caso *tempus imperfectum cum prolatione imperfecta* para los factores de conversión *1:1* y *1:2*, contenga máximas, longas o breves. En este caso el archivo *.pdf de la salida tendrá muchos compases en blanco, especialmente si la figura es una máxima (habrá 8 compases vacíos y ninguna figura,debido a que no hay ninguna representación actual para la máxima). Véase Figura **??**. Pero este problema se resuelve fácilmente, solo basta con ingresar al archivo de salida *.ly y editarlo para que en lugar de mostrar estas figuras (y muchos compases vacíos) muestre su subdivisión en figuras equivalentes ligadas a través de las barras de compás.

Figura 4.10: Notas en notación actual con figuras de breves, longas y máximas



El algoritmo para resolver el problema de transcripción es:

Algorithm 12 Solución al problema de transcripción

input_file ← the LilyPond file of the mensural score
output_file ← the LilyPond file that will contain the transcription of *input_file*
Call TextNotes(*input_file*, *output_file*)
close both files

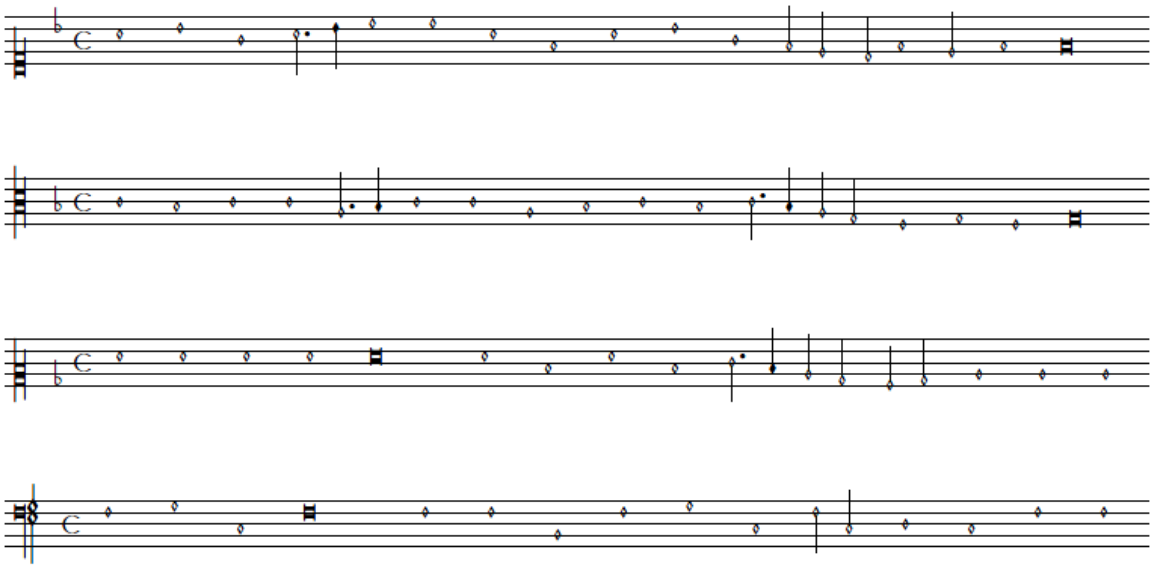
Puede que muchos musicólogos duden del funcionamiento del algoritmo argumentando que el proceso de transcripción es complejo, que hay casos difíciles de resolver para un estudioso, más aún para un algoritmo que trata casos generales. Efectivamente, el proceso de transcripción de una partitura es complejo, el estudioso transcribe la pieza siguiendo ciertas reglas y al finalizar este proceso evalúa si la transcripción tiene sentido, ya que en numerosos casos se ha detectado que los originales contienen errores cometidos por el copista; este análisis se realiza al estudiar diversos aspectos. Entre estos, el musicólogo observa que el número de compases entre las distintas voces sea constante, de no ser así las figuras musicales deben sufrir algunas modificaciones. El estudioso también realiza un análisis armónico para observar si no hay disonancias, en caso de que las haya la pieza transcrita debe sufrir algunas modificaciones en la altura de sus notas, para que la pieza tenga sentido armónico.

El proceso de transcripción es complejo y el presente trabajo no contradice este hecho; el algoritmo no pretende presentar un resultado final y sin errores de la pieza transcrita, sino facilitar un poco el trabajo de transcripción llevado a cabo por el musicólogo. El trabajo del estudioso se puede dividir en dos fases, según lo señalado anteriormente. La primera fase consiste en el proceso mecánico de transcripción siguiendo las reglas establecidas hasta el momento. La segunda fase consiste en evaluar que la pieza esté correctamente transcrita mediante un estudio armónico y rítmico. El algoritmo se centra en la primera fase, en la aplicación mecánica de las reglas. El estudioso puede tomar el resultado del algoritmo como un "borrador" sobre el cuál puede trabajar al utilizarlo para evaluar si la transcripción tiene sentido (con el análisis armónico y rítmico) para producir el resultado final de la pieza transcrita.

4.7.2. Funcionamiento del modelo A continuación se muestran algunos ejemplos del funcionamiento del modelo algorítmico en la transcripción de partituras. De las mensuraciones mencionadas en este trabajo, la más común en su uso es la *tempus imperfectum cum prolatione imperfecta*. Es raro encontrar partituras en *tempus perfectum cum prolatione imperfecta*, y mucho más extraño son los casos en *tempus imperfectum cum prolatione perfecta*; de hecho, después de buscar en muchos cancioneros en notación mensural, no se encontró ningún ejemplo de este último caso, con excepción del Kyrie de Ockeghem (ver Figura 4.19) donde una de las voces se encuentra en esta mensuración; sin embargo, dicha voz posee coloración, aspecto que no está contemplado en el algoritmo.

- ***tempus imperfectum cum prolatione imperfecta*** La siguiente pieza es parte de los *Manuscritos de Santa Eulalia*, un conjunto de libros con piezas originales de la Guatemala colonial.

Figura 4.11: *Hoy nace la nueva estrella*, pieza guatemalteca de la época colonial



Esta partitura fue introducida a LilyPond mediante el siguiente código:

Figura 4.12: Archivo *.ly de la partitura de la Figura 4.11

Hoy nace la nueva estrella.ly (%USERPROFILE%\Dropbox\TESIS\PROGR\MI

```
\version "2.16.2"

\new MensuralVoice{
        \clef "petrucci-c1"
        \time 4/4
        \key  f \major
        \relative c''{
                a1 bes1 g1 a2. bes4 c1 c1 a1 f1
                a1 bes1 g1 f2 e2 d2 f1 e2 f1 f\breve
        }
}
\new MensuralVoice{
        \clef "petrucci-c3"
        \time 4/4
        \key  f \major
        \relative c'{
                c1 bes1 c1 c1 a2. bes4 c1 c1 a1 bes1
                c1 bes1 c2. bes4 a2 g2 f1 g1 f1 g\breve
        }
}
\new MensuralVoice{
        \clef "petrucci-c2"
        \time 4/4
        \key f \major
        \relative c'{
                f1 f1 f1 f1 f\breve f1 d1
                f1 d1 e2. d4 c2 bes2 a2 bes2 c1 c1 c1
        }
}
\new MensuralVoice{
        \clef "petrucci-f"
        \time 4/4
        \relative c{
                f1 g1 c,1 f\breve f1 f1 b,1
                f'1 g1 c,1 f2 c2 d1 c1 f1 f1
        }
}
```

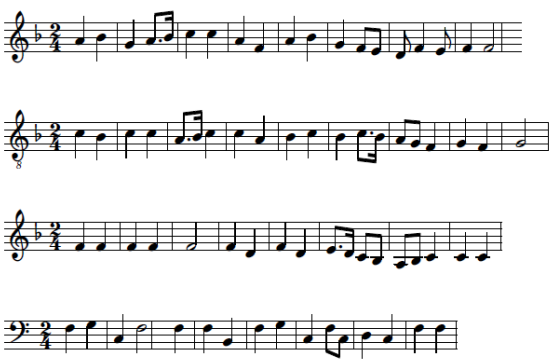
La figura anterior muestra el texto del archivo *.ly de entrada que, al ser modificado por el algoritmo, se convierte en:

Figura 4.13: Transcripción de *Hoy nace la nueva estrella* (Figura 4.11), según el algoritmo

Hoy nace la nueva estrella _ transcription.ly (%USERPROFILE%\Dropbox\TESIS\PROGRAMA PYTHON\Pr

```
\version "2.16.2"

\new Voice{
\clef G
\time 2/4
\key f \major
\relative c''{
a4 bes4 g4 a8. bes16 c4 c4 a4 f4 a4 bes4 g4 f8 e8 d8 f4 e8 f4 f2 }
}
\new Voice{
\clef "G_8"
\time 2/4
\key f \major
\relative c'{
c4 bes4 c4 c4 a8. bes16 c4 c4 a4 bes4 c4 bes4 c8. bes16 a8 g8 f4 g4 f4 g2 }
}
\new Voice{
\clef G
\time 2/4
\key f \major
\relative c'{
f4 f4 f4 f4 f2 f4 d4 f4 d4 e8. d16 c8 bes8 a8 bes8 c4 c4 c4 }
}
```

(a) Archivo *.ly de salida



(b) Partitura *.pdf generada por el archivo *.ly de salida

Nótese que en la Figura 4.13b las voces están distribuidas de la siguiente manera:

- la primera es la soprano (S)
- la segunda es el tenor (T)
- la tercera es la contralto (A)
- y la cuarta voz es el bajo (B)

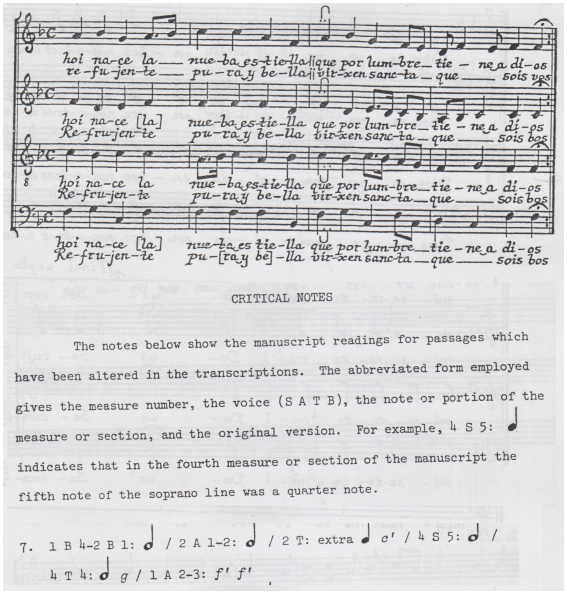
En la Figura 4.14 se encuentra la transcripción de la pieza *Hoy nace la nueva estrella* (de la Figura 4.11) realizada por la M.Mus. Sheila Raney. Junto con la transcripción hay ciertas anotaciones de Raney que indican que realizó algunas modificaciones a la transcripción original, nótese que sin estos cambios la partitura de la Figura 4.14 y de la Figura 4.13b serían iguales. Por lo tanto, la transcripción realizada por el algoritmo es un "borrador" adecuado sobre el cual el musicólogo puede trabajar, estudiando dos aspectos:

1. que la pieza tenga sentido armónico
2. que el número de compases sea constante para todas las voces

Efectivamente, las modificaciones realizadas por Raney son de dos tipos:

1. cambios en la altura de algunas notas, para que la pieza tenga sentido armónico
2. cambios en la duración de algunas notas, para que el número de compases sea el mismo para todas las voces.

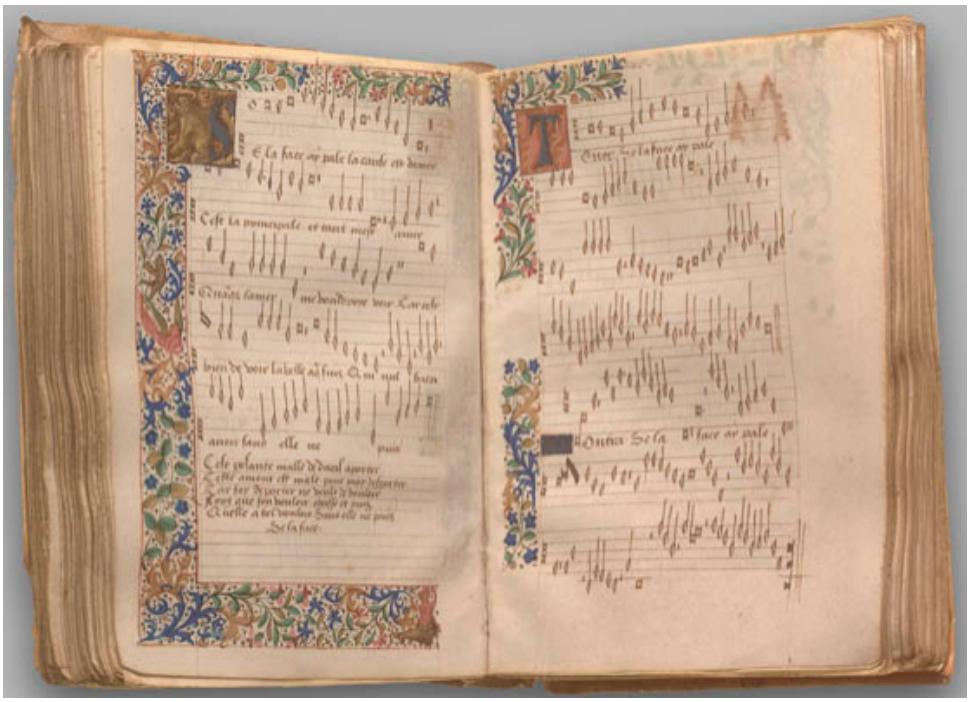
Figura 4.14: Transcripción real de la partitura de la Figura 4.11



(Raney, 1981, p. 155, 206)

- ***tempus perfectum cum prolatione imperfecta*** La siguiente pieza es *Se la face ay palle* de *Guillaume Du Fay*.

