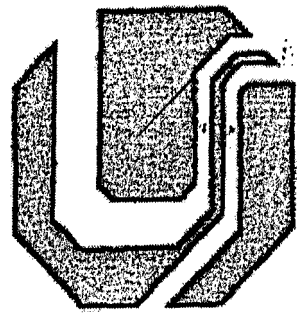
UNIVERSIDADE FEDERAL DE UBERLÂNDIA
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
PÓS-GRADUAÇÃ0 EM ENGENHARJA ELÉTRICA



CRIAÇÃO DE UMA INTERFACE COMUM PAPA MANIPULAÇÃO DE MIDI USANDO A LINGUAGEM FUNCIONAL CLEAN

WELLESLEY BARROS FERREIRA

SETEMBRO
2000

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

CRIAÇÃO DE UMA INTERFACE COMUM PARA MANIPULAÇÃO DE MIDI USANDO A LINGUAGEM FUNCIONAL CLEAN

Dissertação apresentada por Wellesley Barros Ferreira a Universidade Federal de Uberlândia para obtenção do titulo de Mestre em Engenharia Elétrica aprovada em 11/09/2000 pela Banca Examinadora:

Professor Antônio Eduardo Costa Pereira, Ph.D.(orientador)
Professor Cedric Luiz de Carvalho, Dr. (UFO)
Professor Jamil Salem Barbar, Dr. (UFU)

CRIAÇÃO DE UMA INTERFACE COMUM PARA MAMPULAÇÃO DE MIDI USANDO A LINGUAGEM FUNCIONAL CLEAN

WELLESLEY BARROS FERRERIA

Dissertação apresentada por Wellesley Barros Ferreira a Universidade Federal de Uberlândia como parte dos requisitos para obtenção do titulo de Mestre em Engenharia Elétrica.

Professor Antônio Eduardo Costa Pereira Orientador	Luiz Carlos de Freitas Coordenador do Curso de Pós-Graduação

Ao Professores Antônio Eduardo Costa Pereira, orientador acadêmico, pelo apoio e confiança depositada. Ao Professor Luciano Vieira Lima pela inestimável colaboração. Tive o privilegio e a felicidade de tê-los conhecido e tê-los como amigos.

Ao CNPq – Conselho Nacional de Desenvolvimento Cientifico e Tecnológico, pela ajuda financeira recebida no decurso do meu trabalho de Mestrado.

Ferreira, W.B. Criação de uma interface comum para manipulação de MIDI Usando a Linguagem funcional CLEAN Uberlândia, UFU, 2000, 191p

RESUMO

Apresenta-se um estudo, em forma de software, onde um editor musical simples (editor de partitura) e ferramentas adequadas ao universo musical, dá entrada a uma música qualquer e a converte para o formato MIDI mais apropriado ao universo da computação. Neste processo de conversão entre um formato e outro usa-se um padrão intermediário que tem por objetivo melhorar o entendimento daqueles que têm algum conhecimento tanto da sintaxe musical quanto a do MIDI.

PALAVRAS CHAVES

MUSICA COMPUTACIONAL MIDI, EDITOR DE PARTITURA

Ferreira, W. B. Creation of a common interface for MIDI manipulation using the functional language CLEAN, Uberlândia, UFU, 2000. 191p

ABSTRACT

One shows a study in software, where a simple musical editor (score editor) using suitable tools adopted to musical format, get a music arid convert it in MIDI format which is more appropriate to computational processing. In the Conversion process between both formats one use a pattern who improve understanding for those who know some about music and MIDI format.

KEYWORDS

COMPUTER MUSIC, MIDI SCORE EDITOR.

CRIAÇÃO DE UMA INTERFACE COMUM PARA MANIPULAÇÃO DE MIDI USANDO A LINGUAGEM FUNCIONAL CLEAN

SUMÁRIO

CRIAÇÃO DE UMA INTERFACE COMUM PARA MANIPULAÇÃO DE MIDI USANDO A LINGUAGEM FUNCIONAL CLEAN

CAPÍTULO 1

INTRODUÇÃO

A proposta deste trabalho de dissertação é criar ferramentas de software pelas quais possam, músicos e não músicos, entendidos em software, ou simplesmente alguém que necessita manipular o padrão MIDI desenvolverem aplicações usando este padrão. As idéias que serão expostas aqui, não têm a pretensão de resolver, de forma absoluta, os vários problemas que serão tratados, explica-se: A entrada das notas na partitura, as idéias básicas que envolvem a questão foram implementadas, mas não foram as ferramentas que façam inserir todos os blocos rítmicos de notas que constituem a teoria musical. Pode-se, é claro, expandir aquelas ferramentas para que façam este trabalho.

A divisão lógica do trabalho é a seguinte: 1) introdução de conceitos, musicais e MIDI básicos e proposta de um padrão intermediário a meio caminho da notação musical tradicional e o MIDI. 2) Desenvolvimento das teorias apresentadas no item 1 produzindo um editor musical simples que transforma o formato visual das notas num formato texto. A seguir transforma-se este texto num arquivo MIDI.

Supõe-se no leitor conhecimentos musicais suficientes para entender os jargões, básicos da música. Mostrar-se-á blocos rítmicos e testar-se-ão entradas de notas no editor de partitura e ao fazê-lo irá se supor que o leitor consiga decifrar o texto musical. A experiência demonstrou que a ênfase deste trabalho deve ser no esclarecimento do formato MIDI, por ser mais obscuro e por transformar aquela forma musical gráfica, mais apropriada ao entendimento humano, naquela mais apropriada aos paradigmas computacionais Para levar a cabo a tarefa perde-se no caminho a clareza do método gráfico em detrimento ao matemático que leva em consideração o fenômeno temporal relativo da música Como prova disso pode-se citar que um pulso dita o ritmo de uma música percebida por um ser humano, o que é um fenômeno bastante natural para nosso cérebro, levando-se em consideração que se está acostumado, pelo próprio corpo, aos ritmos orgânicos. Muitas coisas neste processo, no entanto, passam despercebidas pela

nossa consciência objetiva. Ajustes são naturalmente feitos quando dos erros na interpretação da marcação do pulso musical, ao se ler determinada obra, o que não lhe desfigura o sentido, contudo, as marcações dos pulsos musicais feitos por um computador não possuem, é claro, ajuste natural e podem alterar significativamente uma composição caso não se leve em consideração a relatividade rítmica das notas entre si e em relação a um *clock*, que como se disse, não é um "*clock*" humano, mais sofisticado e adaptativo. Estas e outras questões a serem resolvidas estarão insertas nos exemplos que virão e muitas delas serão resolvidas simplesmente com a apreensão do formato MIDI.

Para iniciar a tarefa nada mais óbvia (mas nada mais importante) do que partir do principio das coisas. O que vem a ser o padrão MIDI?

MIDI significa "*Musical Instrument Digital Interface*" e como o nome diz serve para se criar ou reproduzir música por computador, usando este padrão para a tarefa.

A criação dos sintetizadores de som abriu uma gama fantástica de possibilidades aos apaixonados pelo processo de geração e execução de música. Isto permitiu que se pudesse reproduzir em um único equipamento um grande número de timbres, modificá-los conforme a necessidade, e, também, gerar sons até então inexistentes na natureza ou criado pelas mãos do homem.

Neste momento (apenas uma década atrás) cada sintetizador podia executar um instrumento de cada vez. Mesmo hoje em dia, o número de instrumentos a serem executados por um mesmo equipamento ainda é bem limitado.

Este fator exigiu que um mesmo grupo possuísse mais de um tecladista, conforme a necessidade do arranjo efetuado, desta necessidade surgiu a capacidade de poder interligar vários sintetizadores de forma que pudessem trabalhar em conjunto, sincronizados, e, se desejado, ativados por um músico apenas. Existiam, como hoje, um grande número de fabricantes que se propuseram a construir sintetizadores, cada qual com características e recursos diferentes. No começo deste processo de integração os vários equipamentos se limitavam a portabilidade apenas de um mesmo fabricante, exigindo interfaces de hardware quando se desejava interligar aparelhos de fabricantes diferentes. Assim, surgiu a necessidade de se criar um padrão que tomasse compatíveis equipamentos de diversos modelos e marcas.

É preciso fazer uma observação importante, relativa à comunicação entre instrumentos musicais, no presente caso, entre sintetizadores analógicos. Por volta dos anos 60 e 70, os instrumentos musicais possuíam apenas a tecnologia eletrônica analógica. Naquela época, alguns fabricantes norte americanos como *Moog, Polyfiision, Oberheim* e posteriormente a *Sequential Circuits*, adotavam uma espécie de padrão que possibilitava tocar um instrumento a partir do teclado de outro instrumento. Ao se

[1] O timbre é o som característico de um instrumento é o que nos faz distingui-los

pressionar uma tecla de um dado instrumento. (monofônico), uma tensão era gerada transmitida por um cabo a outro instrumento, fazendo com que este outro instrumento emitisse uma nota de mesma freqüência que a do primeiro. O padrão adotado foi o de 1 volt por oitava musical. *Moog* teve muito sucesso com seus sistemas modulares, onde vários osciladores podiam ser controlados ao mesmo tempo por uma mesma tecla. Evidentemente este processo só servia para instrumentos monofônicos e monotimbrais, pois cada nota possuía uma tensão correspondente, e se se quisesse transmitir ao instrumento várias notas, seriam necessários diversos cabos, um para cada nota/tensão.

Com o surgimento de sintetizadores polifônicos, a transmissão analógica inviabilizava a transmissão de informações polifônicas. Com o tempo os sintetizadores incorporaram microprocessadores em seus circuitos e, dai, a comunicação de dados digitalmente era previsível e lógica.

O SURGIMENTO DO MIDI

Em 1981, três homens trabalhavam em companhia de sintetizadores, *Dave Smith da Sequential Circults I. Kakehashi da Roland Corporation e Tom Oberheim da Oberheim Erectronics,* eles se encontraram na mostra de junho da *National Association Merchants (NAMM).* Eles discutiram a possibilidade da criação de um padrão para controle de sintetizadores de diferentes companhias. Como fruto desta conversa foi idealizada a *USI (Universal Synthesizer Inteface)* Em novembro do mesmo ano eles tomaram esta proposta pública na *Audio Engineers Society.* Em janeiro de 1982 eles promoveram um encontro de fabricantes de sintetizadores na mostra do *NAMM,* voltados à apresentação da *USL.* Neste encontro estavam presentes representantes da *Circuits, Roland, Oberhein, Yamaha, Korg e Kawai.* Estes fabricantes já possuíam seus próprios padrões de comunicação entre suas máquinas. Quando estas companhias se encontraram novamente, na mostra de junho de 1982 do *NAMM,* como esforço conjunto, resultou a idealização do que é hoje o padrão MIDI.

A primeira noticia que se teve sobre a criação do padrão MIDI foi através de um artigo escrito por *Robert Moog* para a revista norte-americana *KEYBOARD* de outubro de 1982 a dezembro de 1982, a *Sequential Circuits* colocou no mercado o primeiro instrumento musical dotado de interface MIDI, que foi o sintetizador *Prophet600.* Isto aconteceu antes mesmo da publicação da MIDI Specification 1.0 na mostra do *NAMM* de janeiro de 1983. Vários equipamentos apresentados já incorporavam esta interface

MIDI sendo que, como exemplo, foram interligados um sintetizador da *Roland (JP6)* com um sintetizador da *Sequencial Circuits (Prophet 600)*. A apresentação foi um sucesso para delírio e satisfação de seus idealizadores dos participantes e da comunidade musical,

Em Agosto de 19830 uma versão completa da especificação MIDI foi levada a público com a denominação MIDI 1.0.

A intenção inicial do MIDI era transmitir informações de teclado, ou seja, era um sistema orientado para instrumentos de teclado. Realmente a proposta do padrão *USI de Dave Smith* definia apenas comandos de tecla pressionada e tecla solta. Felizmente, a idéia inicial foi aprimorada e passou a permitir a adição de funções futuras, o que realmente ocorreu, e continua ocorrendo ainda hoje.

Pode-se dizer que a *MIDI Specifícation 1.0,* em sua essência, é válida até hoje Naquela época foram definidos alguns poucos comandos, mas, por causa da previsão inclusões têm sido feitas dentro das primeiras regras estabelecidas pelos criadores. Até o presente momento, tem havido espaço para inclusão de novos comandos ou funções. Posteriormente a *IMA(International MIDI Association*), foram criadas outras entidades como a *MMA (MIDI Manufactures Association)* EUA *e a JMSC (Japan MIDI Standard Comittee)* Japão. Essas associações, que reúnem os principais fabricantes mundiais, deliberam conjuntamente sobre a matéria, tendo poderes para alteração, quando necessário da especificação origina.

Desde 1984 o padrão MIDI está bastante estável e os fabricantes que demoraram a incorporá-lo em seus equipamentos sentiram uma grande dificuldade de colocar seus produtos no mercado.

Voltada ao objetivo de se utilizar computadores pessoais como equipamento MIDI a *Roland Corporation,* em 1984, lançou no mercado urna placa de interface MIDI denominada *MPU4O1*. Só por volta de 1987 que a comunidade dos utilitários de *PCs* descobriram este padrão e suas possibilidades no processo da automação musical. Com a utilização de computadores neste processo de interligação de máquinas com MIDI houve um crescimento muito grande na qualidade e nas opções que os músicos e gravadoras passaram a usufruir.

CAPITULO II

UMA REVISÃO MUSICAL E MIDI BÁSICO

2.1) VISÃO PANORÂMICA:

MIDI é um conjunto de especificações padrão utilizado por fabricantes de instrumentos eletrônicos musicais o qual permite que instrumentos de fabricantes diferentes possam ser interligados com total compatibilidade.

A palavra INTERFACE DE MIDI é um termo técnico utilizado para um hardware que permite que dois sistemas diferentes possam ser interligados por um soquete, denominado MIDI *PORT*, através de um cabo MIDI. Este hardware se responsabiliza por transmitir e receber as informações utilizadas na comunicação entre os instrumentos em questão. O formato de dados recebidos e enviados através desta interface são digitais (como o D de MIDI já indica). Isto justifica a total compatibilidade na troca de informações, sendo que cada instrumento envolvido na comunicação não necessita conhecer as características internas do(s) outro(s) dispositivo(s) envolvido(s). Cada mensagem MIDI é a comunicação de um evento musical. Evento musical são ações efetuadas por um instrumentista, tais como: Freqüência da nota, duração, ajuste de pedais, etc...

Um instrumento que é capaz de enviar e receber corretamente informações MIDI é denominado dispositivo MIDI. O mais comum dispositivo encontrado na atualidade é o sintetizador. Ele possui um microprocessador interno que o capacita a gerar sons e ler as informações enviadas por um teclado. Assim se torna simples para o fabricante acrescentar *ports* MIDI e programar este microprocessador interno para receber, enviar e interpretar o padrão MIDI. Máquinas de ritmo também possuem microprocessadores internos e, portanto, são também fáceis de serem adaptadas ao padrão MIDI. Já os instrumentos convencionais tais como guitarras, flautas, etc necessitam de uma nova interface contendo um sistema microprocessado de forma que possa interpretar, através de sensores especiais, os sinais enviados. Um computador pode ser um dispositivo MIDI, desde que possua um adaptador, usualmente denominado de interface MIDI, adote *ports* MIDI e possua um conjunto de programas que habilite o processador deste computador receber, enviar e interpretar os sinais MIDI existentes na comunicação pretendida. Amplificadores modernos microprocessados,

mesas de som e de gravação também podem ser encarados como dispositivos MIDI desde que estas preencham as características citadas.

O evento musical mais simples que pede ser transmitido ou recebido em MIDI é a presença de uma nota, ou seja, quando um instrumentista executa uma nota em seu instrumento, o hardware MIDI envia unia mensagem identificando-a. Quando ele liberara a nota outra mensagem é enviada através dos *ports* e cabos MIDI

Uma mensagem MIDI significa a freqüência da nota a ser executada, sua duração, volume, reverberação, informações de chaves sendo ativadas e desativadas.

Informações MIDI podem, também, transmitir informações quanto à execução de músicas pré-gravadas nos dispositivos MIDI e podem sincronizar a execução de uma determinada sinfonia.

Teoricamente o MIDI pode conectar um número infinito de dispositivos. Na prática este número é limitado. Primeiramente pelo fator econômico. Cada instrumento a ser conectado, necessita de um *port* MIDI. Outro fator diz respeito à quantidade simultânea de informações MIDI. Isto pode comprometer a capacidade de recebimento destas informações pelo teclado mestre ou pelo computador, e dificultar as decisões que passam ser efetuadas numa determinada velocidade. Um sistema profissional, revestido de uma qualidade mínima, deve possuir a capacidade de gerenciar no mínimo seis equipamentos MIDI simultaneamente.

A utilização mais comum em MIDI é a sincronização das notas produzidas pelos sintetizadores. Este processo é denominado seqüenciamento. Hoje o MIDI capacita o usuário a seqüenciar mais de 30 sintetizadores. Através de um teclado gerenciador ou de um computador pode-se selecionar, em cada instrumento, o som desejado a cada instante da execução da música em questão. Através deste teclado, selecionar o instrumento adequado a cada trecho musical, sem que seja necessário estar trocando de lugar e teclados o tempo todo. Além do processo de sequenciamento de músicas, o MIDI facilita os processos de gravação e edição de músicas, como exemplo, pode-se citar a recepção de informações de um teclado e a colocação desta informação musical numa partitura. O MIDI envia para o computador o código das notas tocadas durante uma execução e elas são armazenadas conjuntamente com suas características tais como volume, duração e som (freqüência). Após a inserção da música pode-se alterar o tempo em que um determinado ritmo foi executado e transpô-la para outra tonalidade. Estas tarefas são bastante árduas pelo sistema manual, mas não o é por um editor de partituras ou pelo MIDI.

O padrão MIDI permite, quando utilizado por computadores, que eles possam ser transformados em gravadores multicanais, bastando para isto que cada instrumento seja gravado independentemente e posteriormente tocado simultaneamente. Através de software especifico o MIDI permite que partituras musicais sejam instantaneamente obtidas à medida que se vai tocando. Nestes programas, instrumentos diferentes podem ser separados em partituras diferentes. Esta característica soluciona uma das tarefas mais rejeitadas pela maioria dos músicos. O processo reverso pode ser também executado, ou seja, através das partituras de diversos instrumentos o seqüenciador pode ativar simultaneamente os diversos dispositivos MIDI de forma a que eles executem simultaneamente a música em questão

Apesar de se ter excelentes programas MIDI para a linha dos *IBM* PCs, não foram estes os primeiros computadores a utilizar e possuir programas de aplicações musicais utilizando este padrão. Programas de seqüenciamento apareceram primeiro nos equipamentos da linha *MACINTOSH.*

Os primeiros computadores a trabalharem com software MIDI foram: *Atari 520ST, Macintosh Plus, Atari 800, Commodore 64, Amiga e Apple II* por volta de 1981.

Os sistemas do *Atari 520 ST e 1040ST* foram os mais utilizados pelos profissionais de música. Como o *Macintosh, o Atari* era construído com um processador da linha 68.000 da *Motorola.* Este computador possuía um processador de sons interno de 3 vozes. Seu sistema operacional trabalhava com ícones o que o tornava bastante atrativo aos músicos pouco habituados com computadores[1]

O *Macintosh Plus* também possuía um *chip* de som a 4 vozes com qualidade bem superior ao do *Atari,* a ponto de se poder comparar sua performance com alguns bons sintetizadores da época, no entanto, não possuía internamente nenhum *port* MIDI sendo assim era necessário adquirir uma placa de interface.

Quanto aos *IBM PCs,* estavam longe de serem considerados como uma opção para música por computador. Eles ingressaram nesta área bem depois de outros concorrentes da linha 68.000 da *Motorola.*

---

[1] Número de vozes é a quantidade de notas musicais que um instrumento consegue tocar ao mesmo tempo. a isso dá-se o nome de polifonia. Hoje em dia é comum encontrar instrumentos com polifonia de 64 ou mais vozes. É comum confundir polifonia com multitimbralidade. Multitimbralidade é a capacidade de um instrumento poder reproduzir mais de um timbre de instrumento ao mesmo tempo.

Atualmente existem bons recursos na área MIDI para computadores da linha *PC* da *IBM*, só que ainda estão bem atrasados em relação aos da linha da *Motorola*, por exemplo, os *Macintoshs*, no que tange a hardware profissional.

Como placa pioneira em sua função, a *MPU401 da Roland,* definiu o padrão para a linha *IBM PC*. A maioria dos circuitos desta placa foi substituído por placas mais simples controladas internamente por softwares dedicados. Quase sempre, ao se referir a uma interface MIDI para computadores da linha *IBM PC*, refere-se também a ela ser ou não compatível com a *MPU401*. A grande maioria das placas de som atual já possui incorporada uma interface MIDI necessitando para completar a ligação com outros dispositivos MIDI apenas um MIDI *CABLE*, que contém os conectores *DIN* e a interface ótica de acoplamento entre eles.

Utilizava-se a placa *TDSAP da Syntech Corporation no Apple II.* Para o *Commodore 64 a TDSC64 também da Syrtech Corporation e para a Atari Plus a MIDIMAC da Opcode System.*

No *MIDI Specifications 1.0* foram definidas as características básicas de interfaceamento para o padrão, em nível de *hardware* e *software*. A comunicação é unilateral, a não ser em casos específicos do Modo *SYSEX*. Conectar dispositivos MIDI é a parte mais simples da utilização do MIDI. A padronização dos *ports*, cabos, e conectores simplificam o processo.

2.2) ELEMENTOS GERAIS DA MÚSICA:

A diferença entre a freqüência de cada nota musical na escala temperada é dada pela relação:

$$\sqrt[12]{2}$$

A freqüência de referencia para afinação dos instrumentos musicais (diapasão) é de 440 Hertz, o que equivale à nota musical A4.

Para se entender melhor estas denominações A4, C5, G5, etc apresentar-se-á um teclado de piano com suas respectivas notas.

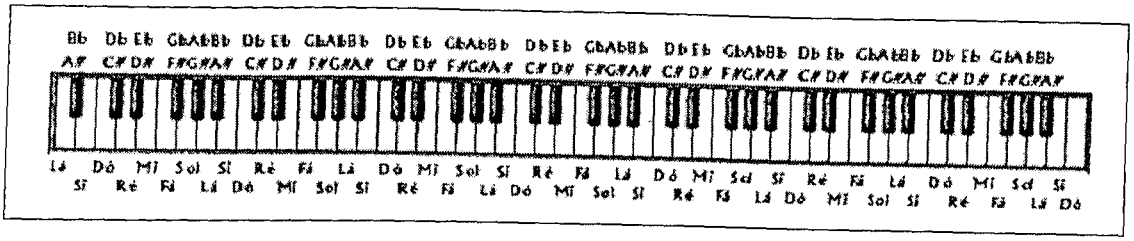


Figura: 2. 1 – Teclado de piano com a notação alemã

Neste teclado, utilizou-se, propositadamente, duas nomenclaturas diferentes para dar nome as notas. Usa-se neste trabalho a convenção alemã para notas (veja abaixo) e a imagem de um teclado por facilidade de exemplificação.

Lá = A. Si = B. Do = C. Ré = D. Mi = E. Fá = F e Sol = G

A nota mais grave (extrema esquerda do teclado) denomina-se Lá1. A nota mais aguda Dó9. Denominamos tecitura do instrumento a faixa de notas que ele consegue reproduzir, assim, a tecitura de um piano vai da nota Lá1 a nota Dó9, ou pela notação que se passa a adotar: A1 a C9.

Vale lembrar também que este mesmo sistema alemão é empregado para acordes.

As notas se repetem em grupos de 12 notas (dai o dodecafonismo de *Shöenberg*), mas de fato, se não se considera o meio (mas apenas as notas brancas) tem-se grupos de oito notas, assim entre A1 e A2 tem-se 8 notas a isto dá-se o nome de oitava. A cada oitava acima a freqüência dobra, abaixo, a freqüência cai pela metade. A figura ilustra o fato:

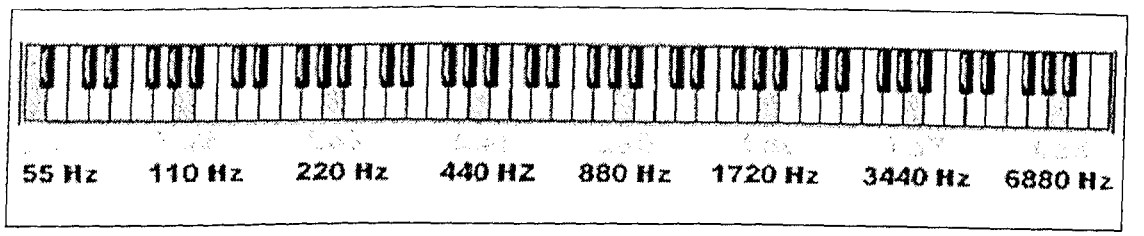


Figura 2.2 – Teclado com freqüências

Como já se disse a cada doze notas (incluindo os sustenidos e bemóis) uma nota se repete com o mesmo nome, mas com freqüência diferente. Olhando para o piano, vê-se teclas brancas e teclas pretas.

Dá-se o nome de acidente à elevação de meio tom de uma nota, por exemplo, de forma ascendente as notas pretas representam os meio-tons, A# está a ½ tom acima de A e

½ tom abaixo de B. Bb está meio tom abaixo de B e ½ tom acima de A. Nota-se que, para cada tecla preta temos dois nomes associados a ela. A regra é a seguinte: Uma tecla preta tem o mesmo nome que a tecla branca anterior ou posterior. Quando se utiliza o nome da tecla branca anterior, coloca-se o símbolo # (sustenido) à frente do nome desta tecla, quando se utiliza o nome da tecla posterior, coloca-se a símbolo **b** (bemol), Assim, Dó# = Réb, Fá# = Solb e assim por diante, como mostra a figura 2.3. A este fenômeno dá-se o nome de enarmônicos. Este é um conceito musical que designa uma nota real com vários nomes assim Bb=A#, D#=Eb, C=Dbb (duplo bemol). D=Cx(duplo sustenido).

Resumindo: sustenido significa meio tom acima e bemol significa meio tom abaixo da nota de mesmo nome.

A seguir uma figura mostra em destaque uma oitava musical, no piano, com as duas nomenclaturas com respectivos bemóis e sustenidos.

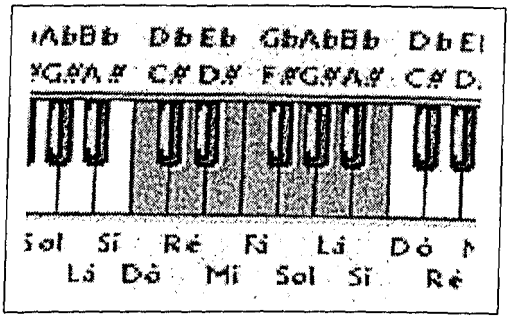


Figura 2.3

2.3) DISTRIBUIÇÃO RÍTMICA DAS NOTAS

Cada nota musical pode soar por um certo intervalo discreto no tempo. Em notação musical o tempo de cada nota é múltiplo uma das outras.

O gráfico da figura 2.4 não cobre todos os tipos de ritmos. Faltam aqui a nota breve que equivale a duas semibreves e as fusas, semifusas e quartifusas.