Figura 4.15: Se la face ay palle, Guillaume Du Fay



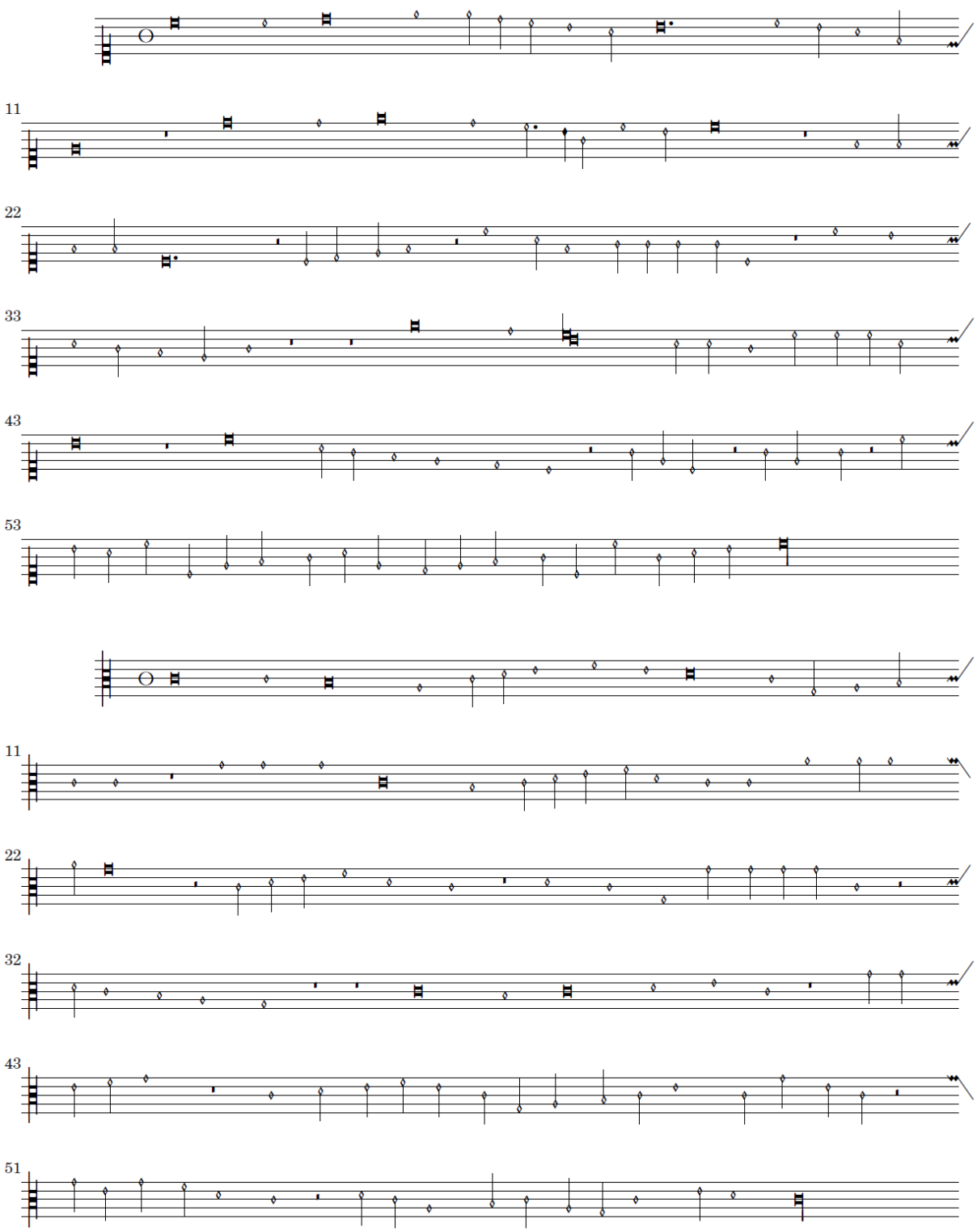
Nótese que hay tres voces en esta partitura. La última de ellas posee coloración, aspecto que no está previsto por el algoritmo. Por lo tanto, el archivo LilyPond de entrada que se muestra en la Figura 4.16, no toma en cuenta la última voz.

Figura 4.16: Archivo *.ly de las dos primeras voces de la partitura de la Figura 4.15

Se_la_face_ay_palle_1.ly (%USERPROFILE%\Dropbox\TESIS\PROGRAMA PYTHON\Programas\COMPLETO - LISTO PARA PRUEBAS EN PARTITURAS\)

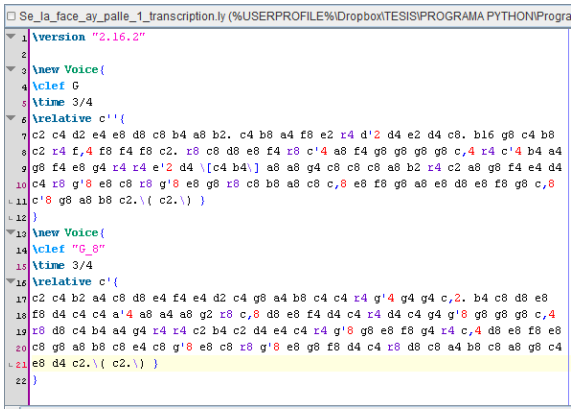
```
\version "2.16.2"

\new MensuralVoice{
        \clef "petrucci-c1"
        \time 3/2
        \relative c''{
                c\breve c1 d\breve e1 e2 d2 c2 b1 a2 b\breve. c1 b2 a1 f2 e\breve r1
                d'\breve d1 e\breve d1 c2. b4 g2 c1 b2 c\breve r1 f,1 f2 f1 f2 c\breve. r2 c2 d2 e2 f1 r2
                c'1 a2 f1 g2 g2 g2 g2 c,1 r1 c'1 b1 a1 g2 f1 e2 g1 r1 r1 e'\breve d1
                \[c1 b1\] a2 a2 g1 c2 c2 c2 a2 b\breve r1 c\breve a2 g2 f1 e1 d1 c1 r2 g'2 e2 c2 r2 g'2 e2 g2 r2
                c2 b2 a2 c2 c,2 e2 f2 g2 a2 e2 d2 e2 f2 g2 c,2 c'2 g2 a2 b2 c\longa
        }
}
\new MensuralVoice{
        \clef "petrucci-c3"
        \time 3/2
        \relative c'{
                c\breve c1 b\breve a1 c2 d2 e1 f1 e1 d\breve c1 g2 a1 b2 c1 c1 r1
                g'1 g1 g1 c,\breve b1 c2 d2 e2 f2 d1 c1 c1 a'1 a2 a1 a2 g\breve r2 c,2 d2 e2 f1 d1 c1 r1
                d1 c1 g1 g'2 g2 g2 g2 c,1 r2 d2 c1 b1 a1 g1 r1 r1 c\breve b1 c\breve d1 e1 c1 r1 g'2 g2 e2 f2 g1 r1 c,1 d2 e2
                f2 e2 c2 g2 a2 b2 c2 e1 c2 g'2 e2 c2 r2 g'2 e2 g2 f2 d1 c1 r2 d2 c2 a1 b2 c2 a2 g2 c1 e2 d1 c\longa
        }
}
```

Figura 4.17: Archivo *.pdf que muestra las dos voces generadas por *Figura 4.16*

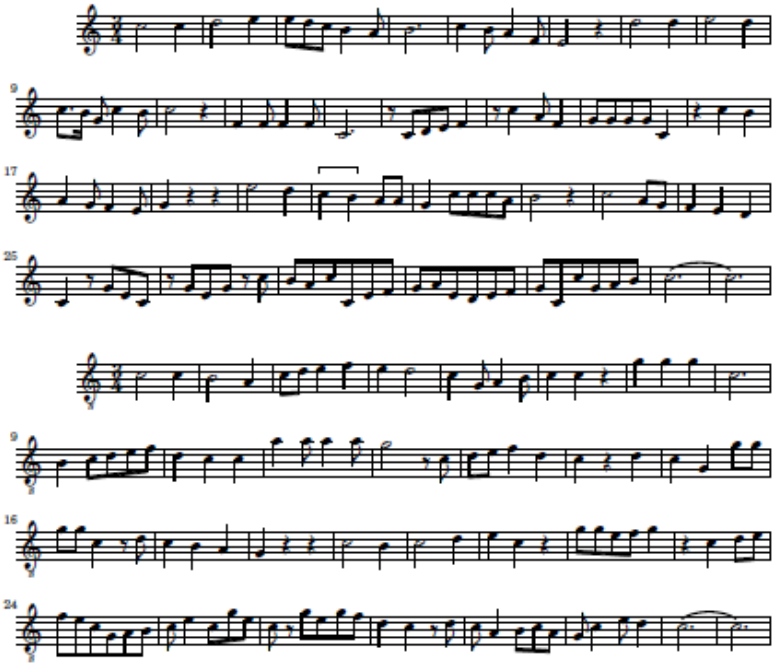


Al utilizar el archivo de la Figura 4.16 como entrada del algoritmo, se genera la salida:

Figura 4.18: Transcripción de las primeras voces de la *Se la face ay palle* de *Guillaume Du Fay* (Figura 4.15), según el algoritmo



(a) Archivo *.ly de salida



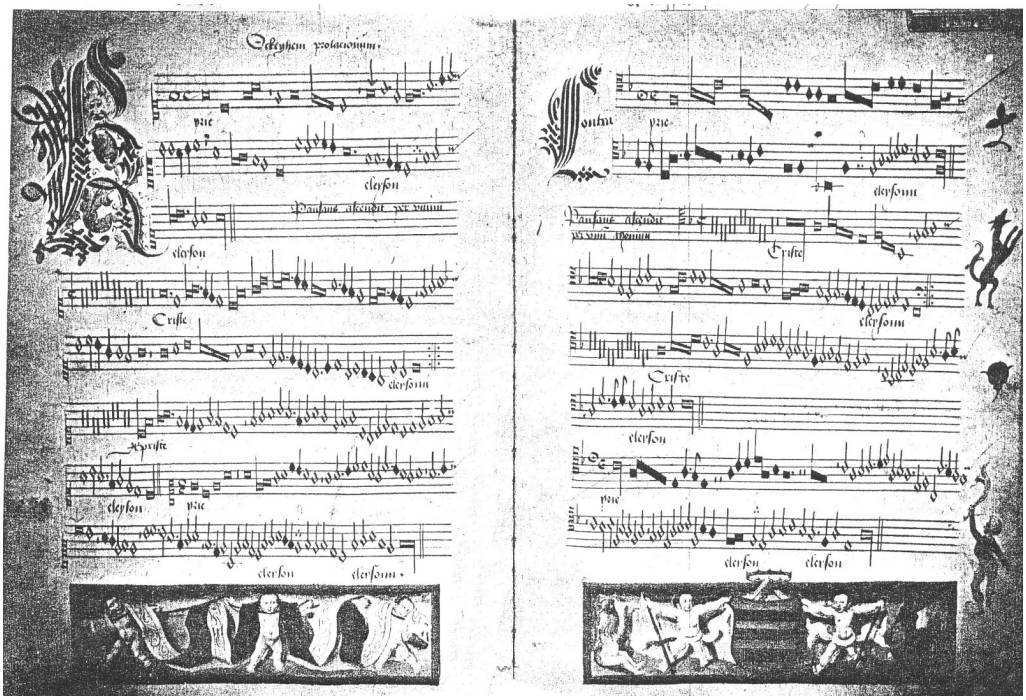
(b) Partitura *.pdf generada por el archivo *.ly de salida

Nótese que ambas voces poseen el mismo número de compases, por lo que esta transcripción no aparenta tener ningún problema en cuanto a duración, sólo restaría hacer un análisis armónico para observar si el altura de las notas de la partitura original es la adecuada.

- **Combinación de ambos casos** La Figura 4.19 muestra el Kyrie de Ockeghem. En la parte superior de ambas páginas se observa un doble símbolo, esto significa que son dos voces en cada página, una se lee con el primer símbolo y la otra con el segundo. En otras palabras: la primera voz de la página izquierda se encuentra en *tempus imperfectum cum prolatione*

imperfecta; la segunda voz está en *tempus perfectum cum prolatione imperfecta*; la primera voz de la página a la derecha está en *tempus imperfectum cum prolatione perfecta*; y la segunda voz de la misma página se encuentra en *tempus perfectum cum prolatione perfecta*.

Figura 4.19: Kyrie de Ockeghem



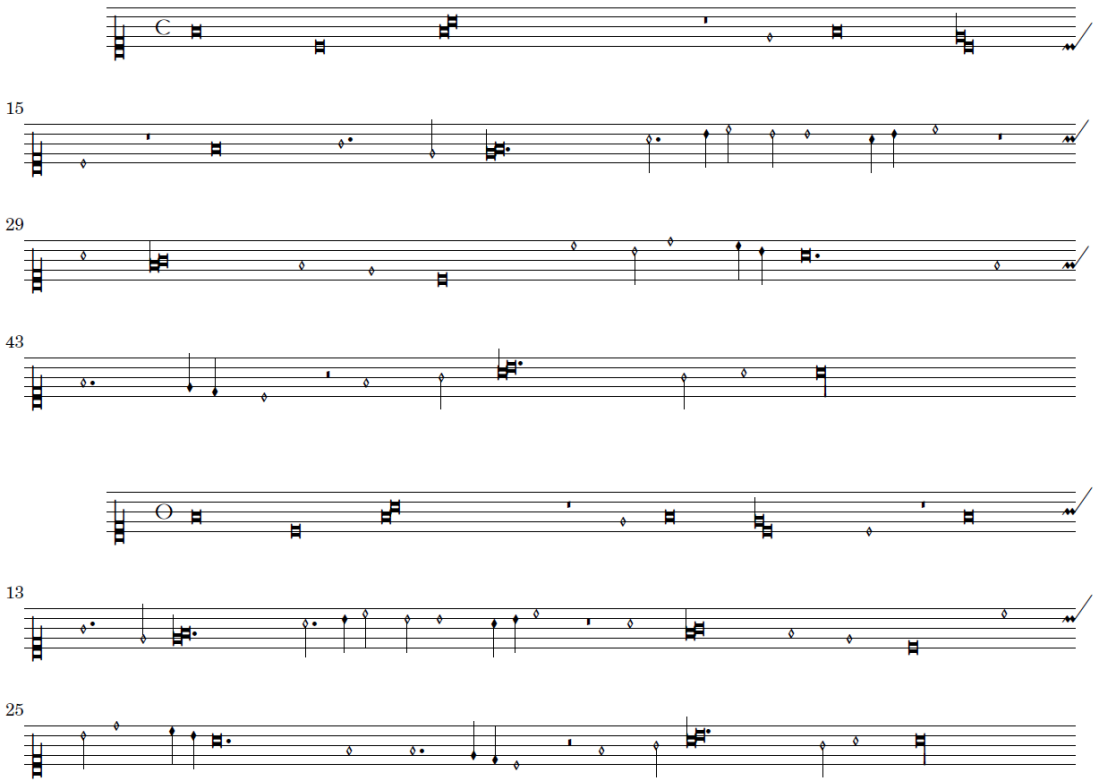
El algoritmo sólo puede trabajar con las dos voces del inicio de la página izquierda de la Figura 4.19, porque la continuación de estas voces se encuentra en otra mensuración y las voces de la página izquierda poseen coloración. Por lo tanto, la entrada del algoritmo es:

Figura 4.20: Archivo *.ly del principio de las dos primeras voces de partitura de la Figura 4.19

Kyrie_Ockeghem.ly (%USERPROFILE%\Dropbox\TESIS\PROGRAMA PYTHON\Programas\COMPLETO - L

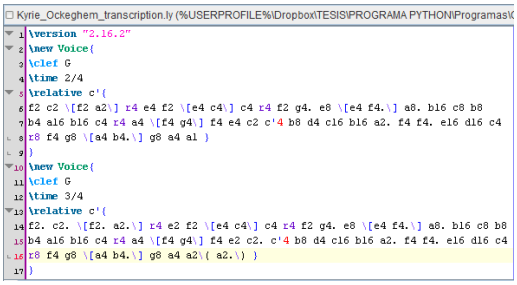
```
\version "2.16.2"
\new MensuralVoice{
        \clef "petrucci-c1"
        \time 4/4
        \relative c'{
                f\breve c\breve \[f\breve a\breve\] r1 e1 f\breve \[e1 c1\]
                c1 r1 f\breve g1. e2 \[e1 f1.\] a2. b4 c2
                b2 b1 a4 b4 c1 r1 a1 \[f1 g1\] f1 e1 c\breve c'1 b2 d1 c4 b4
                a\breve. f1 f1. e4 d4 c1 r2 f1 g2
                \[a1 b1.\] g2 a1 a\longa
        }
}
\new MensuralVoice{
        \clef "petrucci-c1"
        \time 3/2
        \relative c'{
                f\breve c\breve \[f\breve a\breve\] r1 e1 f\breve \[e1 c1\] c1
                r1 f\breve g1. e2 \[e1 f1.\] a2. b4 c2
                b2 b1 a4 b4 c1 r1 a1 \[f1 g1\] f1 e1 c\breve c'1 b2 d1 c4 b4
                a\breve. f1 f1. e4 d4 c1 r2 f1 g2
                \[a1 b1.\] g2 a1 a\longa
        }
}
```

Figura 4.21: Archivo *.pdf que muestra las dos voces generadas por Figura 4.20



De esto se obtiene la siguiente salida:

Figura 4.22: Transcripción del principio de las dos primeras voces de la Kyrie de Ockeghem (Figura 4.19), según el algoritmo



```
Kyrie_Ockeghem_transcription.ly (%USERPROFILE%\Dropbox\TESIS\PROGRAMA PYTHON\Programas\
1  \version "2.16.2"
2  \new Voice{
3  \clef G
4  \time 2/4
5  \relative c'{
6  f2 c2 \[f2 a2\] r4 e4 f2 \[e4 c4\] c4 r4 f2 g4. e8 \[e4 f4.\] a8. b16 c8 b8
7  b4 a16 b16 c4 r4 a4 \[f4 g4\] f4 e4 c2 c'4 b8 d4 c16 b16 a2. f4 f4. e16 d16 c4
8  r8 f4 g8 \[a4 b4.\] g8 a4 a1 }
9  }
10 \new Voice{
11 \clef G
12 \time 3/4
13 \relative c'{
14 f2. c2. \[f2. a2.\] r4 e2 f2 \[e4 c4\] c4 r4 f2 g4. e8 \[e4 f4.\] a8. b16 c8 b8
15 b4 a16 b16 c4 r4 a4 \[f4 g4\] f4 e2 c2. c'4 b8 d4 c16 b16 a2. f4 f4. e16 d16 c4
16 r8 f4 g8 \[a4 b4.\] g8 a4 a2\( a2.\) }
17 }
```

(a) Archivo *.ly de salida

(b) Partitura *.pdf generada por el archivo *.ly de salida

5 Discusión

El algoritmo desarrollado en el presente trabajo modela el proceso de transcripción de una partitura mensural a notación contemporánea. Toma como entrada un archivo *.ly que, al ser compilado por LilyPond, genera una partitura en notación mensural blanca. La salida del algoritmo es también un archivo *.ly, pero la información que contiene genera una partitura en notación contemporánea que corresponde a la transcripción de la partitura mensural de entrada.

El algoritmo puede considerarse como una función cuyo dominio es un subconjunto del conjunto de todos los archivos *.ly que representan una partitura en notación mensural blanca, ya que tanto los archivos *.ly de entrada como las partituras que representan están sujetos a ciertas restricciones:

- Restricciones del archivo *.ly de entrada:
  - Todas las notas del archivo de entrada deben tener indicada su duración (ya sea con un número o un texto)
  - No deben haber líneas en blanco entre las notas pertenecientes a una voz
  - Si se quiere indicar que dos silencios entre breves pertenecen a distinta perfección, es necesario hacer uso de un punto de división entre ellos en el código del archivo *.ly
- Restricciones de la partitura mensural blanca representada por el archivo:
  - La mensuración de la partitura sólo puede ser:
    - tempus imperfectum cum prolatione imperfecta
    - tempus perfectum cum prolatione imperfecta
    - tempus imperfectum cum prolatione perfecta
  - No debe presentar coloración
  - No debe hacer uso de proporciones
  - Si la partitura está en *tempus perfectum cum prolatione imperfecta* es necesario que cada voz finalize en una longa y que no haya ninguna otra longa en otro sitio de la voz mensural.

El dominio del algoritmo es entonces el conjunto de todos los archivos *.ly que generan una partitura en notación mensural blanca sujetos a las restricciones señaladas anteriormente. Pero para afirmar que el algoritmo es realmente una función no es suficiente con mostrar su dominio y que cada elemento de éste se relaciona con la transcripción generada por el algoritmo; hace falta probar que

cada elemento del dominio se relaciona con uno y solo un elemento del contradominio, es decir, se necesita demostrar que para una partitura mensural blanca el algoritmo no puede generar dos transcripciones diferentes.

Si se toman dos partituras mensurales exactamente iguales, se tiene que todas sus notas son las mismas para ambas partituras; para la caso *tempus imperfectum cum prolatione imperfecta*, este hecho tiene como consecuencia inmediata que la transcripción de ambas partituras sea la misma, ya que la transcripción de partituras para esta mensuración se realiza nota por nota.

Si se toman dos partituras mensurales exactamente iguales para la mensuración *tempus perfectum cum prolatione imperfecta*, se tiene que para cada una de las voces de las dos partituras todas sus notas son las mismas. Por lo tanto, para cada voz (soprano, alto, tenor o bajo), la primera sucesión de transcripción (que va desde la primera nota de una voz a la siguiente breve que aparezca en la voz) en ambas partituras es la misma. Por consiguiente el número de semibreves en la primera sucesión de transcripción para dicha voz en ambas partituras es el mismo; y esto genera una misma transcripción para ambas sucesiones (debido a que ésta depende del número de semibreves). Por lo tanto, la última nota transcrita de la primera sucesión para una determinada voz es la misma en ambas partituras, lo que implica que la nota de inicio de la segunda sucesión es también la misma y, por tanto, la segunda sucesión de transcripción (que va desde esta nota de inicio a la siguiente breve) es idéntica para cada voz en ambas partituras.

Si se sigue este razonamiento se tendrá que la tercera sucesión ν de transcripción para cada voz es la misma en ambas partituras, lo mismo se dirá de la cuarta, la quinta, ..., hasta llegar a la última sucesión de transcripción en dicha voz. Por lo tanto, todas las sucesiones ν de transcripción de una determinada voz en ambas partituras serán iguales y, por consiguiente, la transcripción de esa voz es la misma para ambas partituras. Como esto ocurre para cada voz, la transcripción (total) de las dos partituras es idéntica.

Para formalizar el argumento anterior (el que si la primera sucesión ν de transcripción es igual en ambas partituras para una determinada voz, entonces la segunda también y, por tanto, la tercera, etc.) se utilizará el principio de inducción. Para esto es necesario probar dos aspectos: 1. que la primera sucesión de transcripción para una determinada voz es idéntica en las dos partituras y 2. que si una sucesión de transcripción es igual en ambas partituras entonces la siguiente también lo será

Supóngase que A y B representan dos partituras mensurales exactamente iguales. Y sean:

- A_s: sucesión de todas las notas de la soprano para la partitura A
- A_a: sucesión de todas las notas de la alto para la partitura A
- A_t: sucesión de todas las notas del tenor para la partitura A
- A_b: sucesión de todas las notas del bajo para la partitura A

- B_s: sucesión de todas las notas de la soprano para la partitura B
- B_a: sucesión de todas las notas de la alto para la partitura B
- B_t: sucesión de todas las notas del tenor para la partitura B
- B_b: sucesión de todas las notas del bajo para la partitura B

Ahora bien, es necesario probar que todas las sucesiones de transcripción de cada voz son idénticas para las dos partituras. A continuación se encuentra la demostración para una voz específica (la soprano), la prueba es exactamente igual para las otras voces (alto, tenor y bajo).

Sea A_{s_n} y B_{s_n} la n-ésima sucesión de transcripción de la voz soprano para las partituras A y B. Es necesario probar los siguientes dos enunciados, para probar la igualdad de todas las sucesiones de transcripción de la voz soprano en ambas partituras.

- Para cualquier i, A_{s_1} y B_{s_1}
- Si $A_{s_n} = B_{s_n}$ entonces $A_{s_{n+1}} = B_{s_{n+1}}$

Como $A = B$, la sucesión de todas las notas de una voz son iguales para ambas partituras, por lo tanto $A_s = B_s$. Por consiguiente la sucesión de notas que inicia desde la primera nota de A_s hasta la siguiente breve que aparezca debe en A_s ser igual a la sucesión de notas que inicia desde la primera nota de B_s hasta la siguiente breve que aparezca debe en B_s; es decir, $A_{s_1} = B_{s_1}$.

Ahora, supóngase que $A_{s_n} = B_{s_n}$ entonces el número de semibreves que preceden a la última breve de la sucesión A_{s_n} y de la sucesión B_{s_n} es el mismo, por lo tanto su transcripción es la misma (ya que ésta depende del número de semibreves). Entonces la última nota transcrita de las sucesiones A_{s_n} y B_{s_n} es la misma nota, por lo que la nota de inicio de la $n+1$ sucesión de transcripción para las voces A_s y B_s es idéntica. Esto implica que las sucesiones $A_{s_{n+1}}$ y $B_{s_{n+1}}$ son iguales, porque ambas parten desde esa nota de inicio a la siguiente breve.

De esta forma, queda demostrado por inducción que todas las sucesiones de transcripción de la voz soprano son idénticas en ambas partituras, por lo cuál la transcripción de dicha voz es la misma para las dos partituras mensurales. Como esto ocurre también para las demás voces, la transcripción de (total) de las partituras A y B es idéntica. Por lo tanto, al tomar como entrada cualquier partitura mensural blanca el algoritmo genera una única transcripción de la misma. De esto se puede asegurar que el algoritmo desarrollado en este trabajo es una función.

De aquí en adelante para una partitura mensural blanca A, $f(A)$ sera la transcripción de A generada por el algoritmo. Este algoritmo cumple además con ser una función inyectiva. Esto quiere decir que si dos partituras mensurales son diferentes no pueeden tener la misma transcripción; es decir, si A y B son partituras mensurales tales que $A \neq B$, entonces $f(A) \neq f(B)$.

Para probar la inyectividad del algoritmo, considérese dos partituras mensurales A y B tales que $A \neq B$. Si A difiere de B por el altura de alguna de sus notas, es evidente que $f(A) \neq f(B)$.

Entonces supóngase que las partituras A y B no difieren por la altura de sus notas; como $A \neq B$ existe alguna nota cuya duración es diferente en las partituras A y B; si la mensuración de las partituras A y B es *tempus imperfectum cum prolatione imperfecta*, es trivial que $f(A) \neq f(B)$ debido a que la transcripción de A y B se realiza nota por nota.

Ahora resta probar la inyectividad para el otro caso de mensuración tratado en el presente trabajo: *tempus perfectum cum prolatione imperfecta*. Como $A \neq B$ entonces hay al menos una voz que tiene al menos una nota diferente en ambas partituras. Tómese una de estas voces y defínase: $(a_i)_{i=1}^{n}$ como la sucesión de todas las notas de esta voz en A y $(b_i)_{i=1}^{m}$ como la sucesión de todas las notas de la misma voz en B. Como existe al menos una nota diferente en ambas sucesiones de notas, es posible definir una subsucesión para $(a_i)_{i=1}^{n}$ y otra para $(b_i)_{i=1}^{m}$ conformada por todas las notas diferentes entre ambas voces. Ahora, tómese el primer elemento de las dos subsucesiones, sea éste a_j y b_j. Por lo tanto, a_j y b_j son la primera nota diferente en ambas voces; de esto se tiene que $(a_i)_{i=1}^{j-1} = (b_i)_{i=1}^{j-1}$. Entonces $(a_i)_{i=1}^{j-1}$ y $(b_i)_{i=1}^{j-1}$ tendrán las mismas sucesiones ν de transcripción, por lo que su última nota transcrita también sera la misma, sea ésta a_k y b_k (para $^{o} \leq k \leq j-1$); entonces $a_k = b_k$ con $1 \leq k \leq j-1$. Por lo tanto, la primera nota de la siguiente sucesión ν de transcripción para las voces $(a_i)_{i=1}^{n}$ y $(b_i)_{i=1}^{m}$ es a_{k+1} y b_{k+1}, respectivamente, con $k+1 \leq j$. De esto se tiene que la siguiente sucesión ν de transcripción para las voces $(a_i)_{i=1}^{n}$ y $(b_i)_{i=1}^{m}$ son, respectivamente:

- $(a_{k+1}, \cdots, a_j, \cdots, a_{j+r})$ con $k+1 \leq j$ y $j+r$ el +indice de la siguiente breve que aparece luego de a_{k+1}
- $(b_{k+1}, \cdots, b_j, \cdots, b_{j+s})$ con $k+1 \leq j$ y $j+s$ el +indice de la siguiente breve que aparece luego de b_{k+1}

Como ambas sucesiones ν son diferentes (debido a la presencia de a_j en una y b_j en otra), su transcripción es diferente (esto no sera demostrado pero se evidencia al observar las Figuras 5.1 y 5.2, que muestran la transcripción de las distintas sucesiones ν); por lo tanto, la transcripción entera de esa voz es diferente en ambas partituras; lo que implica que la transcripción (total) de las dos partituras es distinta, es decir, $f(A) \neq f(B)$.

Así queda probado que el algoritmo se comporta como una función inyectiva. Como la salida del algoritmo es un archivo *.ly que, al ser compilado por LilyPond, genera una partitura en notación contemporánea, entonces el contradominio del algoritmo es el conjunto de todos los archivos *.ly que representan una partitura en notación contemporánea; la imagen del algoritmo, sin embargo, se restringe a las partituras en notación contemporánea que son transcripciones de alguna partitura en notación mensural. Como no puede garantizarse que toda partitura contemporánea pueda ser reescrita en notación mensural, el algoritmo no se comporta como una función sobreyectiva. Pero si se restringe el contradominio al conjunto imagen, se garantiza la sobreyectividad de la función.

Figura 5.1: Sucesiones ν de transcripción que inician y terminan en breve

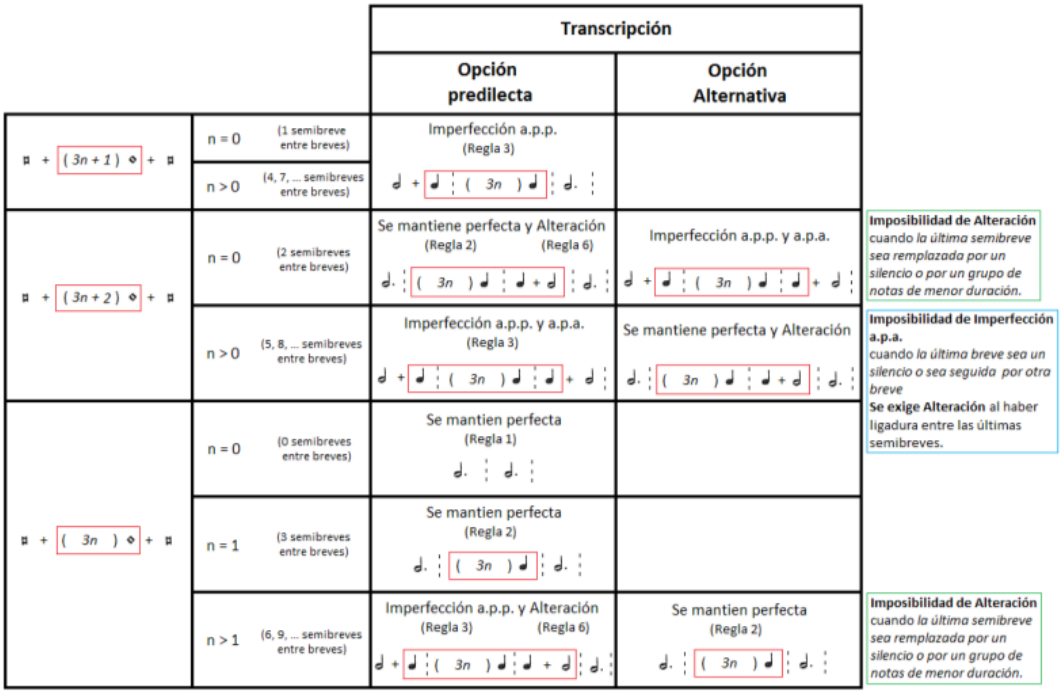


Figura 5.2: Sucesiones ν de transcripción que inician con semibreve y terminan en breve

Esto último hace del algoritmo una función biyectiva; por lo cuál, teóricamente, se puede escribir un algoritmo que trabaje como una función inversa, es decir, se puede desarrollar un algoritmo que tome una transcripción de una partitura mensural y obtenga como resultado la partitura mensural original. Nótese que el algoritmo inverso no toma cualquier partitura contemporánea y genera su equivalente mensural,su dominio sólo consta de las partituras en notación contemporánea que son transcripciones de una partitura en notación mensural blanca.