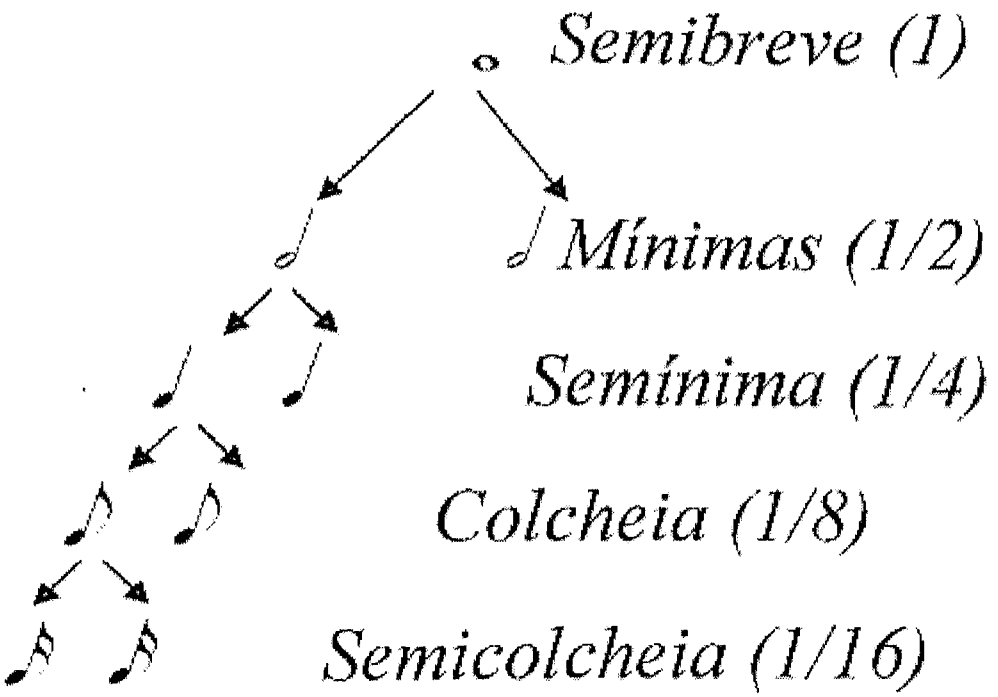


figura 2.4 – Distribuição Rítmica

Se à frente de uma nota colocar-se um ponto (.), significa que o valor desta nota é aumentado pela metade. A isto dá-se a nome de nota pontuada. Assim, uma semínima pontuada (□.) possui o valor de uma semínima mais uma colcheia (já que uma colcheia é metade de uma semínima). Se se coloca dois pontos à frente de uma nota, por exemplo, uma semínima com dois pontos (□..) significa que o tempo desta nota é dado por uma semínima, mais metade de uma semínima (uma colcheia) e mais a metade de uma colcheia (uma semicolcheia). Desta forma, o ponto sempre aumenta a metade do valor da última nota ou ponto que o precede.

Na figura a seguir 1 e 2 são equivalentes.

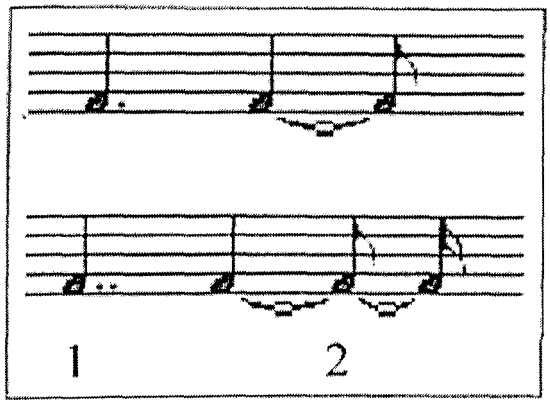


Figura 2.5 – Equivalência de Rítmicos

A figura a seguir mostra as notas musicais, com as respectivas pausas e alguns símbolos normalmente utilizados na partitura para definir duração e alterar a altura (freqüência da nota).

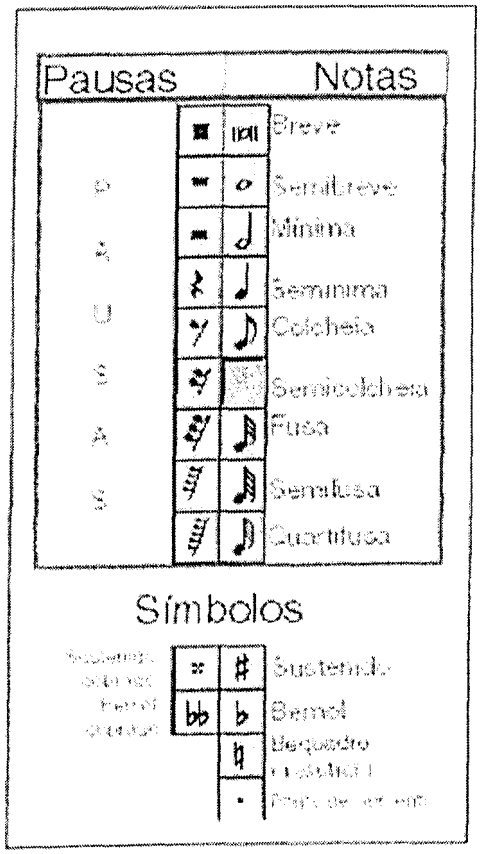


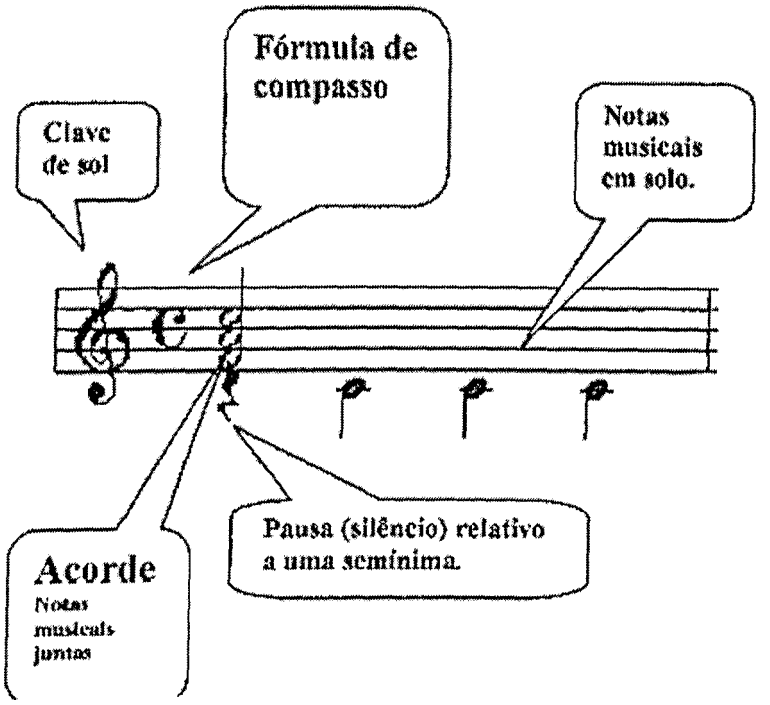
Figura 2.6

O símbolo bequadro equivale a retirar da nota, seu acidente. Se uma nota Fá# recebe um bequadro ela passa a ser Fá natural. Nota-se que as frações da figura 2,4 são

uma abstração matemática tão somente com valor de relação entre os objetos considerados. Poderia-se, por isto, ter qualquer valor para a semibreve e as outras notas acompanhariam esta relação matemática.

2.4) ELEMENTOS BASICOS DE UM PENTAGRAMA

Figura2.7 – Elementos básicos de um pentagrama



2.4.1) CLAVE (CLEF)

O símbolo & da figura 2.7 é um clave, a clave de sol. Ela é quem coordena as notas no pentagrama (as cinco linhas onde pousam as notas). O ponto onde o desenho da clave inicia dá nome à nota, no caso da figura anterior, o ponto inicial se dá na segunda linha de baixo para cima. Esta linha passa a representar a nota musical Sol (no caso G5) por causa da clave de sol.

A seguir alguns exemplos de outras claves.

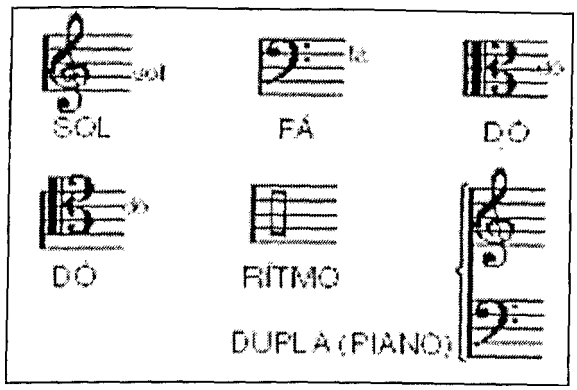


Figura 2.8

A seguir as posições das notas no pentagrama com as claves de Sol e de Fá.

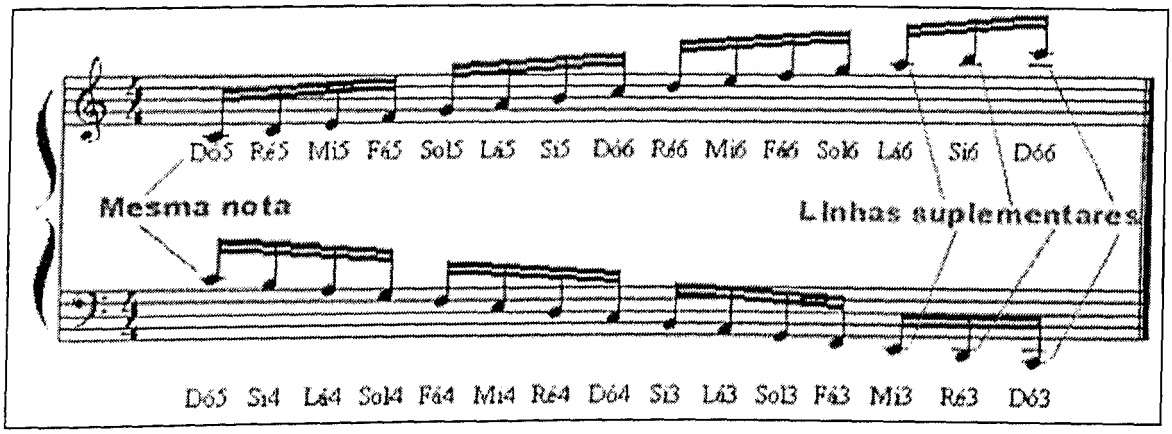


Figura 2.9 Equivalência das notas no pentagrama

Pode-se perceber que o G5 e o F4 estão grafados na linha onde começam os desenhos da Clave. As linhas suplementares servem para evitar-se colocar mais de 5 linhas em cada pentagrama, assim, nos lugares onde deveria ter mais urna linha tem-se um traço na nota, sobre ela ou abaixo dela. É o caso, na clave de F, do E3 que tem uma linha cortando-o. Do D3 que tem um traço acima e de B6. na clave de G, que tem um traço abaixo. A estes traços dá-se o nome genérico de linhas complementares. A figura 2.10 mostra a nota C5, que é comum às duas claves.

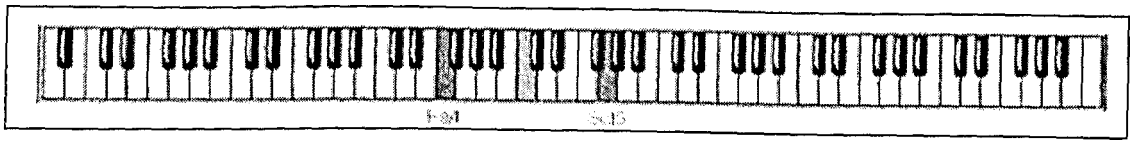
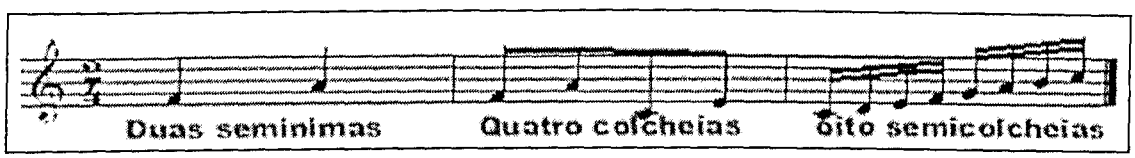


Figura 2.10

2.4.2) FÓRMULA DE COMPASSO (*TIME SIGNATURE)*

O símbolo "**c**" contido na figura 2.7, determina a divisão do tempo de cada compasso (cada bloco entre duas barras |) indicando a quantidade de notas que ele terá e a sua unidade de tempo. Por exemplo, a fórmula de compasso 2/4 significa 2 notas do tipo ¼ que pela figura 2.4 se sabe ser a semínima (unidade de tempo de toda música). Isto significa que cada compasso terá o tempo de duas semínimas ou uma h que é a unidade de compasso da música. Quando se diz do tempo de duas semínimas ou uma h não quer dizer que elas devam ser sempre empregadas. mas qualquer conjunto de figuras que as equivalham. No caso da figura 2.11, duas semínimas, quatro colcheias ou 8 semicolcheias ocupam o mesmo tempo em um compasso, ou seja, são tempos (mas não ritmos) equivalentes.