Por último, es necesario discutir algunos aspectos sobre la validez del algoritmo. Como se indicó en la última sección del capítulo anterior, es fácil dudar de la validez del modelo de transcripción desarrollado, ya que en muchas ocasiones cuando un experto transcribe las partituras mecánicamente se ve en la necesidad de hacer correcciones sobre su transcripción debido a que, después de un análisis armónico y rítmico, encuentra disonancias o incongruencias de tiempo en la transcripción. Los errores que el musicólogo corrige sobre la transcripción mecánica que realizó (que es la misma que realiza el algoritmo) son en su mayoría errores que el copista cometió al escribir la partitura original. Al interpretar la partitura mensural con estos errores el sonido que

produce no es agradable, ya sea por ser disonante (por errores en la altura de las notas) o porque las distintas voces terminan en momentos diferentes, por una nota o dos, a pesar de tratarse de un coral (por errores al escribir alguna figura de más o de menos). Estos dos tipos de error se ilustran en la partitura mensural de la Figura 4.11 de la pieza *Hoy nace la nueva estrella*. Luego de transcribir mecánicamente la partitura de la Figura 4.11, la M.Mus. Raney obtuvo el mismo resultado que el algoritmo (Figura 4.13b) pero tuvo que hacer modificaciones debido a las disonancias y a los distintos tiempos en que terminaron cada una de las voces, obteniendo así la transcripción de la Figura 4.14. Pero casi todos los errores que Raney corrigió en la transcripción están presentes en la partitura mensural: algunas notas mensurales poseen una altura incorrecta (hay dos fa en la voz de la alto que debían ser re y mi) y algunas figuras mensurales fueron omitidas, agregadas o dibujadas incorrectamente (se usaron breves en lugar de dos semibreves en las voces del alto y el bajo, y también había una semibreve extra en la voz del tenor).

Los errores en la partitura mensural se detectan porque no tiene sentido que la pieza original tenga esas disonancias (errores de altura) o esos desfases de tiempo (errores en las figuras); de modo que la partitura mensural no es fiel a la pieza original. Por lo tanto, si la partitura mensural está libre de error, la transcripción generada por el algoritmo, al aplicar mecánicamente las reglas, es correcta, es fiel tanto a la partitura mensural como a la pieza que representa. Si la partitura mensural contiene algún error, aunque la transcripción generada por el modelo algorítmico no sea fiel a la pieza original, esto no elimina la utilidad del modelo. Como se describió en la última sección del capítulo anterior, hay dos fases para la transcripción de una partitura: la primera es transcribirla mediante la aplicación mecánica de las reglas y la segunda consiste en corregir los errores (de altura o de ritmo). El algoritmo se enfoca en la primera fase, permitiendo al experto centrarse únicamente en la segunda.

Como se discutió anteriormente, en teoría se puede desarrollar un algoritmo que trabaje como una función inversa f^{-1}, tomando la transcripción de una partitura mensural y generando la partitura mensural original. Como consecuencia directa de esto un musicólogo sería capaz, teóricamente, de: luego de corregir la transcripción generada por el algoritmo f, utilizar esa transcripción como entrada del algoritmo inverso f^{-1} y obtener la partitura mensural original de la pieza libre de errores.

6 Conclusiones

En este trabajo se desarrolló un algoritmo que modela el proceso de transcripción de las partituras en notación mensural blanca a notación contemporánea. Este algoritmo es una función, cualquier sucesión de notas posee una única transcripción, lo cuál corrobora la consistencia del sistema de notación mensural blanca, no hay ambigüedad al interpretar cualquier sucesión de notas.

El *dominio* del algoritmo consiste en todos los archivos *.ly que, al ser compilados por LilyPond, generan una partitura mensural blanca; dichos archivos *.ly y dichas partituras tienen ciertas restricciones. Las restricciones de los archivos *.ly son: todas las notas escritas en él deben tener indicada su duración (mediante un texto o número), no deben haber líneas en blanco entre las notas pertenecientes a una voz y si se necesita indicar que dos silencios que siguen a una breve pertenecen a distinta perfección se ha de utilizar un *punctus divisiones* entre ellos. Las restricciones de la partitura mensural generada por el archivo *.ly son: que pertenece únicamente a uno de los tres casos de mensuración trabajados (*tempus imperfectum cum prolatione imperfecta*, *tempus perfectum cum prolatione imperfecta* y *tempus imperfectum cum prolatione perfecta*), no posee coloración ni proporciones, si se encuentra en *tempus perfectum cum prolatione imperfecta* todas las voces deben terminar en una longa y no debe haber ninguna otra longa en otro sitio.

El *contradominio* del algoritmo son los archivos de LilyPond que representan cualquier partitura en notación actual. La *imagen* de la función consiste en todos los archivos *.ly que, al ser compilados por LilyPond, generan una partitura que corresponde a la transcripción en notación actual de una partitura mensural blanca.

El algoritmo se comporta como una función inyectiva. No se puede garantizar que la función sea sobreyectiva, por lo cuál tampoco se puede garantizar que se pueda generar una partitura mensural blanca a partir de cualquier partitura en notación actual. Pero si se restringe el *contradominio* del algoritmo a su *imagen*, se garantiza la sobreyectividad y, por lo tanto, se tiene que el algoritmo desarrollado es una función biyectiva. Téoricamente, al tener una función biyectiva, se puede desarrollar otro algoritmo que se comporte como la función inversa; es decir, en teoría se puede realizar un algoritmo que genere una partitura original en notación mensural blanca a partir de su transcripción.

El proceso de transcripción llevado a cabo por un experto tiene dos fases. La primera consiste en transcribir la partitura mediante la aplicación mecánica de las reglas. La segunda consiste en corregir errores en su transcripción, debido a que, después de una análisis armónico y rítmico, el

experto encuentra disonancias o desfases de tiempo entre las voces. Por lo general, los errores que el experto corrige en su transcripción son errores que se encuentran originalmente en la partitura mensural: errores en el altura de las notas o errores al agregar, omitir o sustituir alguna figura (como se observa en la transcripción de M.Mus.Raney discutida en los últimos párrafos del Capítulo 5); y estos errores hacen que la partitura mensural no sea fiel a la pieza original, ya que de lo contrario la pieza presentaría disonancias y desfases de tiempo entre las voces que no son propios de la época. Si la partitura mensural estuviese libre de errores, la transcripción del algoritmo sería fiel tanto a la partitura mensural como a la pieza original. Pero aún si la partitura mensural contiene algún error, el algoritmo facilita la labor del experto realizando por él la primera fase del proceso de transcripción. En teoría, como el algoritmo desarrollado es una función biyectiva f, cabe suponer que si el musicólogo ha corregido ya la transcripción de generada por el algoritmo f, podría usar el algoritmo inverso f^{-1} para generar la partitura mensural original de la pieza libre de errores.

7 Bibliografía

Abromont, Claude, et al. 2005. *Teoría de la música*. México: Fondo de Cultura Económica. 622 págs. p. 20, 33, 38-40, 48-51,56, 61.

Apel, Willi. 2010. <<Part II The notation of ensemble music: white mensural notation>>. *The notation of polyphonic music, 900-1600*. Oxford City Press. págs 83-196. p. 85, 88-92, 96-98.

De León, Francisco y T. Pascual. 1963. [Microforma] *Guatemala music manuscripts*. Bloomington, IN: Lilly Library, Indiana University. 2 rollos; 35 mm. (Manuscritos del repertorio guatemalteco de los años 1570-1635).

Doerr, Alan y K. Levasseur. 1985. <<11.4 Z_n the Integers Modulo n>>. *Applied Discrete Structures for Computer Science*. Estados Unidos: Science Research Associates, Inc. págs. 270-272.

Equipo de desarrolladores de LilyPond. 2012. *Manual LilyPond: Referencia de la notación*.

Equipo de desarrolladores de LilyPond. *Introducción: Entrada de texto*. `http://www.LilyPond.org/text-input.es.html` [14 de octubre de 2013]

Gallian, Joseph. 2010. <<Part 2: Groups >>. *Contemporary Abstract Algebra*. Séptima ed. Estados Unidos: Brooks/Cole, Cengage Learning. págs 27-234. p. 40-41, 138-140, 178-181.

Grabner, Hermann. 2001. *Teoría general de la música*. Decimosexta ed. Barcelona: Ediciones Akal. 338 págs. p. 39-49

Michels, Ulrich. 2009. *Atlas de música*. Madrid: Editorial Alianza. p. 114-115

Niven, Ivan; H. Zuckerman y H. Montgomery. 1991. <<Chapter 2: Congruences >>. *An Introduction to the Theory of Numbers*. 5ta ed. Estados Unidos: John Wiley & Sons, Inc. págs. 47-130.

Oystein, Ore. 1988. <<Chapter 9: Congruences >>. *Number Theory and Its History*. Nueva York: Dover Publications, Inc. págs 209-233. p. 209-215

Pérez, Mariano. 1985. *Diccionario de la música y los músicos*. Barcelona: Ediciones Akal. 406 págs. Vol. 1 p. 282-283, 295

Prather, Ronald. 1986. <<Chapter 4: The Notion of an Algorithm >>. *Elements of Discrete Mathematics*. Boston: Houghton Mifflin Company. págs. 202-275.

Preparata, Franco y R. Yeh. 1974. <<Chapter 7: Algorithms and Turing Machines >>. *Introduction to Discrete Structures for Computer Science and Engineering*. 2a ed. Estados Unidos: Addison-Wesley. págs. 310-341.

Randel, Don Michael. 2003. *The Harvard Dictionary of Music*. Cuarta ed. Cambridge, Massachusetts: Belknap Press. 1008 págs. p. 85

Raney, Sheila. 1981. <<Santa Eulalia M. MD. 7: A critical edition and study of sacred part music from colonial northwestern Guatemala>>. Tesis North Texas State University. 226 págs.

8 Anexos

8.1 Resumen de la teoría musical de la notación actual

El sistema de notación musical que se usa en la actualidad es el mismo que se utilizó durante el barroco (siglo XVII), pero este sistema es el resultado de un largo proceso. La notación anterior es la llamada notación mensural, cuyo desarrollo tuvo origen en la edad media y continuó durante el renacimiento, donde tomó el nombre de notación mensural blanca. Es esta notación, la predecesora de la notación contemporánea.

A continuación se explican algunos elementos importantes de la música y cómo son representados en el sistema de notación actual.

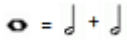
8.1.1. Figuras musicales El sonido es fundamental en la música. Hay muchos aspectos que caracterizan un sonido, por ejemplo: su altura, su duración, su intensidad, su timbre, etc.(Abromont, 2005, p. 20) La notación musical actual permite representar muchos de estos elementos. Con respecto a la duración del sonido, en la música se expresa por medio de figuras musicales. Cabe señalar que en la música es tan importante el sonido como la ausencia del mismo, los silencios; y es muy importante denotar también la duración del silencio. En la actualidad, existen las siguientes figuras musicales:(Abromont, 2005, p. 48-51,56)

Cuadro 8.1: Figuras musicales de la actualidad y su valor

Nombre	Nombre en Inglés	Figura	Silencio	Valor relativo
Redonda	Whole *(1)*			2 blancas
Blanca	Half *(1/2)*			2 negras, la mitad de una redonda
Negra	Quarter *(1/4)*			2 corcheas, la mitad de una blanca
Corchea	Eighth *(1/8)*			2 semicorcheas, la mitad de una negra
Semicorchea	Sixteenth *(1/16)*			2 fusas, la mitad de una corchea
Fusa	Thirty-Second *(1/32)*			2 semifusas, la mitad de una semicorchea
Semifusa	Sixty-Fourth *(1/64)*			la mitad de una fusa

Los nombres en inglés de las figuras son muy útiles porque indican la relación de su valor con respecto a la redonda, la cual se toma como unidad (whole significa totalidad, entero), por ejemplo:

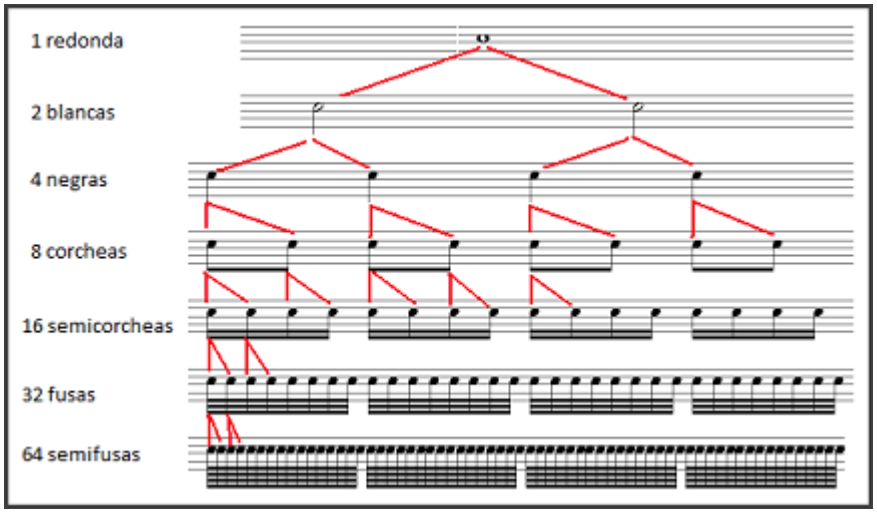
La blanca es llamada half que significa mitad. Por lo tanto, una blanca vale 1/2 de la redonda. Es decir,



La negra es llamada quarter que significa cuarta parte. Por lo tanto, la negra vale 1/4 de la redonda. Es decir,

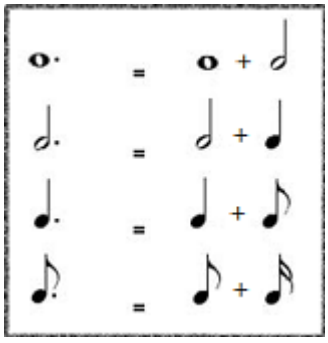
Si se prosigue de este modo, se obtiene:

Figura 8.1: Relación entre las figuras musicales



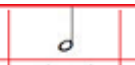
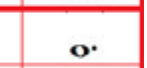
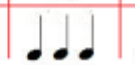
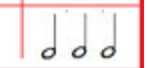
De este modo se observa que la relación entre una figura y su siguiente en menor valor es una relación binaria, tal y como lo indica la Figura 8.1. Esta relación binaria entre figuras consecutivas de distinto grado caracteriza a la notación actual. Una forma de alterar esta relación binaria entre las figuras, es utilizando lo que se conoce como puntillo. El *puntillo* es un punto que se coloca después de la figura, aumentando su valor en un medio del valor original(Abromont, 2005, p. 61). Ejemplos de esto se muestran en la siguiente figura:

Figura 8.2: Figuras con puntillo



Entonces, al aumentarle la mitad de su valor a una figura, el puntillo cambia la relación binaria entre figuras consecutivas a una relación ternaria.

Cuadro 8.2: Relación entre figuras con y sin puntillo

Figuras			
Relación con la siguiente figura de menor valor			
Figuras con Puntillo			
Relación con la siguiente figura de menor valor			

En resumen, las figuras musicales representan la duración de un sonido o un silencio, en la notación actual las figuras y sus silencios manejan una relación binaria. Las figuras musicales contemporáneas son: redonda, blanca, negra, corchea, semicorchea, fusa y semifusa. Al agregarle un puntillo a la figura, esta cambia su relación a ternaria con respecto a la figura que le sigue en menor valor.

8.1.2. Escala diatónica, pentagrama y claves Además de la duración de un sonido, también es importante el aspecto de su altura. La altura se refiere a lo agudo o grave de un sonido. A pesar de que la altura es continua, sólo utilizamos un conjunto discreto de ésta, el cual representamos mediante la secuencia de notas (o tonos) que constituye la llamada escala diatónica, estos tonos son: do, re mi, fa, sol, la y si. En la Figura 8.3 se puede ver esta secuencia de notas en un teclado. Para representar estas notas se utilizan las letras del abecedario (C, D, E, F, G, A, B, respectivamente). Pero si se quiere representar al mismo tiempo el altura y la duración de un sonido, es necesario utilizar las figuras y el pentagrama. El pentagrama es un sistema de 5 líneas horizontales paralelas y equidistantes entre sí.(Abromont, 2005, p. 35)

Figura 8.3: Secuencia de notas en un teclado

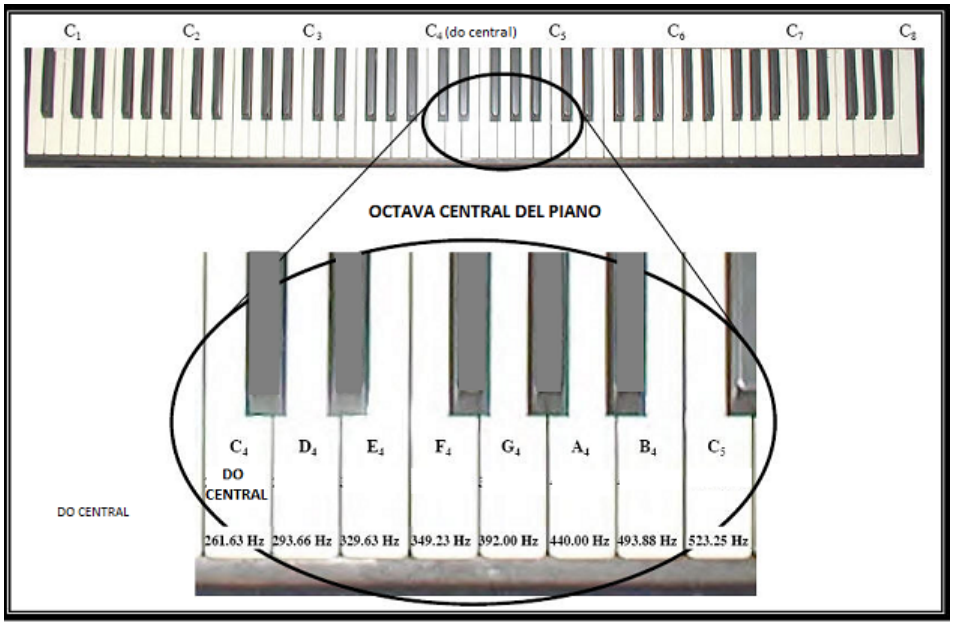
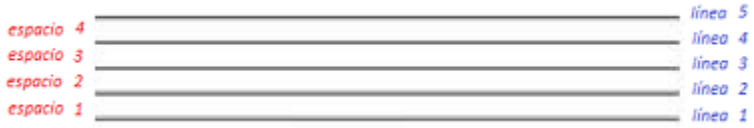


Figura 8.4: Pentagrama, numeración de sus líneas y espacios

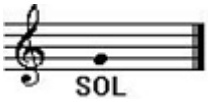


En el pentagrama se colocan las figuras, se pueden posicionar tanto encima de las líneas como en los espacios entre ellas. La altura de un sonido está representada por su ubicación en el pentagrama. Los sonidos agudos se encuentran en las líneas y espacios superiores del pentagrama, los graves en la parte inferior(Abromont, 2005, p. 35). Para saber qué línea o espacio representa una nota determinada, se hace uso de la clave(Pérez, 1985, vol. 1, p. 282-283). Por lo general se utilizan 3 tipos de claves:(Michels, 2009, p. 114-115)

- **Clave de sol:** Generalmente se usa para denotar sonidos agudos. La forma de esta clave es la siguiente:

  La voluta en la parte inferior de la clave va centrada en alguna línea del pentagrama, indicando que esta línea representa la nota sol (G), específicamente el sol cuarto (G4), es decir, el sol perteneciente a la cuarta octava del teclado u octava central. Generalmente, sino es que siempre, esta clave se centra en la segunda línea del pentagrama.(Abromont, 2005, p. 38-40)

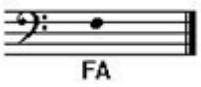
  

  Entonces, las notas más agudas que sol se colocan en los espacios y líneas superiores a la segunda línea (siguiendo siempre el orden de la secuencia de notas: do, re, mi, fa, sol, la, si), y las notas más graves se colocan en las líneas y espacios inferiores a la segunda línea. Entonces, todas las otras notas se posicionan en los demás espacios y líneas del pentagrama una vez ubicada la nota sol en la segunda línea, tal y como lo muestra la Figura 8.5.

- **Clave de fa:** Generalmente se usa para denotar sonidos graves. La forma de esta clave es la siguiente:

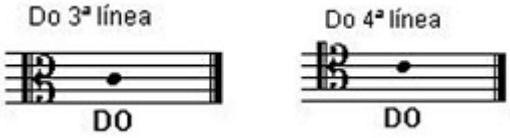
Su voluta está centrada, por lo general, en la cuarta línea del pentagrama, indicando que la nota ubicada en esta línea es fa (F), cabe señalar que este fa es el fa tercero (F3), perteneciente a la tercera octava del teclado.(Abromont, 2005, p. 38-40)



En base a que fa se encuentra en la cuarta línea, entonces en los espacios y líneas superiores son: sol, la, si, do, re, etc. Y las notas ubicadas en los espacios y líneas inferiores en orden descendente son: mi, re, do, si, la, etc. Para observar la distribución de las notas en el pentagrama con clave de fa, ver Figura 8.5.

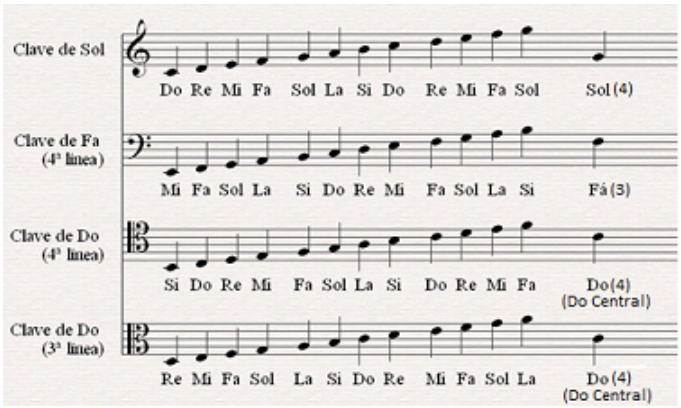
- **Clave de do:** Generalmente se usa para denotar sonidos intermedios. La forma de la clave es la siguiente:

Esta clave puede centrarse en cualquier línea del pentagrama, indicando que esta línea representa la nota do (C), éste es el do central o C4 (de la cuarta octava u octava central del teclado). Por lo general, se usa la clave de do de tercera o cuarta línea, aunque, como ya se mencionó antes, se puede centrar en cualquier línea.(Abromont, 2005, p. 38-40)



Para observar la distribución de las notas en el pentagrama con clave de do, ver Figura 8.5.

Figura 8.5: Distribución de las notas en el pentagrama, en función a la clave



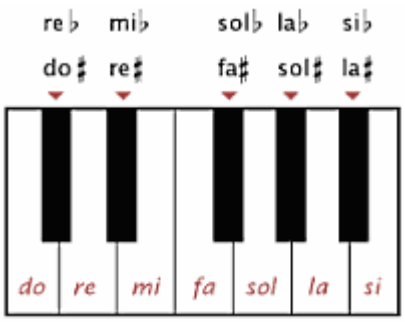
8.1.3. Alteraciones cromáticas de las notas de la escala diatónica

Las notas pueden poseer alteraciones cromáticas (bemoles o sostenidos). En el teclado, las notas simples (do, re, mi, fa, sol, la, si) se encuentran en las teclas blancas, y las notas alteradas en las teclas negras, éstas son:(Abromont, 2005)

- do sostenido (C#) o re bemol (Db)
- re sostenido (D#) o mi bemol (Eb)
- fa sostenido (F#) o sol bemol (Gb)
- sol sostenido (G#) o la bemol (Ab)
- la sostenido (A#) o si bemol (Bb)

En la siguiente figura se muestran las 7 notas y sus alteraciones cromáticas en un teclado.

Figura 8.6: Distribución de las notas y sus alteraciones cromáticas en un teclado



8.1.4. Compás Falta mencionar un elemento rítmico importante, además de la duración de las notas, éste es el compás. La música se divide en períodos de tiempo iguales, marcando el ritmo musical, a cada uno de estos períodos se le conoce como compás. También se le llama así a cada una de las divisiones del pentagrama en que se representan estos períodos en la partitura musical, dichas divisiones se realizan con líneas verticales llamadas barras de compás.(Pérez, 1985, vol. 1, p.295)

El compás es una entidad métrica compuesta por varias unidades de tiempo que se organizan en grupos, en los cuáles se da una contraposición entre partes acentuadas y átonas.(Grabner, 2001, p. 39-49) La indicación de compás se coloca al inicio de la partitura (o luego de una doble barra que indica un cambio en el tipo de compás), y es representado por dos números colocados en forma de fracción, estos números especifican cuántos pulsos o tiempos posee el compás y qué figura musical define cada pulso.(Randel, 2003, p. 85) A continuación se presentan algunos ejemplos de distintos tipos de compás, esto aclarará la explicación anterior, aunque hay muchos tipos de compás, se presentan los que será útiles para el presente trabajo.

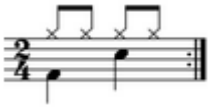
- **Compás 2/4**

  El numerador indica que es un compás binario, es decir, un compás de 2 tiempos, el primero de ellos acentuado y el segundo no. El denominador indica que la unidad de tiempo del compás es la negra (quarter)

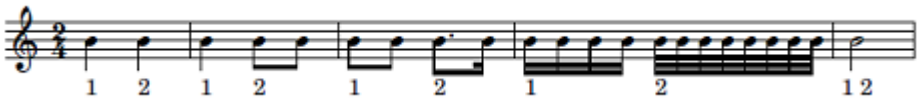
  Este compás posee 2 tiempos de negra. Se puede pensar en la fracción 2/4 de la siguiente manera:

$$\frac{2}{4} = 2 \times \frac{1}{4} = 2 \times \text{quarter}$$

  El compás 2/4 es un compás binario de subdivisión binaria (el compás posee dos negras, que se pueden dividir en dos corcheas cada una), como se muestra a continuación:



  Un ejemplo de este compás es:



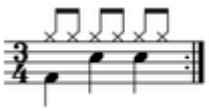
- **Compás 3/4**

  El numerador indica que es un compás ternario, es decir, un compás de 3 tiempos, el primero acentuado y el segundo y tercero sin acentuar. El denominador indica que la unidad de tiempo del compás es la negra (quarter).

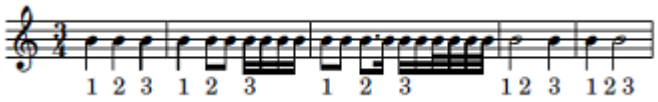
  Este compás posee 3 tiempos de negra. Se puede pensar en la fracción 3/4 de la siguiente manera:

$$\frac{3}{4} = 3 \times \frac{1}{4} = 3 \times \text{quarter}$$

  El compás 3/4 es un compás ternario de subdivisión binaria (el compás posee tres negras, que se pueden dividir en dos corcheas cada una), como se muestra a continuación:



Un ejemplo de este tipo de compás es:



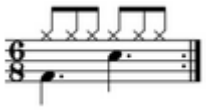
- **Compás 6/8**

  El numerador indica que hay 6 fracciones, dos tiempos con tres fracciones cada uno. El denominador indica que cada fracción es una corchea (eighth).

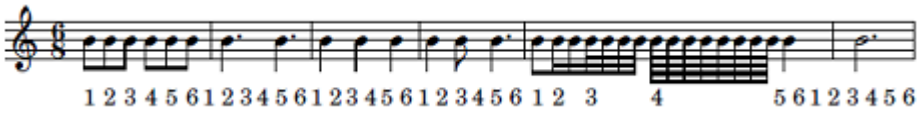
  Este compás posee 6 tiempos de corchea. Se puede pensar en la fracción 6/8 de la siguiente manera:

$$\frac{6}{8} = 6 \times \frac{1}{8} = 6 \times \text{eighth}$$

  El compás de 6/8 es un compás binario de subdivisión ternaria (dos tiempos con tres fracciones cada uno), como se muestra a continuación:



  Un ejemplo de esto es:



- **Compás 9/8**

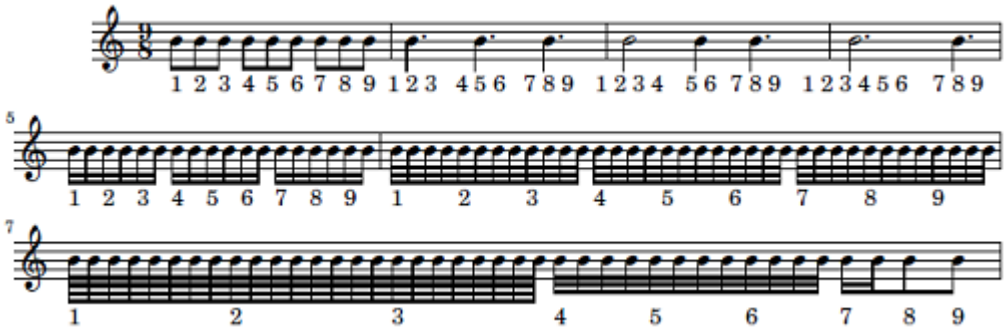
  El numerador indica que hay 9 fracciones, tres tiempos con tres fracciones cada uno. El denominador indica que cada fracción es una corchea (eighth).