Figura2.11 – Equivalência Rítmica



Outras fórmulas de compasso podem ser:

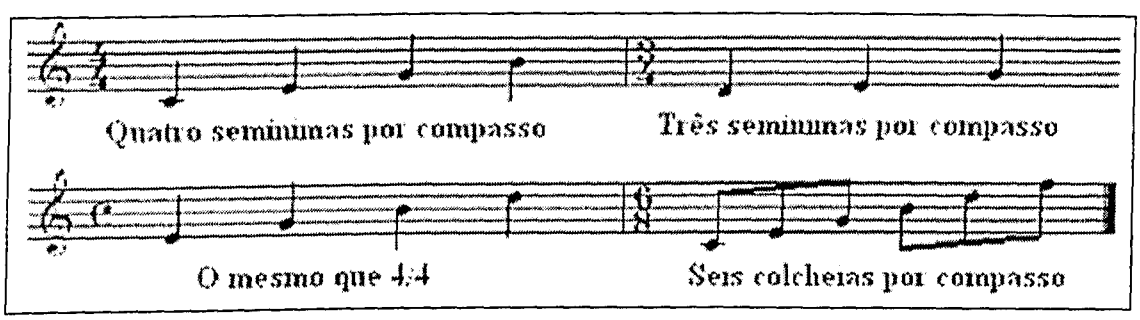


Figura 2.12 – Equivalência Rítmica

2.4.3) ARMADURA DE CLAVE*(KEY SIGNATURE)*

Na figura 2.7 não se tem nenhum acidente na armadura. Entende-se por armadura os acidentes que ocorrerão durante a música. Estes acidentes, até a era romântica obedecem a uma ordem rígida, veja figura 2.13. A Isto dá se o nome de tonalidade da música,

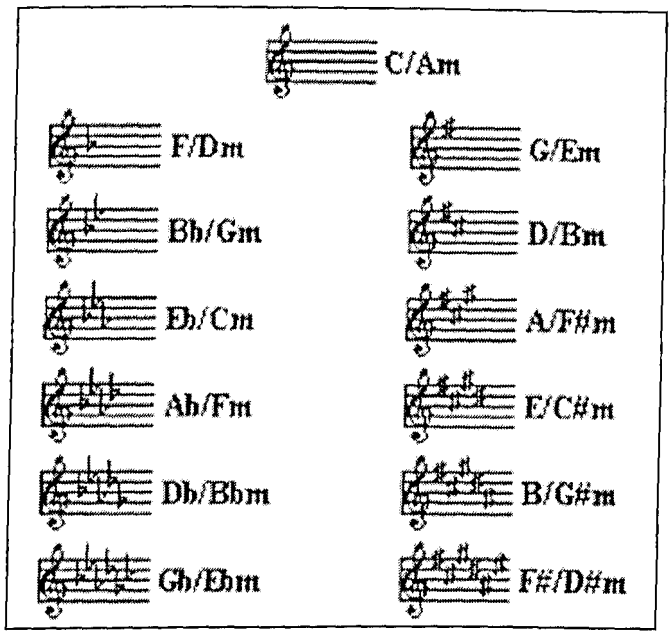


Figura 2.13 – Armadura de Claves indica a tonalidade da música

Estes acidentes na armadura indicam que todas as notas que coincidam com o acidente grafado no pentagrama serão ou bemóis ou sustenidos, seja qual for a altura (freqüência da nota). Isto evita a poluição de acidentes na notação musical.

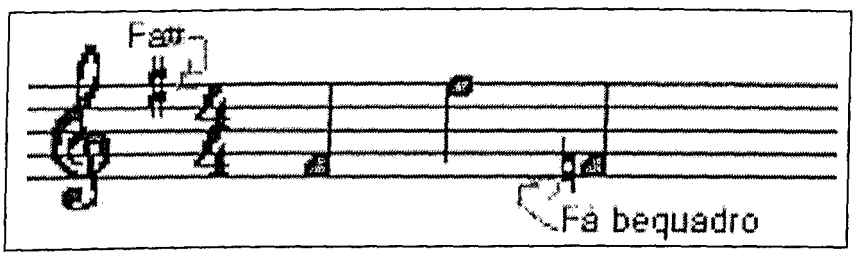


Figura 2.14 – Acidentes na nota

A figura 2.14 mostra um exemplo de armadura de clave com um acidente na nota Fá. Isto indica que todas as notas Fás do pentagrama, sejam qual for altura da nota, serão Fá#, a menos que esta nota seja precedida do símbolo de bequadro, conforme mostrado na figura. Neste caso a nota é um Fá natural (sem acidente),

Assim, as notas são formadas por dois elementos inseparáveis: A duração da nota e sua altura (freqüência).

Um conjunto de notas tocadas simultaneamente, ou não, pode constituir um acorde. Ele pode ser denominado corretamente por diferentes nomes. Assim. quando se for analisar uma peça deve-se levar em consideração a ordem ou a forma com que estes acordes foram tocados. A ordem e a identidade destes acordes diz respeito à harmonia. Esta ordem e a forma de execução do acorde, implícito ou explicito. muitas vezes determinam e identificam o estilo de um autor. O exemplo da figura 2.15 ilustra o fato:

Dados os seguintes pentagramas:

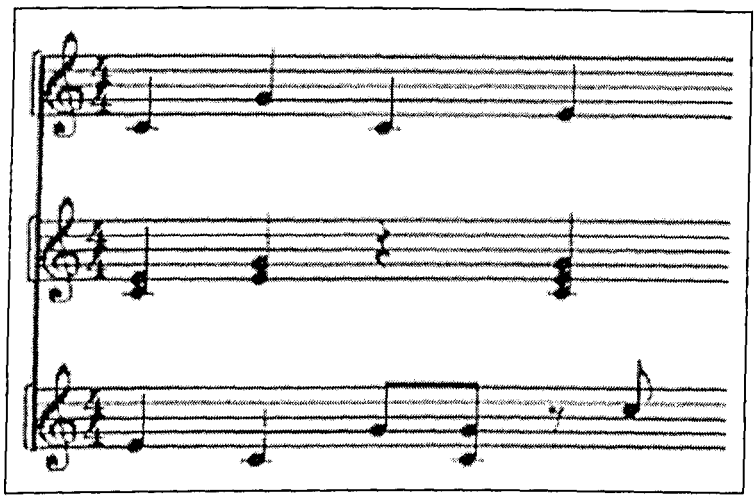


Figura 2.15 – várias formas de se apresentar um acorde

Todos os três pentagramas apresentam um mesmo conjunto de notas (dó, mi e sol) que juntas representam o acorde de Dó maior (C). Desta forma, poder-se-ia denominar qualquer um deles, na base de acordes, como sendo o acorde de C (dó maior). Desta forma. cada um destes acordes receberá um nome especifico que Os diferencie dos demais, como, por exemplo: C1 (acorde de Dó maior do tipo 1). C2 (acorde de Dó maior do tipo 2) e C3 (acorde de Dó maior do tipo 3).

Deve-se esclarecer que este procedimento não é musical, mas adotado nesta dissertação. Isto porque "... El cifrado de los acordes tiene um significado categórico em cuanto a la matenalidad de las notas constitutivas de los acordes, pero carece em absoluto de significado em cuanto al número de voces que deben entrar em juego y em cuanto al curso melódico que cada voz debe seguir para que los acordes se sucedan formando un engranaje em marcha." [2]

[2] Zamacois Joaquim, Tratado de Armonia, Barcelona, Labor, 1982, p 86

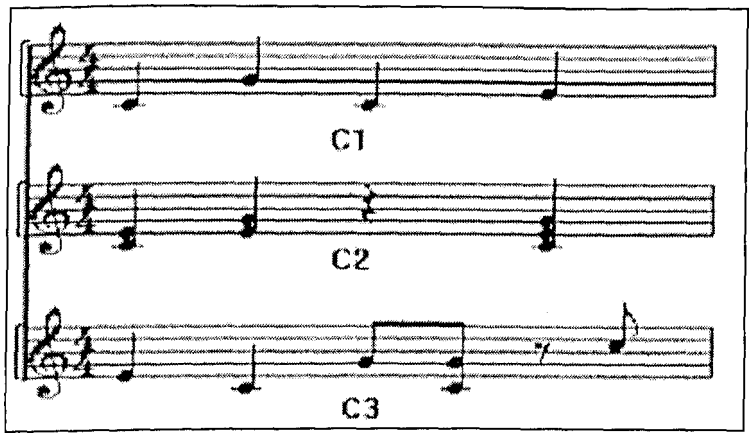


Figura 2.16 – Várias formas de escrever o acorde de C

Em outros casos, tem-se variações nos acordes, como por exemplo:

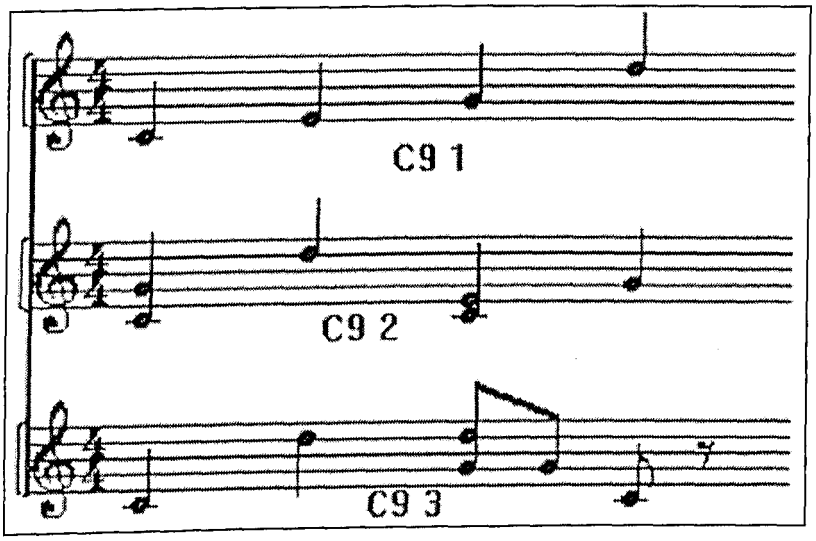


Figura 2.17

Nestes três pentagramas acrescentou-se uma nota Ré às notas Dó, Mi e Sol. com isto se tem um acorde de Dó maior com a nona Como cada um foi escrito de uma forma diferente teremos o acorde de Dó maior com Nona do tipo 1 (C 9 1), do tipo 2 (C 9 2) e do tipo 3 (C 9 3).

Existem três tipos fundamentais de acordes, o Acorde Maior, o Acorde Menor e o Acorde Diminuto. Basicamente cada acorde é formado por três notas, tocadas simultaneamente ou não. Quando não são tocados simultaneamente, diz-se que o acorde é arpejado.

A tabela da figura 2.18 mostra como são formados.

Tipo de Acorde	1ª Nota	2ª Nota	3ª Notas	4ª Nota
MAIOR	No tom	2 tons acima	2 tons acima	variação
MENOR	No tom	1 ½ acima	1 1/2 acima	variação
DIMINUTO	No tom	1 1/1 acima	1 1/2 acima	variação

Figura 2.18

Como exemplo, vamos construir um acorde de Dó maior, depois um de Dó menor e um Dó diminuto.

1 - Dó maior

1ª nota = Dó (nota que dá nome ao acorde)

2ª nota = Mi (dois tons após a nota Dó)

3ª nota = Sol (um tom e meio após a nota Mi)

2.4.4) AS TRÊS FORMAS DE SE ESCREVER UM ACORDE DE CM SÃO:

Figura 2.19

<Acorde com notas simultâneas>< Acordes com notas arpejadas >

A figura a seguir mostra estas notas tocadas em um piano.

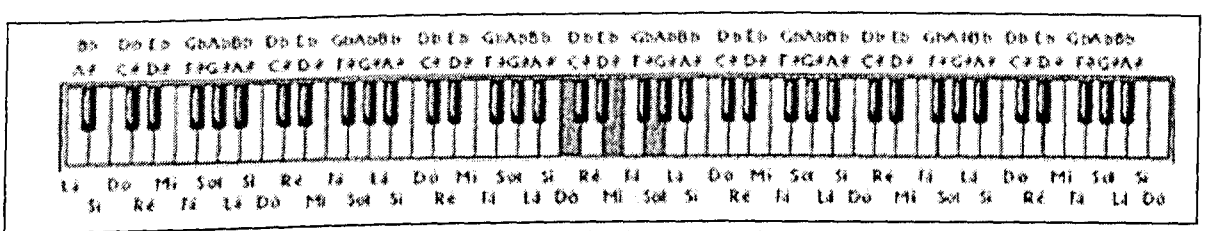


Figura 2.20 – Acorde de Dó no piano

Dó Menor:

1ª nota = Dó (nota que dá nome ao acorde)

2ª nota = Mib (um tom e meio após a nota Dó)

3ª nota = Sol (dois tons após a nota Mib)

Notando em partitura, fica:

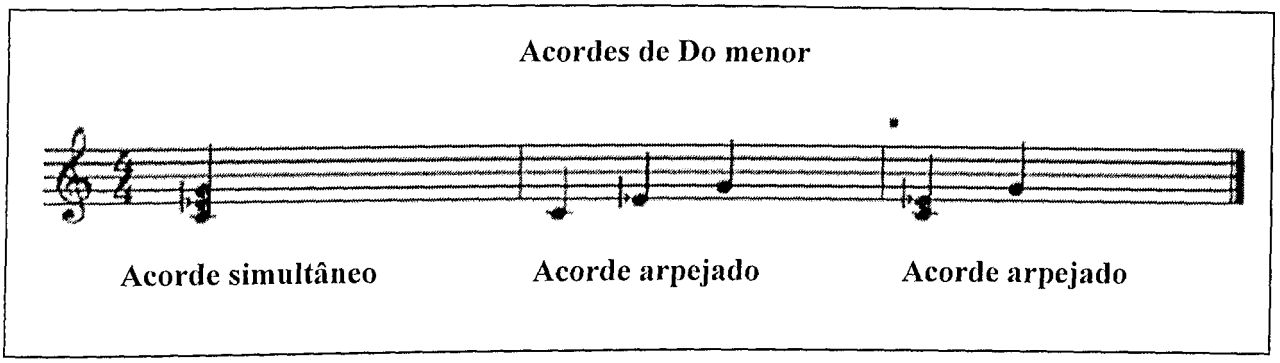


Figura 221

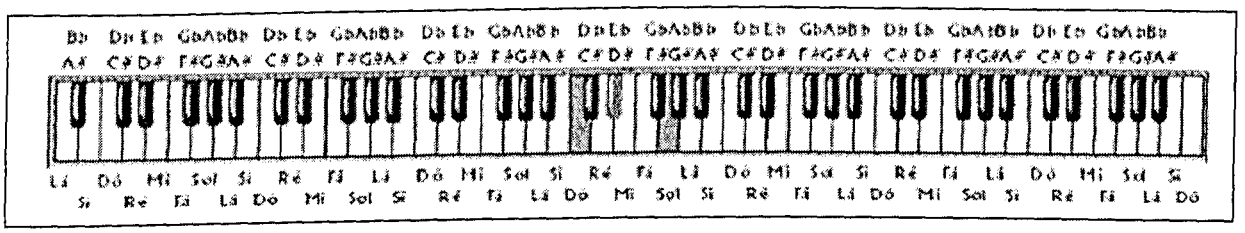


Figura 2.22 - notas tocadas em um piano.