  Este compás posee 9 tiempos de corchea. Se puede pensar en la fracción 9/8 de la siguiente manera:

$$\frac{9}{8} = 9 \times \frac{1}{8} = 9 \times \text{eighth}$$

El compás 9/8 es un compás ternario de subdivisión ternaria (tres tiempos con tres fracciones cada uno), como se muestra a continuación:

Un ejemplo de esto es:



8.2 Notación mensural blanca

La notación mensural es un sistema de notación musical utilizado en Europa desde finales del siglo XIII hasta terminado el siglo XVI. En la sección anterior de Notación actual se indicó que dos elementos importantes del sonido son su altura y su duración. El problema de indicar el altura de un sonido, se resolvió antes de que se desarrollara la música polifónica; la novedad del sistema de notación mensural fue su capacidad de describir con gran precisión la duración de una nota y la relación entre los valores de tiempo de distintas notas, resolviendo el problema rítmico.(Apel, 2010, p. 85)

La notación mensural blanca es el sistema de notación mensural utilizado a mediados del siglo XV y hasta el final del siglo XVI. El término "mensural" proviene de "musica mensurata", nombre que utilizaron los teóricos medievales para referirse a la música polifónica por ser ésta bien definida rítmicamente, a diferencia de la "musica plana" del canto gregoriano que carecía de medida. El término "blanca" se refiere al uso de figuras blancas para las notas de valores más grandes, en lugar de figuras negras como se utilizaba en la notación anterior (mensural negra, notación mensural del finales del siglo XIII a mediados del siglo XV).

8.2.1. Escala diatónica, pentagrama y claves Al ver por vez primera una partitura mensural, uno podría pensar que es un lenguaje incomprensible. Sin embargo, la notación mensural tiene elementos comunes con la notación contemporánea. En la notación mensural ya se hace uso del pentagrama y de la escala diatónica (do, re, mi, fa, sol, la, si), también se usan las tres claves que se mencionaron en la sección de Notación Actual (clave de sol, clave de fa y clave

de do) para indicar la ubicación de cada una de las notas de la escala en las líneas y espacios del pentagrama. La grafía de las claves, sin embargo, es diferente; a continuación se muestra cada una de éstas: (Manual de notación en LilyPond, 2012, p. 16-17)

- **Clave de Sol:** Por lo general, la clave de sol se centra en la segunda línea del pentagrama (igual que en la notación actual), tal y como lo muestra la siguiente imagen.

- **Clave de Fa:** Por lo general, la clave de fa se centra en la cuarta línea del pentagrama (igual que en la notación actual), a continuación se muestran dos formas comunes de representar la clave de fa.

- **Clave de Do:** La clave de do puede ir centrada en cualquiera de las cinco líneas del pentagrama, indicando que esa línea es la nota do (específicamente, do central). Aunque hay muchas formas para la clave de do, las más comunes son las representadas en las siguientes imágenes. En estas tres imágenes, la clave de do esta centrada en la tercera línea.

Las similitudes entre la notación actual y mensural se encuentran en el aspecto melódico (las notas, el pentagrama y las claves que ubican dichas notas en el pentagrama). Sin embargo, en el aspecto rítmico es donde ambas notaciones difieren.

8.2.2. Figuras musicales Las figuras y silencios musicales utilizados en la notación mensural se muestran en el Cuadro 8.3, están ordenadas de mayor a menor valor.

Cuadro 8.3: Figuras musicales de la notación mensural

Nombre	Figura	Silencio
Máxima		
Longa		
Breve		
Semibreve		
Mínima		
Semimínima		
Fusa		
Semifusa		

En el Cuadro 8.3 no se colocó el valor entre las figuras, como se hizo con las de notación actual en el Cuadro 8.1, esto es porque una gran diferencia que hay entre las figuras de notación contemporánea y las figuras de notación mensural es que el valor de éstas últimas no necesariamente es binario. Esto se explicará posteriormente.

8.2.3. Ligaduras Además de notas individuales, la notación mensural emplea símbolos que representan combinaciones de dos o más tonos, estos símbolos son conocidos como ligaduras. Las ligaduras provienen de dos neumas: clivis y podatus; el primero representa un par de notas en movimiento descendente y el segundo en movimiento ascendente. En 1150 la forma de los neumas cambió a la llamada notación coral romana:(Apel, 2010, p. 88)

clivis:

podatus:

En el año 1200 se usaron los mismos símbolos para escribir la música polifónica, y adquirieron ciertos valores métricos. En las dos ligaduras la primera nota tomó el valor de una breve y la segunda de una longa. Sin embargo, para expresar las demás combinaciones de ambas figuras (breve y longa), se desarrollaron los términos: proprietas y perfectio. El primer término se refiere

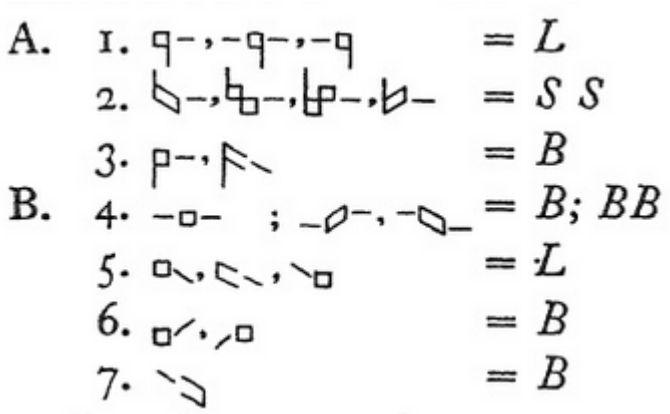
a la primera nota: cum proprietate (con propiedad) implica que la primera nota es una breve y sine proprietate (sin propiedad) implica que es una longa. El segundo término se refiere a la segunda nota: cum perfectione (con perfección) implica que segunda nota es una longa y sine perfectione implica que es una breve. Las cuatro combinaciones tienen sus propios símbolos tanto en clivis como en podatus, y se indican en la siguiente imagen (junto con otra ligadura utilizada en la notación mensural blanca llamada ligadura cum opposita proprietate):(Apel, 2010, p. 88-90)

Cuadro 8.4: Ligaduras de la notación mensural blanca

NOMBRE	VALOR	FORMA	
		desc.	asc.
cum proprietate et cum perfectione	*B L*		
sine proprietate et cum perfectione	*L L*		
cum proprietate et sine perfectione	*B B*		
sine proprietate et sine perfectione	*L B*		
cum opposita proprietate	*S S*		

En el Cuadro 8.4 se observan todas las clases de ligaduras de dos notas; sin embargo, pueden ligarse más de dos notas. Para cualquier tipo de ligaduras deben seguirse algunas reglas para determinar el valor de cada nota incluida en la ligadura. Estos lineamientos se muestran en la imagen siguiente, en donde el inciso A se refiere a las reglas sobre el significado de las colas, y el inciso B a las reglas para las notas no cubiertas en el apartado A. Estas reglas deben seguirse en el orden indicado en la imagen.(Apel, 2010, p. 92)

Figura 8.7: Reglas para la determinación del valor de cada nota perteneciente a una ligadura



8.2.4. Mensuración Como se indicó anteriormente, las figuras de notación mensural no necesariamente poseen un valor binario, como es el caso de las figuras de la notación actual.

Una figura mensural sin puntillo puede tener un valor binario o ternario dependiendo del contexto. La mensuración es precisamente esa relación métrica entre el valor de una nota y el de la siguiente de menor grado. Una figura ternaria se dice que tiene mensuración perfecta, y una binaria tiene mensuración imperfecta. En la notación mensural blanca la elección de mensuración perfecta o imperfecta existe, por lo general, en el caso de dos figuras: la breve y la semibreve; las demás son imperfectas (de valor binario). Estas relaciones se muestran en el siguiente cuadro. Cabe señalar que lo anterior sólo se aplica a las figuras, no a los silencios, el valor de los silencios siempre es binario.(Apel, 2010, p. 96)

Cuadro 8.5: Valor relativo de las figuras mensurales

Nombre	Figura	Valor
Máxima		2 longas
Longa		2 breves
Breve		2 o 3 semibreves
Semibreve		2 o 3 mínimas
Mínima		2 semimínimas
Semimínima		2 fusas
Fusa		2 semifusas
Semifusa		-

La mensuración de la breve es llamada Tempus y existe en las dos variedades:(Apel, 2010, p. 96)

- Tempus perfectum, representada por una circunferencia
- Tempus imperfectum, representada por una media circunferencia abierta por el lado derecho.

La mensuración de la semibreve es llamada Prolatio, y también existe en las dos variedades:(Apel, 2010, p. 96)

- Prolatio perfecta, representada por un punto ubicado al centro de la circunferencia o media circunferencia
- Prolatio imperfecta, representada por la ausencia del punto.

Con las variaciones de Tempus y Prolatio se tienen cuatro posibilidades:(Apel, 2010, p.96-97)

- Tempus imperfectum cum prolatione imperfecta

  También se puede indicar por [2,2]

- Tempus perfectum cum prolatione imperfecta

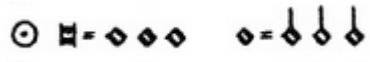
  También se puede indicar por [3,2]

- Tempus imperfectum cum prolatione perfecta

  También se puede indicar por [2,3]

- Tempus perfectum cum prolatione perfecta

  También se puede indicar por [3,3]



Nótese que debido a las divisiones y subdivisiones, tanto binarias como ternarias de estas mensuraciones, se puede pensar en ellas en forma similar a los compases señalados en la sección Notación actual. Si se piensa en transcribir la semibreve como una negra, se obtiene lo siguiente:

- Tempus imperfectum cum prolatione imperfecta:

  La breve equivale a 2 semibreves, y la semibreve a su vez equivale a 2 mínimas. Por lo tanto, puede pensarse en un compás de división binaria con subdivisión binaria, como es el caso del compás 2/4.

- Tempus imperfectum cum prolation perfecta:

  La breve equivale a 2 semibreves, y la semibreve a 3 mínimas. Por lo tanto, puede pensarse como un compás de división binaria con subdivisión ternaria, como es el caso del compás 6/8

- Tempus perfectum cum prolatione imperfecta:

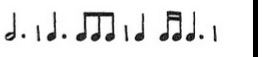
  La breve equivale a 3 semibreves, y la semibreve a 2 minimas. Por lo tanto, puede pensarse como un compás de división ternaria con subdivisión binaria, como es el caso del compás 3/4.

- Tempus perfectum cum prolatione perfecta:

  La breve equivale a 3 semibreves, y la semibreve equivale a 3 mínimas. Por lo tanto, puede pensarse como un compás de división ternaria con subdivisión ternaria, como es el caso del compás 9/8.

Lo anterior se resume en el siguiente cuadro:(Apel, 2010, p. 97-98)

Cuadro 8.6: Mensuración y métrica

MENSURACIÓN	MEDIDA (indicacor de compás)	TRANSCRIPCIÓN DE NOTAS	EJEMPLO
C [2, 2]	2/4		
C· [2, 3]	6/8		
O [3, 2]	3/4		
O· [3, 3]	9/8		

Es claro que algunas figuras tienen carácter binario y otras ternario dependiendo de la mensuración; sin embargo, incluso cuando se ha establecido un tipo de mensuración, hay factores que inducen la perfección o imperfección de una figura. En el proceso de transcripcion de una partitura mensural a notación actual, el transcribir cada nota obedeciendo a la mensuración (si es un Tempus Perfectum cum Prolatione Imperfecta, transformar cada breve en una blanca con puntillo y cada semibreve en una negra) produce una pieza en notación actual que no es fiel en tiempo a la pieza original en notación mensural. Muchos musicólogos se han dado cuenta de este problema: seguir la mensuración al pie de la letra al transcribir una pieza, a veces produce errores en el resultado final. En ocasiones las notas perfectas deben ser imperfeccionadas, y en otras, las imperfectas deben ser perfeccionadas; estos teóricos se han dado a la tarea de buscar reglas generalizadoras para estas circunstancias especiales, y han avanzado mucho al respecto. El fin de este trabajo, es formalizar aún más sus hallazgos y buscar otras reglas para excepciones que aún no se han formalizado, creando así un sistema formal para la transcripción de partituras de notación mensural a notación contemporánea, que permita eventualmente automatizar esta transformación.

8.3 Implementación del algoritmo en python

El algoritmo supone que la partitura mensural que se va a transcribir no tiene ningún error, pero es común que los copistas cometieran algunos errores al escribir las partituras mensurales. El siguiente código python es una implementación del algoritmo, pero posee instrucciones extras para la detección de errores en la partitura que imposibilitan el proceso de transcripción según las reglas. Estas instrucciones señalan qué ocasiona el error y detienen el proceso de transcripción,

pero no sin previamente mostrar la sucesión de notas que sí pudieron transcribirse antes de llegar a la nota que ocasionó el fallo en dicho proceso.

```
##########################################################################
#  FUNCIONES UTILIZADAS EN LOS DISTINTOS METODOS DE TRANSCRIPCION  #
##########################################################################
def GetNoteValue(note):
    MVALUE = ['16.', '8.', '4.', '2.', '1.', '\\breve.', '\\longa.',
    '\\maxima.', '16', '8', '4', '2', '1', '\\breve', '\\longa', '\\maxima']
    mvalue = '4'
    for i in range(0, len(MVALUE)):
        if MVALUE[i] in note:
            mvalue = MVALUE[i]
            break
    return mvalue

def GetNote(note):
    mvalue = GetNoteValue(note)
    list_of_characters = list(note)
    start_mvalue = note.index(mvalue)
    end_mvalue = start_mvalue + len(mvalue)
    for j in range (start_mvalue, end_mvalue):
        list_of_characters.pop(start_mvalue)
    just_note = ''.join(list_of_characters)
    return just_note

def ChangeNoteValue(note, mensural_value, actual_value):
    list_of_characters = list(note)
    start = note.index(mensural_value)
    end = start + len(mensural_value)
    list_of_characters[start:end] = actual_value
    word = ''.join(list_of_characters)
    return word

def TimeSemibreve(mnote):
    MVALUE = ['16.', '8.', '4.', '2.', '1.', '16', '8', '4', '2', '1']
    TIMESEMIB = [3/32, 3/16, 3/8, 3/4, 3/2, 1/16, 1/8, 1/4 , 1/2 , 1]
```

```
    time_semib = 0.25
    for i in range (0, len(MVALUE)):
        if MVALUE[i] in mnote:
            time_semib = TIMESEMIB[i]
            break
    return time_semib

def GetActualValue(mnote):
    MVALUE = ['16.', '8.', '4.', '2.', '1.', '16', '8', '4', '2', '1']
    AVALUE = ['64.', '32.', '16.', '8.', '4.', '64', '32', '16', '8', '4']
    avalue = '16'
    for i in range(0, len(MVALUE)):
        if MVALUE[i] in mnote:
            avalue = AVALUE[i]
            break
    return avalue

def Remove(word, char):
    word_list = list(word)
    word_list.remove(char)
    new_word = ''.join(word_list)
    return new_word

def AddLigature(note, ligature):
    if('\\]' in note):
        word_list = list(note)
        word_list.insert( word_list.index(']') - 1 , ligature)
        word = ''.join(word_list)
    else:
        word = note + ligature
    return word
##########################################################################################
# METODO DE TRANSCRIPCION PARA TEMPUS IMPERFECTUM CUM PROLATIONE IMPERFECTA #
##########################################################################################
def TranscripteII(MNOTES, new_file):
    MVALUE = ['16.', '8.', '4.', '2.', '1.', '\\breve.', '\\longa.',
```

```
    '\\maxima.', '16', '8', '4', '2', '1', '\\breve', '\\longa', '\\maxima']
    AVALUE12 = ['32.', '16.', '8.', '4.', '2.', '1.', '\\breve.',
    '\\longa.', '32', '16', '8', '4', '2', '1', '\\breve', '\\longa']
    AVALUE14 = ['64.', '32.', '16.', '8.', '4.', '2.', '1.',
    '\\breve.', '64', '32', '16', '8', '4', '2', '1', '\\breve']
    ratio = input("Insert the ratio (1:1, 1:2 or 1:4): ")
    while (not (ratio == '1:1' or ratio == '1:2' or ratio == '1:4')):
        ratio = input("Mistake, please insert the ratio (1:1, 1:2 or 1:4): ")
    if ratio == '1:1':
        AVALUE = MVALUE
    elif ratio == '1:2':
        AVALUE = AVALUE12
    else:
        AVALUE = AVALUE14
    ANOTE = []
    for mnote in MNOTES:
        for i in range (0, len(MVALUE)):
            if MVALUE[i] in mnote:
                mvalue = MVALUE[i]
                avalue = AVALUE[i]
        anote = ChangeNoteValue(mnote, mvalue, avalue)
        ANOTE.append(anote)
        new_file.write(anote + ' ')
        print("Se transcribio correctamente la siguiente sucesion de notas: ",
        ANOTE)
################################################################################
# METODO DE TRANSCRIPCION PARA TEMPUS PERFECTUM CUM PROLATIONE IMPERFECTA #
################################################################################
def TranscriptePI(NOTE, new_file):
        ## Condiciones de inicio (primera nota)
    start = 0
    #mvalue = '4' a falta de duracion en la primera nota,
    #esta tiene el valor 4.

    NEW_NOTE = []
```

```
## Determinacion de los indices donde hay notas breves
## en una determinada voz mensural
INDEXBREVES = []
for i in range (0, len(NOTE)):
    if "\\breve" in NOTE[i]:
        INDEXBREVES.append(i)
INDEXBREVES.append(len(NOTE)-1)
#print(INDEXBREVES)


for j in range (0, len(INDEXBREVES)):
    if INDEXBREVES[j] > start:
        end = INDEXBREVES[j]
        break
#print("end: ", end)

while(start < end):
    NU = []
    ## Sucesion NU
    for j in range (start, end+1):
        NU.append(NOTE[j])
    print("La sucesion nu a transcribir es: ", NU)

    NEW_NU = []

    ## CONDICIONES DE TRANSCRIPCION:

    # 0 -> Punctus Perfectionis: Si NU[0] es breve con punto ("\\breve.") #
    if ( GetNoteValue(NU[0]) == '\\breve.' ):
        print("Punctus divisionis.")
        NEW_NU.append( ChangeNoteValue( NU[0], '\\breve.', '2.' ) )
        start = start + 1

    # 1 -> Si NU[0] es un silencio mayor o igual a una breve #
    elif (('r' in GetNote(NU[0])) and
    ((GetNoteValue(NU[0]) == '\\breve')
```

```
or (GetNoteValue(NU[0]) == '\\longa')
or (GetNoteValue(NU[0]) == '\\maxima'))):

    if ( GetNoteValue(NU[0]) == '\\breve' ):
        NEW_NU.append( ChangeNoteValue(NU[0], '\\breve', '2.') )

    elif ( GetNoteValue(NU[0]) == '\\longa' ):
        NEW_NU.append( ChangeNoteValue(NU[0], '\\breve', '2.') )
        NEW_NU.append( ChangeNoteValue(NU[0], '\\breve', '2.') )

    else:
        NEW_NU.append( ChangeNoteValue(NU[0], '\\breve', '2.') )
        NEW_NU.append( ChangeNoteValue(NU[0], '\\breve', '2.') )
        NEW_NU.append( ChangeNoteValue(NU[0], '\\breve', '2.') )
        NEW_NU.append( ChangeNoteValue(NU[0], '\\breve', '2.') )

    start = start + 1

# 2 -> Si NU[0] es un sonido de breve #
elif ((not('r' in GetNote(NU[0]))) and
(GetNoteValue(NU[0]) == '\\breve')):
    k = end - start
    t = 0

    if (k > 1):
        for j in range(1, k):
            t = t + TimeSemibreve(NU[j])
            #Tiempos de semibreve de todas las notas de la sucesion
            #cuyo valor es menor a la breve

    # 2.1 -> Si t es un numero ENTERO (Z), es decir,
    # NO hay PUNCTUS DIVISIONIS #
    if((int(t)) == t):
        r = t % 3

        # 2.1.1 #
```

```
if (r == 1):
    print("Imperfeccion a.p.p.")
    NEW_NU.append( ChangeNoteValue(NU[0], '\\breve', '2') )
    for i in range(1, (k-1)+1):
        NEW_NU.append( ChangeNoteValue( NU[i],
        GetNoteValue(NU[i]), GetActualValue(NU[i]) ) )
    start = end        #Inicia en la nota denotada por NU[k]

# 2.1.2 #
elif (r == 2):
    # 2.1.2.1 >> Exactamente 2 semibreves entre breves #
    if(t == 2):
        # Opcion predilecta: Alteracion;
        # pero hay circunstancias que la eliminan:

        # EXCEPCION (para la alteracion)
        # Lo que implica: Imperfeccion a.p.p. y a.p.a. #
        ## Para Alteracion
        ##### Si la NU[k-1] es sonido diferente a semibreve
        ##### O si NU[k-1] es un silencio
        if( ( not( GetNoteValue(NU[k-1]) == '1' ) )
        or ( 'r' in GetNote(NU[k-1]) ) ):
            ########## BLOQUE DE ERROR ##########
            # Excepcion (para la imperfeccion) #
            if( 'r' in GetNote(NU[k]) ):
                print ("ERROR, no se puede imperfeccionar
                (porque la ultima breve de la sucesion es
                silencio, y debe ser perfecto) ni alterar
                (por el valor de la penultima nota de la
                sucesion)")
                break

            # Ambas condiciones (esta y la de abajo) son
            # importantes para que haya una breve que siga
            # a la de END
            elif(len(NOTE) > end + 1):
```

```
                    # Esto permite que se pueda evaluar la condicion
                    # de abajo ( utiliza NOTE[end + 1] )
                        if ( '\\breve' in NOTE[end+1] ):
                        # Cuando una breve sigue a la ultima breve
                        # no se puede imperfeccionar
                            print ("ERROR, no se puede imperfeccionar
                            (porque la ultima breve de la sucesion
                            va seguida de otra breve) ni alterar
                            (por el valor de la penultima nota de
                            la sucesion)")
                            break
                    ######################################################
                    print("Imperfeccion a.p.p. y a.p.a.")
                    NEW_NU.append(ChangeNoteValue(NU[0],'\\breve','2'))
                    for i in range(1, (k-1)+1):
                        NEW_NU.append(ChangeNoteValue(NU[i],
                        GetNoteValue(NU[i]),GetActualValue(NU[i])))
                    NEW_NU.append(ChangeNoteValue(NU[k],
                    GetNoteValue(NU[k]),'2'))
                    start = end + 1
                    #Inicia en la nota denotada por NU[k+1]

                ## DEFAULT -> Alteracion ##
                else:
                    print("Alteracion.")
                    NEW_NU.append(ChangeNoteValue(NU[0],'\\breve','2.'))
                    for i in range(1, (k-2)+1):
                        NEW_NU.append(ChangeNoteValue(NU[i],
                        GetNoteValue(NU[i]), GetActualValue(NU[i])))
                    NEW_NU.append( ChangeNoteValue(NU[k-1],'1','2'))
                    start = end #Inicia en la nota denotada por NU[k]


            # 2.1.2.2 >> 5, 8, 11, etc. semibreves entre breves #
            else:
                # Opcion predilecta: Imperfeccion a.p.p. y a.p.a.
```

```
# pero hay circunstancias que la eliminan:

# EXCEPCION (para la imperfeccion -a.p.p. y a.p.a.-)
# Lo que implica: Alteracion #
##Para Imperfeccion a.p.p.
##### Si la primera breve es silencio (pero esto no
#####pasa por la condicion [1])
## Para Imperfeccion a.p.a.
##### Si la ultima breve es silencio (NU[k])
##### Si la nota siguiente a la ultima breve es
#####tambien breve (NOTE[end + 1], si existe, claro)
## Tambien es imperativa la Alteracion cuando hay
##ligadura c.o.p.
if ( 'r' in GetNote(NU[k]) ):
    ############### BLOQUE DE ERROR ##############
    # Excepcion (para la alteracion)           #
    if( ( not( GetNoteValue(NU[k-1]) == '1' ) )
    or ( 'r' in GetNote(NU[k-1]) ) ):
        print ("ERROR, no se puede imperfeccionar
        (por el silencio de breve al final de la
        sucesion a transcribir, este debe ser
        siempre perfecto) ni alterar (por el
        valor de la penultima nota)")
        break
    ##############################################
    print("Alteracion.")
    NEW_NU.append(ChangeNoteValue(NU[0],'\\breve','2.'))
    for i in range(1, (k-2)+1):
        NEW_NU.append(ChangeNoteValue(NU[i],
        GetNoteValue(NU[i]), GetActualValue(NU[i])))
    NEW_NU.append(ChangeNoteValue(NU[k-1],'1','2'))
    start = end #Inicia en la nota denotada por NU[k]
elif(len(NOTE) > end + 1):
# Cuando una breve sigue a la ultima breve de la
# sucesion (no se puede imperfeccionar, y se opta
# por alterar)
```

```
                if ( '\\breve' in NOTE[end+1] ):
                    ########## BLOQUE DE ERROR ############
                    # Excepcion (para la alteracion)       #
                    if( (not(GetNoteValue(NU[k-1]) == '1'))
                    or ('r' in GetNote(NU[k-1])) ):
                        print ("ERROR, no se puede imperfeccionar
                        (porque hay otra breve luego de la
                        sucesion a transcribir) ni alterar (por
                        el valor de la penultima nota)")
                        break
                    ########################################
                    print("Alteracion.")
                    NEW_NU.append(ChangeNoteValue(NU[0],'\\breve',
                    '2.'))
                    for i in range(1, (k-2)+1):
                        NEW_NU.append(ChangeNoteValue(NU[i],
                        GetNoteValue(NU[i]),GetActualValue(NU[i])))
                    NEW_NU.append(ChangeNoteValue(NU[k-1],'1','2'))
                    start = end
                    #Inicia en la nota denotada por NU[k]
                else:
                    print("Imperfeccion a.p.p. y a.p.a.")
                    NEW_NU.append(ChangeNoteValue(NU[0],'\\breve',
                    '2'))
                    for i in range(1, (k-1)+1):
                        NEW_NU.append(ChangeNoteValue(NU[i],
                        GetNoteValue(NU[i]),GetActualValue(NU[i])))
                    NEW_NU.append(ChangeNoteValue(NU[k],
                    GetNoteValue(NU[k]),'2'))
                    start = end + 1
                    #Inicia en la nota denotada por NU[k+1]
            elif( ']' in NU[k-1] ):
                print("Alteracion.")
                NEW_NU.append(ChangeNoteValue(NU[0],'\\breve','2.'))
                for i in range(1, (k-2)+1):
                    NEW_NU.append(ChangeNoteValue(NU[i],
```

```
                    GetNoteValue(NU[i]),GetActualValue(NU[i])))
                NEW_NU.append( ChangeNoteValue(NU[k-1],'1','2'))
                start = end
                #Inicia en la nota denotada por NU[k]

            ## DEFAULT -> Imperfeccion a.p.p. y a.p.a. ##
            else:
                print("Imperfeccion a.p.p. y a.p.a.")
                NEW_NU.append(ChangeNoteValue(NU[0],'\\breve','2'))
                for i in range(1, (k-1)+1):
                    NEW_NU.append(ChangeNoteValue(NU[i],
                    GetNoteValue(NU[i]),GetActualValue(NU[i])))
                NEW_NU.append(ChangeNoteValue(NU[k],
                GetNoteValue(NU[k]),'2'))
                start = end + 1
                #Inicia en la nota denotada por NU[k+1]


# 2.1.3 #
if (r == 0):

    # 2.1.3.1 -> breve, 0 semibreves, breve; es decir:
    # breve seguida de breve #
    if (t == 0):
        print("Se mantiene perfecta.")
        NEW_NU.append(ChangeNoteValue(NU[0],'\\breve','2.'))
        start = end

    # 2.1.3.2 -> breve, 3 semibreves, breve #
    elif (t == 3):
        print("Se mantiene perfecta.")
        NEW_NU.append(ChangeNoteValue(NU[0],'\\breve','2.'))
        for i in range(1, (k-1)+1):
            NEW_NU.append(ChangeNoteValue(NU[i],
            GetNoteValue(NU[i]),GetActualValue(NU[i])))
        start = end
```

```
# 2.1.3.3 -> semibreves entre breves: 6, 9, 12, etc. #
else:
    # Opcion predilecta: Imperfeccion a.p.p. y Alteracion
    # pero hay circunstancias que no lo permiten:

    # EXCEPCION (para Imperfeccion a.p.p. o  para la
    # Alteracion), lo que implica: Perfeccion
    ## Para Imperfeccion a.p.p.
    ##### Si la primera breve es silencio
    #####(pero esto no pasa por la condicion [1])
    ## Para Alteracion
    ##### Si la NU[k-1] es sonido diferente a semibreve
    ##### O si NU[k-1] es un silencio
    if( ( not( GetNoteValue(NU[k-1]) == '1' ) )
    or ( 'r' in GetNote(NU[k-1]) ) ):
        print("Se mantiene perfecta (son mas de tres
        semibreves pero la alteracion no puede darse).")