3 – Dó Diminuto

1ª nota = Dó (na nota do nome do acorde)

2ª nota = Mib (um tom e meio após a nota Dó)

3ª nota = Solb (um tom e meio após a nota Mib)

Em partitura fica:

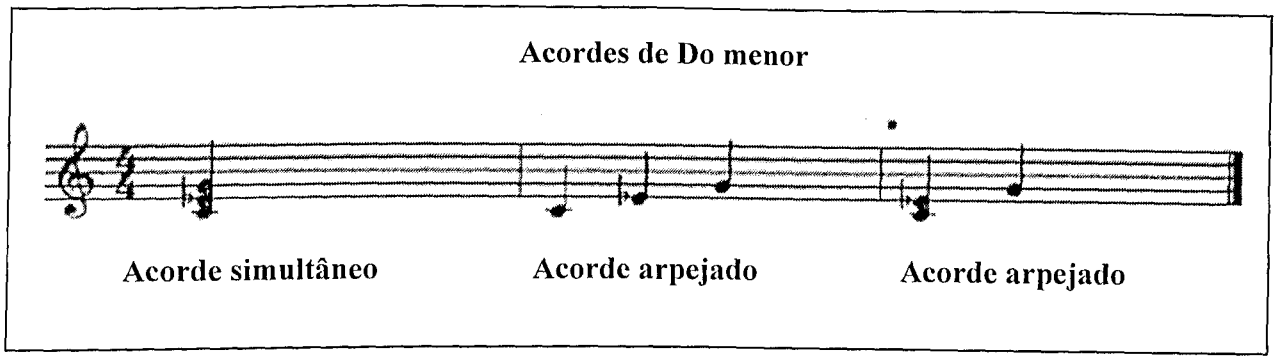


Figura 2.23 – mais arpejados

A figura a seguir mostra estas notas tocadas em um piano.

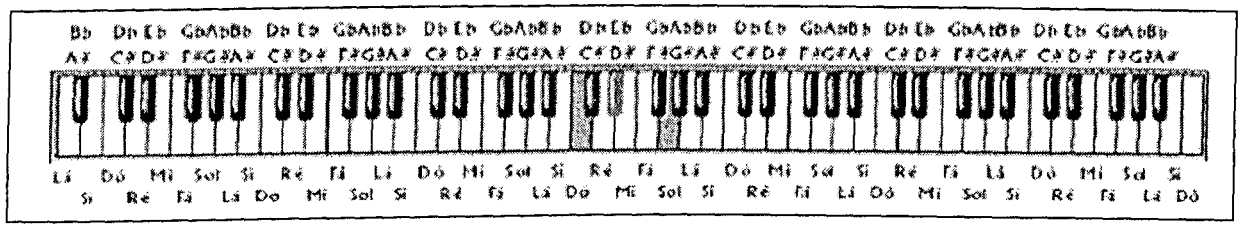


Figura 2.24

Vê-se em exemplos anteriores um outro tipo de acorde, ou seja. com mais de três notas, como foi o caso do acorde de Dó maior com nona (C9). Neste acorde, além das três notas básicas acrescenta-se a nota Ré e diz-se que era um acorde com nona. O que ocorre é que sempre se pode acrescentar uma ou mais notas ao acorde, como o caso de um acorde de Dó maior com sétima e nona ($C^7/_9$). Neste caso, além das notas básicas o acorde possuiria, também, as notas Ré e Sib. Muitas vezes ao invés de se acrescentar uma nota, pode-se substituir duas das três notas básicas. Não se deve substituir a segunda nota porque esta nota define se o acorde é maior ou menor. Damos a esta nota o nome de 3ª maior ou 3ª menor por que ela é a terceira nota da escala a contar da nota do acorde. Veja o caso do acorde de Dó maior, a segunda nota é um Mi ou seja. a terceira nota, contando ascendentemente da nota Dó (Dó – Re – Mi). O Mi e a 3ª maior do acorde de Dó. No caso do acorde de Dó menor a terceira nota utilizada foi o Mib. O Mib é a terça menor do acorde de Dó. Assim se se utilizar a 3ª menor em um acorde, ele é menor, e, se utilizar a 3ª maior ele é maior.

A tabela da figura 2,25 mostra todas as variações de notas que se pode acrescentar aos acordes fundamentais, com seus respectivos nomes.

A convenção em alguns casos não é uniforme e podemos ver claramente este caso aqui. O "m" significa menor, o "M" significa maior o "J" significa justa. De 9 em diante tem-se os sinônimos de 2 em diante.

T O M	2 m	2	3 m	3 M	4	5 m	5J	5 M	6	7 m ou 7	7 M	8	9 m	9	9 M	10	11	11 M	12	12 M	13
A	A#	B	C	C#	D	D#	E	F	F#	G	G#	A	A#	B	C	C#	D	D#	E	F	F#
A#	B	C	C#	D	D#	E	F	F#	G	G#	A	A#	B	C	C#	D	D#	E	F	F#	G
B	C	C#	D	D#	E	F	F#	G	G#	A	A#	B	C	C#	D	D#	E	F	F#	G	G#
C	C#	D	D#	E	F	F#	G	G#	A	A#	B	C	C#	D	D#	E	F	F#	G	G#	A
C#	D	D#	E	F	F#	G	G#	A	A#	B	C	C#	D	D#	E	F	F#	G	G#	A	A#
D	D#	E	F	F#	G	G#	A	A#	B	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
D#	E	F	F#	G	G#	A	A#	B	C	C#	D	D#	E	F	F#	G	G#	A	A#	B	C
E	F	F#	G	G#	A	A#	B	C	C#	D	D#	E	F	F#	G	G#	A	A#	B	C	C#
F	F#	G	G#	A	A#	B	C	C#	D	D#	E	F	F#	G	G#	A	A#	B	C	C#	D
F#	G	G#	A	A#	B	C	C#	D	D#	E	F	F#	G	G#	A	A#	B	C	C#	D	D#
G	G#	A	A#	B	C	C#	D	D#	E	F	F#	G	G#	A	A#	B	C	C#	D	D#	E
G#	A	A#	B	C	C#	D	D#	E	F	F#	G	G#	A	A#	B	C	C#	D	D#	E	F

Figura 2.25

Com o auxílio desta tabela pode-se implementar qualquer acorde. Como exemplo, montar-se-á o acorde A^4_6 (Lá maior com quarta e sexta). A primeira regra a se considerar é que um acorde é formado, normalmente, de fundamental (nota que dá nome ao acorde), terça e quinta.

A^4_6

1ª nota = Lá (nota que dá nome ao acorde)

2ª nota = Dó# (dois tons após a nota Lá ou ainda, a terceira nota após Lá: Lá-Si-Dó#)

3ª nota = Mi (um tom e meio após a nota Dó#, ou ainda, a quinta após Lá; Lá-Si-Dó-Ré-Mi).

4ª nota = Re (a quarta nota -> numero 4 da tabela, na tonalidade de A).

5ª nota = Fá# (a sexta nota -> número 6 da tabela, na tonalidade de A).

Muitas vezes é necessário transpor-se urna música. Transpor a tonalidade é uma tarefa simples Utilizar-se-á a figura a seguir, onde se colocou cada uma das doze notas musicais no lugar de um dos números de um relógio.

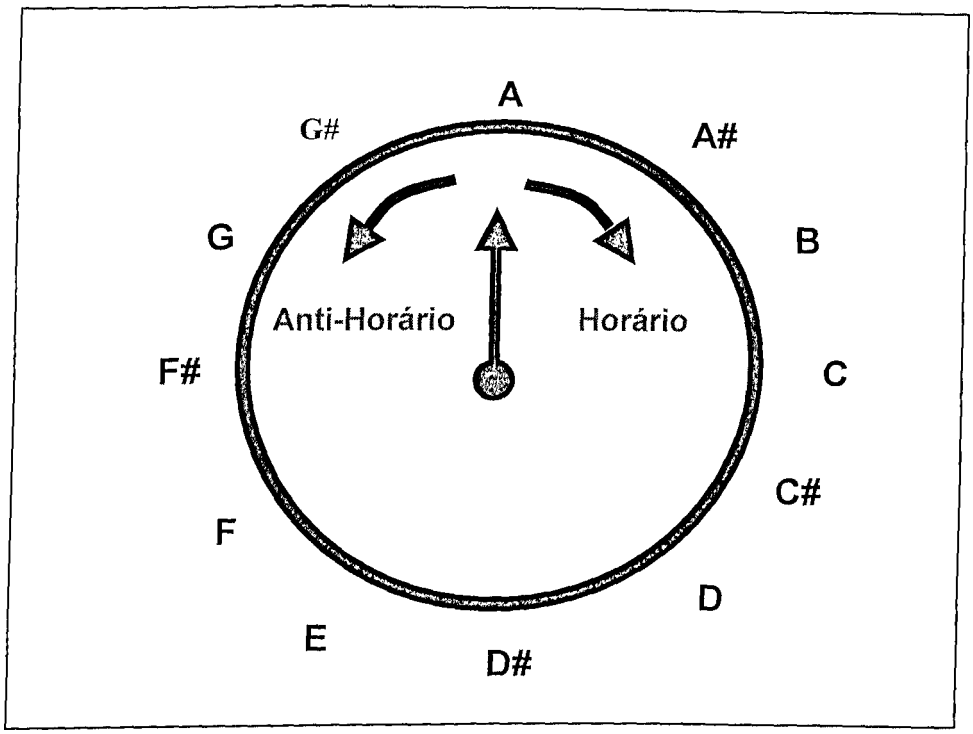


Figura 2.26 – mudança de tonalidade

Supor-se-á uma seqüência de acordes na tonalidade de Dó maior como a dada a seguir, e se deseja transpô-la para Lá maior:

Seqüência = C9 -> Dm -> Em G^4_7 -> G7 -> C

Para transpor esta seqüência basta inicia-la com o acorde de Lá maior seguido da mesma variação, no caso a nona.

O 1°. elemento da seqüência nova fica = A9 ->

Para descobrir quais são os novos acordes que seguirão A9 basta analisar a seqüência original e utilizar o gráfico transpositor dado.

Na seqüência original de C9 foi-se para Dm. No Gráfico da figura 2.26 de transposicão, nota-se que partindo de C, para atingir D anda-se no sentido horário dois acordes (C# e D). Assim da mesma forma, partindo de A tem-se que andar dois acordes (A# e B), ou seja, o acorde de B. Como na seqüência original o acorde de D é menor (Dm), acorde da nova seqüência também é menor (Bm).

A seqüência nova fica ->A9 -> Bm ->

Agora partindo da seqüência original, de D, anda-se no sentido horário novamente dois acordes (D# e E), encontra-se o acorde de E. Da mesma forma, partindo de B anda-se dois acordes no sentido horário (C e C#), caindo no acorde de C#. O acorde é menor porque o original também era.

A seqüência nova fica -> A9 -> Bm -> C#m ->

Partindo da seqüência original do acorde de E, anda-se três acordes no sentido horário (F, F# e G) e se encontra o acorde de G. Da mesma forma, partindo de C#, anda-se três acordes no sentido horário (D, D# e E) e se encontra o acorde de E. No caso, o acorde será de E^4_7 porque o original era com 4^a. e 7^a.

A seqüência nova fica -> A9 -> Bm -> C#m -> E^4_7 ->

Como o acorde final não mudou, ou seja, foi de G^4_7 -> G7, na seqüência nova o acorde vai de E^4_7 -> E7.

A seqüência nova fica -> A9 -> Bm -> C#m -> E^4_7 -> E7->

Finalmente, o último acorde da seqüência volta para a própria tonalidade, com apenas a diferença de que um era com nona e o outro não. Na nova seqüência isto também deverá ocorrer, portanto, a nova seqüência completa fica:

Nova seqüência = A9 -> Bm -> C#m -> E^4_7-> E7-> A

CAPITULO III

O PADRÃO MIDI DO SMF

Nesta dissertação não se tratará dos detalhes da configuração do hardware MIDI. Os esforços estarão concentrados tão somente na criação de urna interface de software.

Como o fito de melhorar a compreensão, seguir-se-á com a metodologia empregada até aqui, que é a de exemplificar mais que teorizar qualquer aspecto do MIDI. Os exemplos dos capítulos anteriores serão de grande valia e se fará referência a eles sempre que necessário.

Este capítulo tratará do que se convencionou chamar SMF – Standard Midi Files. Este é um padrão definido por meio do qual se escrevem arquivos MIDI, e como o nome diz, possui a vantagem de ser aceito por todos os editores musicais, instrumentos com circuitos MIDI, módulos de som etc.

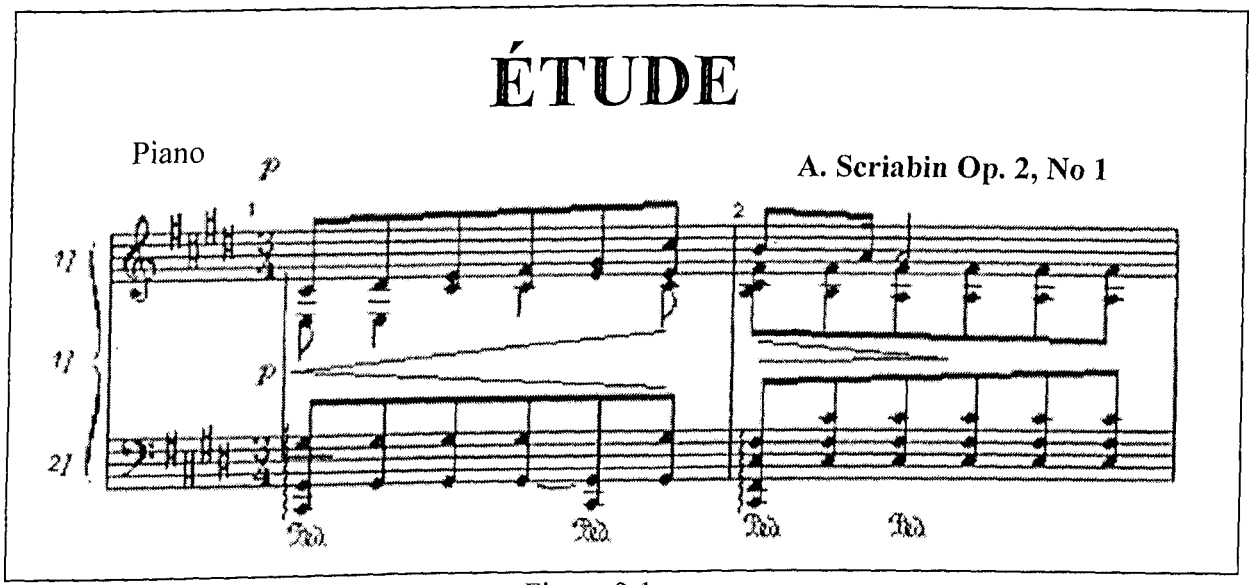


Figura 3.1

Relembrando o que foi dito no capitulo II, a fórmula de compasso dá a distribuição rítmica das notas num compasso. Se se tem, por exemplo, 3/4 isto quer dizer 3 notas do tipo (1/4). Procurando na figura 2.4 da página 17 vê-se que (1/4) é a semínima e, portanto, cada compasso da música deve ter como unidade de tempo 3 semínimas. Mas porque então na figura 3.1 não se vê nenhuma semínima? A resposta é simples. No exemplo acima a semínima é a unidade de tempo, ou seja, ela é o pulso que

neste caso é forte-fraco-fraco, é o acento. Isto serve para se enquadrar o ritmo, ou de outra forma, são as pilastras temporais. Por isto também que se disse anteriormente que o que era relevante, em se falando das notas, era a relação delas entre si e entre a semibreve. No exemplo, deve-se pensar que cada semínima pode ser dividida em duas colcheias e que esta mesma semínima é a metade de uma mínima, portanto, pode-se facilmente conferir que no primeiro compasso seis colcheias preenchem-no e no segundo, na voz superior, a melodia é dividida em colcheia pontuada (que é o mesmo que o tempo de uma colcheia mais o tempo de uma semicolcheia), semicolcheia e uma mínima, pelas relações já apresentadas na figura 2.4 e, fazendo uma conta simples, chega-se a conclusão que também o compasso 2 está devidamente preenchido.

Falou-se no parágrafo anterior de vozes. No compasso 2 têm-se três vozes, ou seja, pode-se pensar este trecho coma sendo um coro em que cada conjunto de vozes canta uma melodia diferente, como mostrado. Podemos deduzir deste compasso a essência da música polifônica, que é a constante, mas não necessária, independência das vozes. Outra coisa que se deve notar é que a independência de qualquer voz não a exime de seguir a rígida marcação imposta pela unidade de tempo e de compasso, expressa na fórmula de compasso.

Um dos princípios básicos da interpretação musical é o andamento que quer dizer da velocidade com que se marca as unidades de tempo. Esta velocidade é controlada pelo *set tempo;* é a unidade de tempo. Ela é representada pelo *ppq*.

3.1) ARQUIVO SMF:

O *SMF* foi criado na década de oitenta como uma proposta de se unificar um método de reprodução de som por computador. O padrão que primeiramente fora codificado diretamente no hardware, para os primeiros módulos de som, passou a ser adotado como padrão oficial em software, quando os primeiros editores de partitura surgiram. A metodologia prevê três tipos de formato: O formato 0, 1 e 2. O formato 2 raramente é usado. O *SMF* foi idealizado de maneira a receber acréscimos futuro já que os criadores do padrão, acertadamente, previram a evolução *hardware-sotfware* e da música, sendo necessários novos recursos. Hoje o formato mais usado é o 1, e este é objeto deste trabalho. Vale dizer que a formato 0, por ser mais restrito, é mais complexo de se implementar.

O código *MIDI* é enviado seqüencialmente e em tempo real de um instrumento *MIDI* para outro, incluindo computadores. Quando um instrumentista emite um som em seu instrumento *MIDI* este som é traduzido no padrão *SMF* em termos da duração da nota (o quanto o executante permanece emitindo a som) e da altura da nota, ou seja, qual nota se está tocado! Além das pausas que traduzem silêncios. O *MIDI* deve simular as ações efetuadas pelos instrumentistas, enviando eventos ao módulo de som ou outro equipamento *MIDI* associado de forma paralela e concorrente. Vários instrumentistas tocam, ao mesmo tempo, as partes que lhes cabem no conjunto. Às vezes e necessário que isto se faça de maneira concorrente já que pode ocorrer de se ter apenas um canal de saída para dois ou mais instrumentos e por isto devem ser estabelecidos métodos para que eles tenham no tempo, pelo menos a ilusão de simultaneidade. Este problema já foi resolvido ao ser criado o hardware MIDI.

O MIDI possui uma metodologia de controle de tempo baseado num contador *up-down.* Ele funciona assim. O contador *Up->Down* armazena os chamados *delta-times*. Estes *delta-times* podem ter no máximo o comprimento de dois bytes. Cada byte possui 7 bits e eles contam de 00H a 7FH (de 0 a 127 em decimal). Quando um delta-time for menor ou igual a 127 (7FH) ele será representado por apenas um byte com oitavo bit igual 0, caso seja maior o byte superior sempre será representado pela equação (n* 128) onde n e o numero de 128s a serem contados. De fato se tem aqui um contador de 128s. O evento para a execução de uma nota ou pausa segue os seguintes passos:

1) Carrega-se o contador *up-down* de *delta-time* com o valor desejado.

2) Logo após o sistema carrega a mensagem de qual evento deverá ser executado.

3) A execução do evento só acontece quando a contador chegar a zero. Quem determina a velocidade com que o contador será decrementado é o valor do *ppq* (que será explicado mais tarde, *PPQ* quer dizer *pulses per quarter note*, é parte do metrônomo adotado).

4) Carrega-se o contador com um novo valor de *delta-time*.

5) Repete-se todo o processo até encontrar o código FF2F00 (fim de *track*).

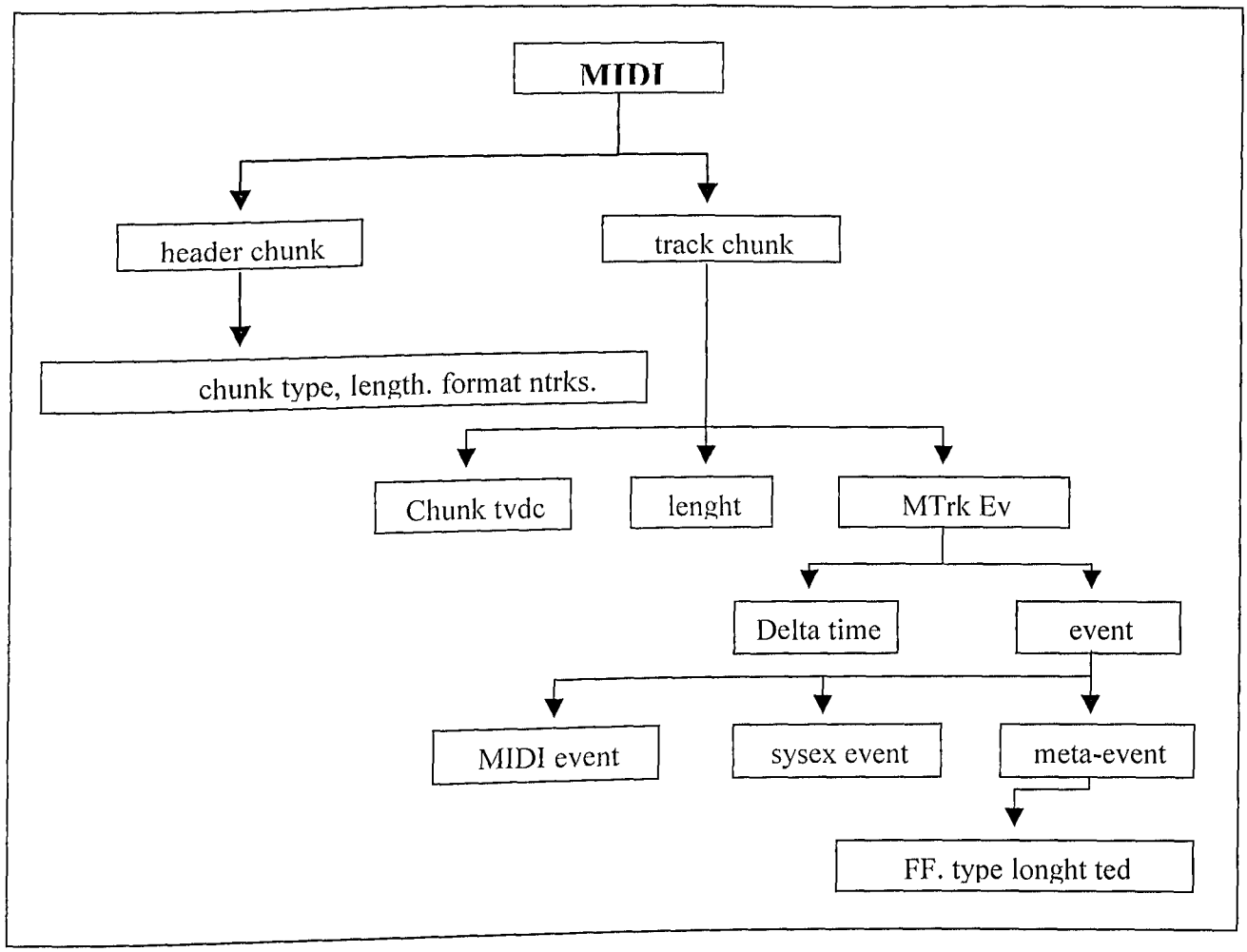


Figura 3.2 – Estrutura geral do protocolo MIDI

A figura 3.2 mostra as subdivisões do protocolo MIDI. Este esquema será exemplificado adiante.

Um arquivo MIDI é escrito no formato: 4D 54 68 64 00 00 00 00 06 00 01 00 02 04 00 4D 54 72 6B 00 00 00 1A 00 FF 58 04 04 02 etc.

Esta longa linha de bytes possui um esqueleto que lhe dá sentido e sem o qual o formato não seria reconhecido. O trecho acima é formado de grupos que para serem percebidos tem-se que examinar a figura 3.2. O formato a ser estudado é, como foi dito, o formato MIDI 1.

O arquivo MIDI é composto apenas de dois cabeçalhos o ***Header Chunk*** e o ***Track Chunk***. As subdivisões subseqüentes ampliam o nível de detalhamento da estrutura. Alguns destes níveis são optativos. Veja como são montados:

3.1.1) *HEADER CHUNCK*

<Header Chunk> = *<Chunk type><length><format><nrtrks><division>*

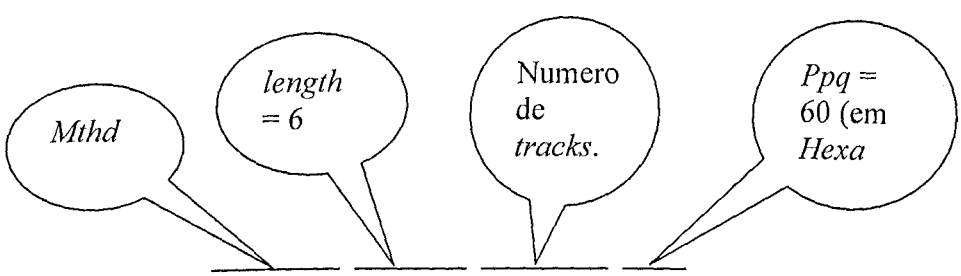
<Chunk type> = *4D546864*, valor constante – São as letras *Mthd* e possuem o tamanho fixo de quatro *bytes*. São elas que identificam o arquivo como sendo um arquivo *MIDI*.

<length> = <comprimento em *bytes* do que vem a seguir> – Este parâmetro é a soma dos tamanhos dos parâmetros *<format>* dois *bytes*, *<ntrks>* dois *bytes* e *<division>* dois *bytes*. Na grande maioria dos casos *<length>* é fixado em seis.

<format> = 0000, 0001 ou 0002 – São os formatos *MIDI* 0,1 e 2 já citados. Tamanho: dois *bytes*.