        NEW_NU.append(ChangeNoteValue(NU[0],'\\breve','2.'))
        for i in range(1, (k-1)+1):
            NEW_NU.append(ChangeNoteValue(NU[i],
            GetNoteValue(NU[i]), GetActualValue(NU[i])))
        start = end

    ## DEFAULT >> Imperfeccion a.p.p. y Alteracion
    else:
        print("Imperfeccion a.p.p. y Alteracion.")
        NEW_NU.append(ChangeNoteValue(NU[0],'\\breve',
        '2')) #Imperfeccion a.p.p.
        for i in range(1, (k-2)+1):
            NEW_NU.append(ChangeNoteValue(NU[i],
            GetNoteValue(NU[i]), GetActualValue(NU[i])))
        NEW_NU.append(ChangeNoteValue(NU[k-1],
        GetNoteValue(NU[k-1]), '2')) # Alteracion
        start = end
```

```
# 2.2 >>        Si t NO es numero ENTERO (Z), es decir,
# SI HAY PUNCTUS DIVISIONIS #
else:
    k = end - start
    t_s1 = 0
    t_s2 = 0
    ind_punto = 0

    for i in range (0, k):
        # Si hay un "puntillo" en la nota
        # representada por NU[i]
        if ('.' in NU[i]):
            t_s1 = 0
            t_s2 = 0
            ind_punto = i

            if(ind_punto > 1):
                for j in range (1, (ind_punto-1)+1):
                    t_s1 = t_s1 + TimeSemibreve(NU[j])
                    #print (TimeSemibreve(NU[j]))
            t_s1 = t_s1 + 2 * TimeSemibreve(NU[ind_punto]) / 3
            #print(TimeSemibreve(NU[ind_punto]))

            if(ind_punto < k-1):
                for j in range( ind_punto + 1, (k-1)+1):
                    t_s2 = t_s2 + TimeSemibreve(NU[j])
                    #print(TimeSemibreve(NU[j]))

            # En este caso ese "puntillo" es un
            # PUCTUS DIVISIONIS, porque DIVIDE LA SUCESION
            # en DOS PARTES con un NUMERO ENTERO de SEMIBREVES
            if( (int(t_s1) == t_s1) and (int(t_s2) == t_s2) ):
                NU[ind_punto] = Remove(NU[ind_punto], '.')
                break

    # Si al final del ciclo, ninguno de los puntillos es
```

```
# Punctus Divisionis, HAY ERROR
if not( ( int(t_s1) == t_s1 ) and ( int(t_s2) == t_s2 ) ):
      print("ERROR. No hay punctus divisionis y el numero
      de tiempos de semibreves entre breves poseen un valor
      fraccionario.")
      break

r1 = t_s1 % 3
r2 = t_s2 % 3

print("Presencia de 'punctus divisionis'.")
print("Para la parte anterior al 'punctus divisionis':")
## 2.2.1 ##
if (r1 == 1):
     print("Imperfeccion a.p.p.")
     NEW_NU.append(ChangeNoteValue(NU[0],'\\breve','2'))
     #Imperfeccion a.p.p.
     for i in range(1, (ind_punto)+1):
          NEW_NU.append(ChangeNoteValue(NU[i],
          GetNoteValue(NU[i]), GetActualValue(NU[i])))

## 2.2.2 ##
elif (r1 == 2):
     NEW_NU.append(ChangeNoteValue(NU[0], '\\breve', '2.'))
     # La breve inicial se mantiene perfecta
     for i in range(1, (ind_punto-1)+1):
          NEW_NU.append(ChangeNoteValue(NU[i],
          GetNoteValue(NU[i]), GetActualValue(NU[i])))
     ###### ERROR, cuando NO PUEDE OCURRIR ALTERACION ######
     ### (pero es imperativa por las condiciones dadas) ####
     if((not(GetNoteValue(NU[ind_punto]) == '1'))
     or ('r' in GetNote(NU[ind_punto]))):
          print("ERROR, no se puede alterar la ultima nota
          antes del 'punctus divisionis'.")
          break
     ############################################################
```

```
        # Alteracion (de ultima nota antes del punto)
        print("Alteracion.")
        NEW_NU.append(ChangeNoteValue(NU[ind_punto],
        GetNoteValue(NU[ind_punto]), '2'))

    ## 2.2.3 ##
    else:
        print("Se mantiene perfecta.")
        NEW_NU.append(ChangeNoteValue(NU[0],'\\breve','2.'))
        # La breve inicial se mantiene perfecta
        for i in range(1, (ind_punto)+1):
            NEW_NU.append(ChangeNoteValue(NU[i],
            GetNoteValue(NU[i]), GetActualValue(NU[i])))

    print("Para la parte posterior al 'punctus divisionis':")
    ## 2.2.4 ##
    if (r2 == 1):
        for i in range(ind_punto+1, (k-1)+1):
            NEW_NU.append(ChangeNoteValue(NU[i],
            GetNoteValue(NU[i]), GetActualValue(NU[i])))
        # ERROR, cuando NO PUEDE OCURRIR  IMPERFECCION A.P.A. #
        #  (pero es imperativa por las condiciones dadas)  #
        if( 'r' in GetNote(NU[k]) ):
            print ("ERROR, no se puede imperfeccionar(porque
            la ultima breve de la sucesion es silencio, y debe
            ser perfecto)")
            break
        if(len(NOTE) > end + 1):
        #Cuando una breve sigue a la ultima breve de la sucesion.
            if ( '\\breve' in NOTE[end+1] ):
                print("ERROR, no se puede imperfeccionar la
                ultima breve de la sucesion por estar seguida
                de otra breve")
                break
        ##################################################################
        # Imperfeccion a.p.a.
```

```
                print("Imperfeccion a.p.a.")
                NEW_NU.append(ChangeNoteValue(NU[k],GetNoteValue(NU[k]),
                '2')) #Imperfeccion a.p.a.
                start = end + 1

            ## 2.2.5 ##
            elif (r2 == 2):
                for i in range(ind_punto+1, (k-2)+1):
                    NEW_NU.append(ChangeNoteValue(NU[i],
                    GetNoteValue(NU[i]), GetActualValue(NU[i])))
                ##   ERROR, cuando NO PUEDE OCURRIR ALTERACION    ##
                ## (pero es imperativa por las condiciones dadas) ##
                if((not(GetNoteValue(NU[k-1]) == '1'))
                or ('r' in GetNote(NU[k-1]))):
                    print("ERROR, no se puede alterar la penultima
                    nota de esta sucesion.")
                    break
                ###############################################################
                # Alteracion
                print("Alteracion.")
                NEW_NU.append(ChangeNoteValue(NU[k-1],
                GetNoteValue(NU[k-1]),'2'))
                start = end

            ## 2.2.6 ##
            else:
                for i in range(ind_punto+1, (k-1)+1):
                    NEW_NU.append(ChangeNoteValue(NU[i],
                    GetNoteValue(NU[i]),GetActualValue(NU[i])))
                start = end


        # 3 -> Si NU[0] es una nota de menor duracion a una breve (sonido o
        # silencio, es indiferente) #
        if not(('\\breve' in GetNoteValue(NU[0]))
        or ('\\longa' in GetNoteValue(NU[0]))
```

```
or ('\\maxima' in GetNoteValue(NU[0]))):
    k = end - start
    t = 0
    for j in range(0, k):
        t = t + TimeSemibreve(NU[j])
        # Tiempos de semibreve de todas las notas de la sucesion
        # cuyo valor es menor a la breve
    r = t % 3

    # 3.1 -> Imperfeccion a.p.a. #
    if(r == 1):
        for i in range(0, (k-1)+1):
            NEW_NU.append(ChangeNoteValue(NU[i],
            GetNoteValue(NU[i]), GetActualValue(NU[i])))
        ### ERROR, cuando NO PUEDE OCURRIR  IMPERFECCION A.P.A. ###
        ###   (pero es imperativa por las condiciones dadas)   ###
        if( 'r' in GetNote(NU[k]) ):
            print ("ERROR, no se puede imperfeccionar(porque la
            ultima breve de la sucesion es silencio, y debe ser
            perfecto)")
        if(len(NOTE) > end + 1):
        #Cuando una breve sigue a la ultima breve de la sucesion.
            if ( '\\breve' in NOTE[end+1] ):
                print("ERROR, no se puede imperfeccionar la ultima
                breve de la sucesion por estar seguida de otra breve")
                break
        ##########################################################################
        print("Imperfeccion a.p.a.")
        NEW_NU.append(ChangeNoteValue(NU[k],GetNoteValue(NU[k]),'2'))
        #Imperfeccion a.p.a.
        start = end + 1

    # 3.2 -> Alteracion #
    elif(r == 2):
        for i in range(0, (k-2)+1):
            NEW_NU.append(ChangeNoteValue(NU[i], GetNoteValue(NU[i]),
```

```
                GetActualValue(NU[i])))
            ###     ERROR, cuando NO PUEDE OCURRIR ALTERACION     ###
            ###  (pero es imperativa por las condiciones dadas)   ###
            if((not(GetNoteValue(NU[k-1]) == '1'))
            or ('r' in GetNote(NU[k-1]))):
                print("ERROR, no se puede alterar (por el valor de la
                penultima nota de la sucesion)")
                break
            ##################################################################
            print("Alteracion.")
            NEW_NU.append(ChangeNoteValue(NU[k-1],
            GetNoteValue(NU[k-1]), '2')) # Alteracion
            start = end

        # 3.3 #
        else:
            for i in range(0, (k-1)+1):
                NEW_NU.append(ChangeNoteValue(NU[i],
                GetNoteValue(NU[i]),GetActualValue(NU[i])))
            start = end

    #print("start: ", start)
    for j in range (0, len(INDEXBREVES)):
        if INDEXBREVES[j] > start:
            end = INDEXBREVES[j]
            break
    #print("end: ", end)

    print("Transcripcion de la sucesion nu: ", NEW_NU)
    NEW_NOTE = NEW_NOTE + NEW_NU

if (start == end):  # Es una breve la nota final
    NEW_NOTE.append(ChangeNoteValue(NOTE[end],
    GetNoteValue(NOTE[end]),'2.'))
    start = start + 1
if(start > end):    # Es una longa la nota final
```

```
    # (anteriormente se tomo como una breve y se le
    # hizo la modificacion respectiva, agrego finalmente
    # la misma nota pero con valor de blanca con puntillo,
    # ambas se ligan (asi se genera la longa)
        final_note = NEW_NOTE[ len(NEW_NOTE) - 1]
        NEW_NOTE[len(NEW_NOTE)-1] = AddLigature(final_note ,'\\(')
        NEW_NOTE.append(AddLigature(ChangeNoteValue(final_note ,
        GetNoteValue(final_note), '2.'),'\\)'))
    print("Al momento se han transcrito correctamente las notas: ", NEW_NOTE)

    for new_note in NEW_NOTE:
        new_file.write(new_note + ' ')


##############################################################################
## FUNCIONES UTILES PARA LA SEPARACION DEL CODIGO LILYPOND DE ENTRADA ##
##        EN SECCIONES Y PARA LA LLAMADA DE LOS PROCEDIMIENTOS DE      ##
##                    CAMBIO DEL TEXTO Y DE LAS NOTAS                  ##
##############################################################################

# Metodo util para cambiar cualquier frase por otra en una palabra dada
def Change(word, old_phrase, new_phrase):
    ind = word.index(old_phrase)
    list_word = list(word)
    list_word[ind : ind+len(old_phrase)] = new_phrase
    new_word = ''.join(list_word)
    return new_word

# Modificacion de los comandos (instrucciones que indican el tipo de voz,
# clave y tiempo)
def GeneralModifications(TEXT, new_file):
    mensuration = 4/4
    MPHRASE = ['MensuralVoice', '4/4', '3/2', '6/4', '"mensural-f"',
    '"petrucci-f"', '"petrucci-c5"', '"mensural-g"', '"petrucci-g"',
    '"petrucci-c1"', '"petrucci-c2"', '"petrucci-c3"', '"petrucci-c4"']
    APHRASE = ['Voice', '2/4', '3/4', '6/8', 'F', 'F', 'F', 'G', 'G',
```

```
    'G', 'G', '"G_8"', '"G_8"']
    for line in TEXT:
        new_line = ''
        for word in line.split():
            new_word = word
            for i in range(0, len(MPHRASE)):
                if MPHRASE[i] in word:
                    new_word = Change(word, MPHRASE[i], APHRASE[i])
                    if (MPHRASE[i] == '4/4' or MPHRASE[i] == '3/2'
                    or MPHRASE[i] == '6/4'):
                        mensuration = MPHRASE[i]
            new_line = new_line + new_word + ' '
        new_file.write(new_line + '\n')
    return mensuration

# Se determina el proceso de transcripcion segun la mensuracion de la voz
def Transcripte(ARRAY_NOTES, mensuration_type, new_file):
    if mensuration_type == '4/4':
        print("Mensuration: TEMPUS IMPERFECTUM CUM PROLATIONE IMPERFECTA")
        print("The notes of this voice are: ", ARRAY_NOTES)
        TranscripteII(ARRAY_NOTES, new_file)
    elif mensuration_type == '3/2':
        print("Mensuration: TEMPUS PERFECTUM CUM PROLATIONE IMPERFECTA")
        print("The notes of this voice are: ", ARRAY_NOTES)
        TranscriptePI(ARRAY_NOTES, new_file)
    #elif mensuration_type == '6/4':
        print("Mensuration: TEMPUS IMPERFECTUM CUM PROLATIONE PERFECTA")
        print("The notes of this voice are: ", ARRAY_NOTES)
     #   TranscripteIP(ARRAY_NOTES, new_file)


def TEXTS_NOTES(source, new_file):
    ##########################################################################
    # SE SECCIONA EL CODIGO LILYPOND EN DOS ARREGLOS (de TEXTO y de NOTAS) #
    ##########################################################################
    count = 0
```

```
COUNT_TEXT_LINE = []
COUNT_NOTES_LINE = []

TEXT = []
TEXTS = []
NOTES_VOICE = []
NOTES = []

for line in source:
    count = count + 1
    words_line = line.split()

    # Si la linea esta vacia
    if (words_line == []):
        TEXT.append(line)
        COUNT_TEXT_LINE.append(count)
    # Si no esta vacia
    else:
        first_word = words_line[0]
        first_character = list(first_word)[0]
        # Si la linea es una linea de instruccion (no de notas)
        if (((first_character == '\\') and ('\\[' not in first_word))
        or ('}' in first_word)):
            if len(COUNT_TEXT_LINE) == 0:
                TEXT.append(line)
            else:
                l = len(COUNT_TEXT_LINE)
                d = count - COUNT_TEXT_LINE[l-1]
                if d > 1:
                    TEXTS.append(TEXT)
                    TEXT = []
                TEXT.append(line)
            COUNT_TEXT_LINE.append(count)
        # Si la linea es una linea de notas
        else:
            if len(COUNT_NOTES_LINE) == 0:
```

```
                    for note in words_line:
                        NOTES_VOICE.append(note)
                else:
                    l = len(COUNT_NOTES_LINE)
                    d = count - COUNT_NOTES_LINE[l-1]
                    if d > 1:
                        NOTES.append(NOTES_VOICE)
                        NOTES_VOICE = []
                    for note in words_line:
                        NOTES_VOICE.append(note)
                COUNT_NOTES_LINE.append(count)
    # Anadiendo los ultimos arrays de texto y notas
    TEXTS.append(TEXT)
    NOTES.append(NOTES_VOICE)

    ##########################################################################
    ##  SE LLAMA A LOS METODOS GeneralModifications y Transcripte   ##
    ##########################################################################

    L = len(NOTES)
    for i in range(0, L):
        mensuration = GeneralModifications(TEXTS[i], new_file)
        Transcripte(NOTES[i], mensuration, new_file)
    GeneralModifications(TEXTS[L], new_file)

print ("Please insert the name of the LilyPond file you want to convert
from mensural to actual notation. Example: file_name.ly. Is important
that your file is in this carpet (the same as the program for converting)")
src = input("Insert input file here: ")
input_file = open(src, "r")
print ("Insert the name of the file you want to contain the transcription.
Example: file_name.ly. The file doesn't have to exist, it will be created in
this carptet after you write the name.")
mod = input("Insert it here: ")
output_file = open(mod, "w")
TEXTS_NOTES(input_file, output_file)
```

```
input_file.close()
output_file.close()
```