<ntrks> *nnnn* – É o numero de *tracks*. No formato 0 só há um *track*. Nos formatos 1 e 2 tem-se um *track* para cada canal *MIDI* mais um para o *setup* da música (metrônomo, armadura de clave, instrumento por canal etc). Os *tracks* são pacotes musicais, com eles, por exemplo, podemos escrever urna grade de orquestra. Tamanho: dois *bytes*.

<division> Determina o tipo de *delta-time*. Se o *bit* mais significativo for 0, os *delta-times* serão do tipo *ppq* se for 1 serão do tipo *SMPTE* que é o tipo de sincronismo som imagem usado em filmes. Aqui se usará o tipo *PPQ*.



<Header Chunk> = 4D546864 0000000 600000001 0060

Figura 3.3 – O que representam as partes do cabeçalho *Header Chunk*

3.1.2) *TRACK CHUNCK*

<Track Chunck> = <chunk type><length><MTrk event>

<chunck type> = 4D54726B — São as letras *MTrk*. Tamanho: quatro bytes. São elas que assinalam o começo do *track*.

<length> Soma de todos os *bytes* após ele, incluindo os três *bytes* de fim de *track* – *FF2F00*. Tamanho: quatro *bytes*.

<MTrk event> por sua vez é divido em *delta-time e event*.

<MTrk event> = <delta time> <event>

<delta time> = Valor em *hexa* de quanto tempo o dispositivo que está lendo e executando o arquivo *MIDI* deverá esperar para iniciar a execução do evento que o segue simbolizado por *<event>*, mas como se faz o cálculo do *delta-time*? A seguir é mostrado um exemplo que pode ser facilmente generalizado.

Como já foi dito, o *delta-time* simboliza a duração de uma nota ou uma pausa, sendo assim se construirá um exemplo onde *ppq=6OH*. Dois valores de notas serão implementados: semínima e semínima pontuada. O procedimento de cálculo é o seguinte:

1) Semínina: supor-se-á aqui que o *ppq* representa a semínima que neste exemplo é adotada como unidade de tempo, portanto, o valor da semínima será de *60H*.
2) Semínima pontuada; Uma semínima pontuada é *60H + 30H* que é *90H*, mas *90H>7FH*, portanto, deve-se usar dois *bytes* para representar a duração da nota para isto se faz *(96+48)/128= 1.125*, valor em decimal. O *byte* menos significativo é determinado peta mantissa, assim *0.125*128=16=>10H*, valor em hexadecimal. O *byte* superior é formado pela parte inteira grafada em hexadecimal, ou seja, 1. O número fica então *110H*, mas ainda se deve *setar* o bit mais significativo para

1(1XXX XXXX) isto é feito acrescentando-se 8 a *110H* na posição mais significativa produzindo *8110H*. Este é o *delta-time* de uma semínima pontuada com *ppq* = *60H*. O valor *8110H* irá mudar caso o *ppq* seja outro.

<event> = *<MIDI event>* | *<sysex event>* | *<meta event>*

<MIDI event> = Qualquer mensagem de canal, ou seja, eventos de notas e os controles aplicados a elas. O primeiro evento do *MTrk*, após o *delta time* correspondente, sempre deverá ser um *byte* de evento de *status* (com o *bit* mais significativo igual a 1), seguido de *byte* de dados (*bit* mais significativo igual a zero). Quando os dados que seguem o *byte* de *status* utilizam o mesmo *status*, pode-se apenas colocar um único *byte* de *status* para todos os *bytes* de dados que o seguem.

<sysex event> = F0 *<length><bytes* a serem transmitidos> – Evento utilizado para mandar mensagens exclusivas para um determinado equipamento.

<length> = número de *bytes* transmitidos na mensagem exclusiva de sistema

<bytes a serem transmitidos> = deverão terminar com o *byte*: F7.

<meta~event> FF<type><length><text> – Eventos não *MIDI* contendo informações necessárias para os equipamentos que executarão os eventos MIDI. Um *meta-evento* sempre inicia com o byte *FF*. É particularmente difícil compreender a função de um *meta-event*. Com ele pode-se iniciar eventos dos mais variados tipos e para mostrar seu funcionamento é que se produziu os quatro exemplos abaixo. Eles deverão elucidar muito mais que sua definição formal.

<type> = tipo do *meta-evento*.

1) Neste trecho do protocolo *MIDI* é que, caso se deseje, se coloca o nome da música para que algum interpretador *MIDI* possa ler. Para isto *type= 06*. O nome da música, então, deve aparecer em *<text>* e *<length>* deve conter o tamanho deste texto.

2) Caso *type=51* significa que o *meta-evento* será o *"set tempo"*. Este *meta-evento* controla o andamento da música.

Quando *type=51 e <length>=03 bytes, o* set tempo fornece um valor de tempo em microssegundos que dá o andamento da música, em três bytes, que na verdade é parte do metrônomo. O trecho todo para o *meta-evento set tempo* seria: FF 51 03 09 89 68. O número (098968 microssegundos) hexa é (625.000 microssegundos) decimal. Convertendo-se para segundos dariam 0,625s Então a unidade de tempo (u.t) seria os 0,625s conjugados com o PPQ e o *MIDI clock* pela fórmula: (*MidiClock* setTempo/ppq*). Caso o *ppq=60H* ou (96) decimal, tem-se 96 divisões de 0,625s ou 0,625 s/96 = 0,00651 *midi-clocks*. 0,00651* 24 *midi clocks* (valor da u.t) = 0.15624s que é a duração no tempo de uma u.t. Sem este procedimento não seria possível para o computador, por exemplo, soar a duração correta da nota colcheia num trecho de música onde a unidade de tempo fosse a semínima. É necessário para a interpretação correta do fenômeno musical, a relação das notas entre si e em relação a um marcador externo, no caso o *set tempo* baseado no *clock* da máquina.

3) Caso type = 58 significa que o *meta-evento seta* o *"time signature"* (fórmula de compasso) *<length>*= 4 *bytes*. O formato destes quatro *bytes* será *nn dd cc bb*, onde nn=numerador, dd=denominador, que no *MIDI* é definido em termos de múltiplo de 2, *cc*—número de *MIDI clocks* por unidade de tempo (já estabelecido pelo *set tempo e ppq*) e *bb*= número de fusas por unidade de tempo. Poder-se-ia ter *FF 58 04 04 02 18 08* significando que a música tem compasso *4/4, 18H MIDI clocks* por unidade de tempo e precisão de 8 fusas por unidade de tempo sendo *18H/8H ou 24/8 = 3 MIDI clocks* por fusa.

4) Caso *type=59* significa que o *meta-evento* será o *"Key signature"* (armadura de clave). *<length>=2 bytes*. O primeiro *byte* é conhecido como ***sf*** e representa os acidentes da armadura. Ele pode variar de -7 a +7 (mas em *hexa*). Os negativos são os bemóis e os positivos os sustenidos. Se por exemplo se tem -3 significa três bemóis na armadura e como os bemóis na armadura obedecem a uma ordem eles seriam: Sib, Mib e Lab. O segundo

byte é conhecido como ***mi*** e caracteriza o modo da música. Se mi=00 é o modo menor se mi=01 caracteriza o modo maior. O trecho *FF 59 02 00 00 FF é um meta-evento key signature*, em CM.

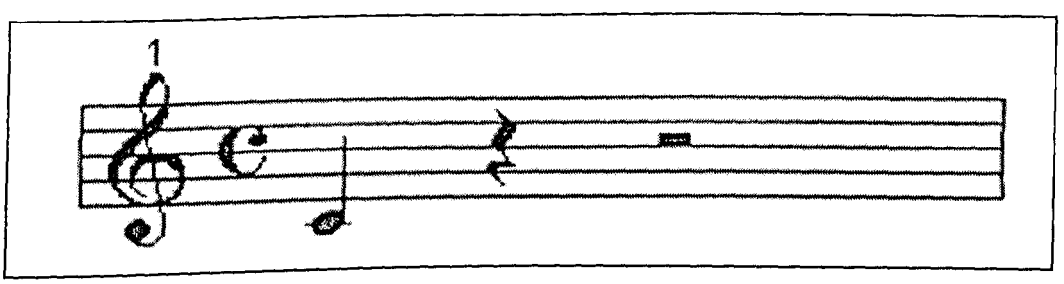
O que foi exposto acima diz respeito ao primeiro *track* do formato 1, ou seja, o cabeçalho de notas com configurações iniciais. Este *track* é finalizado com a seqüência *FF2F00*. Um segundo *track*, este sim, traz as notas propriamente ditas e é onde os *delta-times* fazem o seu melhor trabalho.

3.1.3) O *TRACK* DE NOTAS

O *track* de notas é o pacote onde estão as notas a serem executadas. Estas notas só poderão ser executadas após os *sets* mostrados anteriormente. Existe mais de um modo de se representar estas notas. O editor de partituras *FINALE* usa, para escrever, um modelo mais prolixo (e menos econômico no espaço) o qual será adotado aqui. A estrutura básica é a seguinte:

<MTrk><length><channel><instrument><notes><00FF2F00>. Lembre-se que para cada conjunto de notas e necessária a descrição abaixo:

<notes><np><90><note><vol> <dt><80><note><00> onde ***np*** é o valor de *delta time* das pausas que antecedem uma nota, ou seja, o quanto se tem que esperar para soa-la. ***Vol*** é o volume com que a nota deve soar (o que define a dinâmica), ***90*** "aciona" a nota, ***80*** "desliga" a nota. *Note* é a nota (representada em hexadecimal). ***Dt*** é o *delta time* da nota, ou seja, o quanto se deve esperar para desligar a nota, isto quer dizer da duração mesmo da nota.



O Exemplo a seguir é de um arquivo MIDI real:

4D 54 68 64	MThd
00 00 00 06	seguem 6 bytes
00 01	formato MIDI 1
00 02	irá se usar 2 *tracks*
04 00	*ppq* 400(*hexa*) dá em quantas partes o tempo, em segundos, está dividido.
4D 54 72 6B	*MTrk* - começo do *track*
00 00 00 1A	seguem *1A bytes*
00	*delta time*
FT 58	*Time Signature*
04	seguem 4 *bytes*
04 02	fórmula de compasso, no caso 4/4
18	*MIDI clocks* por unidade de tempo (aqui é a semínima)
08	número de fusas por unidade de tempo.
00	*delta time*
FT 59	*Key Signature*
02	seguem 2 *bytes*
00	*sf* = 0, nenhum acidente - C
00	*mi* = 0, modo maior
00	delta time
FF 51	*Set Tempo*
03	seguem três *bytes*
09 89 68	625.000 microssegundos. Seria 0,625 s
00	delta time
FF 2F 00	fim do *track*
4D 54 72 6B	*MTrk* — começa novo *track*
00 00 00 1F	seguem 1F *bytes*.
00	delta time
C0	*seta* o canal de som 0
00	instrumento piano
00 90 3C 40	*delta time* 00, ligar nota (90), 3C(dó central) com volume 40
88 00 80 3C 00	delta time 8800, desligar nota (80), nota (3C) com volume 00
01 FF 2F 00	fim do *track*

Repare que ppq = 400 (1024 em decimal) e que 1024/128 = 8 (8 em hexa), portanto, 8800 é o valor de uma semínima.

Caso se queira mudar o canal, o instrumento, a fórmula de compasso, a unidade de tempo, a armadura de clave e outras configurações é possível fazê-lo a qualquer momento, basta *setar* o *meta-evento* antes da nota e a partir daí, valerão as novas configurações até que se estabeleça outro *meta-evento* e assim por diante, mas porque não fazê-lo no *track* de notas e porque se precisa de dois *tracks*? É que a *MIDI* não tem configuração *default*, então, é preciso *seta-las* ao menos uma vez.

CAPITULO IV

IMPLEMENTAÇÃO DO MIDI EM CLEAN

Como as figuras de 1 a 9 mostram, a tarefa de se produzir um arquivo MIDI está distribuída por vários módulos.

Este capítulo trata da implementação dos módulos das figuras de 3 a 6 porque eles produzem o arquivo MIDI. Os módulos das figuras de 7 a 9 produzem a interface para obtenção das notas numa partitura e serão tratados no capitulo V.

Não se explicará o módulo *Tokens21* por se tratar apenas de um produtor de *tokens* cuja listagem está na seção anexos.

A tarefa de explicação das partes do projeto sempre irá considerar urna leitura de baixo para cima na relação-dependência dos módulos, assim a explanação inicia-se no módulo *lexMidiO899* seguindo até o módulo *gerl199*.

O trabalho do módulo *lexMidi0899* é reconhecer urna nota, o que quer dizer seu nome, duração, altura e possíveis *midi-eventos*.

4.1) O MÓDULO *LexMidi0899*

Para que todo processo tenha início deve-se "quebrar" o texto original em *tokens*. Veja o exemplo abaixo. Aqui há apenas um compasso que irá passar pelo analisador léxico.

```
inVal :== "1- ([ME(0 FF 58 4 2 2), ME(0 FF 2F 0)]
                    B5 (rn), C#5 (sm ..), [ME(0 FF 58 4 2 2)]B5 (sc));";
Start = Tokens (inVal);
```

Produz:

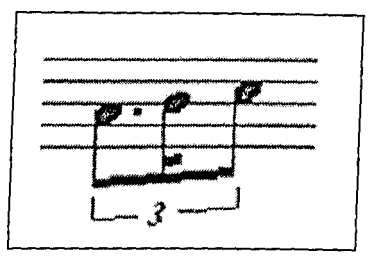
```
[(Ti 1),(Ts "-"),(Ts "("),(Ts "["),(Ts "ME"),(Ts "("),(Ti 0)
        (Ts "FF"),(Ti 58),(Ti 4),(Ti 2),(Ti 2),
        (Ts ")"),(Ts ","),(Ts "ME"),(Ts "("),(Ti 0),
        (Ts "FF"),(Ti 2),(Ts "F"),(Ti 0),(Ts ")"),
        (Ts "]" ),(Ts "B"),(Ti 5),(Ts "("),(Ts "m"),
        (Ts ")"),(Ts ","),(Ts "C"),(Ts "#"),(Ti 5),
```

(Ts "("),(Ts "srn"),(Ts ". "),(Ts "."),
(Ts ")"),(Ts "."),(Ts "B"),(Ti 5),(Ts "("),
(Ts "sc"),(Ts ")"),
(Ts ")"),(Ts ";")]

Determinou-se que as quiálteras devam ser escritas como vem a seguir: *"2 - (D5 (c 3), ES (c 3), F5 (c 3), 05 (sm), AS (m));"* este compasso mostra uma quiáltera com três notas diferentes. O número 3 na expressão *(c. 3)* significa justamente que esta nota faz parte de uma quiáltera de 3 notas.

Cada quiáltera do conjunto não precisa ter a mesma duração assim o compasso abaixo também é possível:

"3 - (D5 (c. 3), ES (sc 3), F5 (c. 3), G5 (sm), A5 (m));" o que equivaleria ao modelo rítmico:



Acontece aqui um fato que deve ser considerado. Os editores de partitura presentes no mercado, até a presente data, têm dificuldade de ler esta estrutura, dentre várias, de um formato *MIDI* e como o formato nasceu e se propõe a ser o TXT da música isto não poderia dar-se, no entanto, não há problema de interpretação pelos principais hardwares do mercado, nem nesta proposta cujo texto, obedecendo a urna estrutura explicada adiante, deixa claro como interpretar a figura em questão.

Um teste seria:

Start = (findPointValue (AllRhy "c" 3.0) 1) + //Colcheia pontuada
(findPointValue (AllRhy "sc" 10) 0) + // semi-colcheia
(findPoint Value (AllRhy "c" 3.0) 0); // colcheia.

As três expressões, juntas, formam uma quiáltera. O resultado é 1, como esperado, ou seja, o tempo de uma semínima.

Para testar outras saídas use a função *Duration* e suas subordinadas.

Em termos de gramática este módulo seria escrito como vem a seguir. Algumas

destas regras são pertinentes ao analisador sintático, mas por questão de clareza são colocadas aqui (Léxico+sintático). O analisar sintático está explicado no item 4.2.

ALFAE-> ((0-9|A-F)$^+$.(0-9|A-F$^+$)) *produz: A0|00|0B|11 etc*

MAISALFAE-> " " . ALFAE | e *produz: _00*

EVENTOSUB-> MAISALFAE . EVENTOSUB | e *produz: 00 09 ....*

EVENTO-> "ME(" . EVENTOSUB . ")" *produz: ME(00 09...)*

MAISEVENTO-> "," . EVENTO . MAISEVENTO | e *produz: ME(00 09...)*

EVENTOS-> "[" . EVENTO . MAISEVENTO . "]" | e *produz [ME(00 0F...),...]*

*TEMIPO-> (b|sb|m|sm|c|cs|f|sf|q|sq)$^+$. (". ") . [0-9]** *produz c.3*

RITMO-> "(" . TEMPO . ")" *produz: (c.3)*

NOTAALTURA-> [A-G]$^+$. (#|##|&|&|&|%) . [0-9]$^+$* *produz: A#4, G5 etc*

NOTA-> NOTAALTURA . RITMO *produz: A#4 (C.3)*

MAISNOTA-> "," . EVENTOS . NOTA . MAISNOTA | e
produz: A#(c.3), [eventos] G5(m), etc

NOTAS-> EVENTOS . NOTAS . MAISNOTA

COMPASSO-> INTEIRO . "-". "(". NOTAS

MAISCOMPASSO-> ";" . COMPASSO. MAISCOMPASSO | e

PARTITURA-> COMPASSO . MAISCOMPASSO

MAISVOZES-> "VOICE" . PARTITURA . "FINALVOICE" . MAIS VOZES | e

VOZES-> "VOICE" . PARTITURA. "FINALVOICE" . MAIS VOZES

4.1.1) Declaração de tipos:

```
Notation= Nt (String, String, Int, Real)|
        Nm String | Oc Int | Dr Real |
        Sr String | Nr Int |
        M [(String, String, Int,Real)] |
        C [Notation] |
        ME [String] |
        Nil;

Result = OK(Notation, Lk]) | Fail;
```

O tipo *Notation* é ,na verdade, uma coleção de tipos. Estes tipos descrevem a música a partir de elementos tais como notas, nomes das notas, melodia, compasso etc.

Esta abstração é característica da abordagem que se adota e não tem nada de definitivo. É apenas uma forma de modelar de forma prática o problema.

O tipo *Result* é uma tupla contendo no primeiro elemento qualquer um dos tipos da notação musical descrita em *Notation* e no segundo a coleção de *Tokens* ainda não tratados.

Nt (String, String, Int, Real) é um tipo que tem a função de representar uma nota, por exemplo, *Nt (" ÿX ", "G",5,1)*. Aqui se vê que:

a) O primeiro elemento da tupla é um *midi evento* que inicia a nota.
b) O segundo elemento mostra que a nota é G.
c) O terceiro elemento mostra que a altura de G é 5.
d) O quarto elemento mostra um número real que representa a duração desta nota, no caso i, ou seja, é uma mínima.

Nm é o tipo que representa o nome da nota, por exemplo, *Nm* "A", Nota Lá

Oc é o tipo que representa a oitava da nota, por exemplo, *Oc 5*, Quinta oitava da nota.

Dr é o tipo que representa a duração da nota. Este número deve ser um real já que existem durações "quebradas".

Nr é o tipo que representa o compasso que está sendo tratado no momento.

Sr é um tipo geral usado quando nenhum dos outros tipos se enquadra no problema.

M [(String, String, Int,Real)] é o tipo que representa uma linha melódica ele é uma lista de *Nts*.

C [Notation] é um tipo peculiar e poderoso já que ele é definido como sendo uma lista que conterá qualquer combinação com qualquer quantidade dos tipos *Notation*. Este tipo representa um compasso.

ME [String] é o tipo que representa qualquer evento que aconteça antes de uma nota.

$ = toChar;

4.1.2) *finalPoint*:

```
finalPoint:: Result -> Result;
finalPoint (OK(,LT5 ".":rest])) = OK(Nil,rest);
finalPoint xs = Fail,
```

Esta função reconhece o ponto final na lista de *tokens*.

4.1.3) *OpenParen*:

```
openParen (OK(_, [Ts "(":rest])) = OK(Nil, rest);
openParen xs= Fail;
```

A função ***openParen*** simplesmente retira o parêntese de abertura da lista de *tokens* e retorna um resto sem o *token* "(".

4.1.4) *closeParen*:

A função *closeParen* simplesmente retira o parêntese da lista de *tokens* ao retornar um resto sem o *token*: ")". Ao finalizar um compasso só restara um ";".

```
tkn :==[(Ts ")"),(Ts ";") ];

Start = closeParen (OK(Nil, tkn))

A saída é:

(Ok(Nil, [(Ts "."))]))

closeParen (OK(_ [Ts ")":rest])) = OK(Nil, rest);
closeParen xs= Fail,
```

4.1.5) *openBra*:

A função *openBra* reconhece um abre colchete na lista de *tokens*.

```
openBra :: Result -> Result;
openlBra (OK(_ [Ts "[":rest])) = OK(Nil, rest);
openBra xs= Fail;
```

4.1.6) *closeBra*:

Esta função reconhece um fecha colchete na lista de *tokens*.

```
closeBra :: Result -> Result;
doseBra (OK(x, [Ts "]" :rest])) = OK(x,rest);
closeBra xs= Fail;
```

4.1.7) *Comma*:

A função *comma* simplesmente retira a vírgula da lista de *tokens* ao retomar um resto sem o *token* ",".

4.1.8) *semicolon*:

Esta função retira o ponto e vírgula que delimita o final de um compasso. No exemplo abaixo, tem-se o ponto e vírgula, restante do compasso anterior, e o próximo compasso simbolizado Por (Ti 2),(Ts "-"),....

Exemplo:

```
tkn :== [(ts ". "),(Ti 2),(Ts "-"),(Ts "("),(Ti "C"),(Ts "#"),(Ts 5),
(Ts "("),(Ts "sm"),(Ts "d"),(Ts ")"),(Ts ", "),
  (Is "B"),(Ti 5), (Ts "("),(Ts "sc"),(Ts ")"),(Ts ")"),(Ts ";")
  ];

Start = semiColon (OK(Nil, tkn));
```

A saída é:

```
(OK (NiL,[(Ti 2),(Ts "-"),(Ts "("),(Ts "C'),(Ts "#"),(Ti 5),
    (Ts "("),(Ts "sm"), (Ts "d"),(Ts "d'),(Ts ")"),(Ts ", "),
    (Ts "B"),(Ti 5), (Ts "("),(Ts "sc"),(Ts ")"),(Ts ")"),(Ts ", ")
       ]))

semiColon (OK(_,[Ts ", ":rest])) = Ok(Nil,rest);
```

semiColon = Fail;

A partir daqui se verá funções para montagem da estrutura musical num compasso.

4.1.9) *MidiEvent*:

Este trecho de código é a implementação das linhas de gramática:

ALFAE-> (0-9|A-F)$^+$ (0-9|A-F)$^+$

MALSALFAE->" ". ALFAE | ε

EVENTOSUB-> MAISALFAE . EVENTOSUB E

Um dos eventos incluídos aqui é a mudança de canal.

A ordem das funções abaixo é rigorosamente esta. Ela reconhece quatro tipos de *midi-eventos*. Os *midi-eventos* têm tamanhos fixos.

```
MidiEvent (OK(_, [Ti a0, Ts "FF", Ti al, Ti a2, Ti a3, Ti a4,Ti aS,Ti a6: rest]))
        = OK(Sr ({$ a0, $ 0xFF, $ al, $ a2, $ a3,$ a4,$ a5,$ a6 } ), rest);
MidiEvent (OK(_, [Ti b0, Ts "FF', Ti bl, Ti b2, Ti b3, Ti b4: rest]))
        = OK(Sr ({$ b0, $ 0xFF, $ bl, $ b2, $ b3, $ b4}), rest),
MidiEvent (OK(_, [Ti b0, Ts "FF", Ti 2, Ts "F", Ti b: rest]))
        = OK(Sr ({$ b0,$ 0xFF,$ 0x2F,$ b} ), rest);
MidiEvent (OK(_, [Ti b0, Ts "C", Ti c: rest]))
        = OK(Sr ({$ b0,$ 0xC,$ c} ), rest);
MidiEvent_ = abort (" Erro na rotina MidiEvent ");
```

A função ***MidiEvent*** devolve o *midi-evento* como uma *string* para posteriormente ser inserido no trecho musical.

Exemplo de emprego:

```
tkn :== [(Ts "ME"),(Ts "("),(Ti 0),(Ts "FF"),(Ti 58),
        (Ti 4),(Ti 2),(Ti 2),(Ts ")"),(Ts "'),(Ts "ME"),
        (Ts "("),(Ti 0),(Ts "FF"),(Ti 2),(Ts "F"),(Ti 0),
        (Ts ")"),(Ts ","),(Ts "B"),(Ts "5"),(Ti 5),(Ts "("),(Ts "m"),
        (Ts ")"),(Ts ","),(Ts "C"),(Ts "#"),(Ti 5),(Ts "("),
```

```
        (Ts "sm"),(Ts ". "),(Ts ". "),(Ts ")"),(Ts ", "),
        (Ts "B"),(Ti 5),(Ts "("),(Ts "sc"),(Ts ")"),
         (Ts ")"),(Ts ";")
       ];

Start = MidiEvent (OK(Nil, tkn));
```

A saída é:

```
(0K ((Sr " ÿ±23 '"),[(Ts ")"),(Ts ", "),(Ts "ME"),(Ts "("),(Ti 0),
          (Ts "FF"),(Ti 2),(Ts "F"),(Ti 0),(Ts ")"),
          (Ts ", "),(Ts "B"),(Ti 5),(Ts "("),(Ts "m"),
          (Ts ")"),(Ts ". "),(Ts "C"),(Ts "#"),(Ti 5),
          (Ts "("),(Ts 'sm"),(Ts "d),(Ts "d"),(Ts ")"),
          (Is ", "),(Ts "B"),(Ti 5),(Ts "("),(Ts "sc"),
         (Ts ")"),(Ts ")"),(Ts ";")
         ]))
```

```
MidiEvent :: Result -> Result;
MidiEvent (OK (_,[Ti a0, Ts "FF", Ti al, Ti a2, Ti a3, Ti a4,Ti a5,Ti a6: rest]))
      = OK(Sr ({$ a0, $ 0xFF, $ al, $ a2, $ a3,$ a4,$ a5,$ a6 } ), rest);
MidiEvent (OK (_,[Ti b0, Ts "FF", Ti b1, Ti b2, Ti b3, Ti b4: rest]))
      = OK(Sr ({$ b0, $ 0xFF, $ bl, $ b2, $ b3, $ b4}), rest);
MidiEvent (OK (_,[Ti b0, Ts "FF", Ti 2, Ts "F", Ti b: rest]))
      = OK(Sr ({$ b0, $ 0xFF,$ 0x2F,$ b} ), rest);
MidiEvent (OK (_,[Ti b0, Ts "C", Ti c: rest]))
      = OK(Sr ({$ b0, $ 0xC,$ c} ), rest);
MidiEvent _= abort (" Erro na rotina MidiEvent ");
```

4.1.10) *Octave*:

```
Octave (OK(, [Ti i:rest])) = OK(Oc i, rest);
Octave = abort ("Erro na rotina Octave");
```

Como já se pode notar a lista *tkn* está diminuindo por causa dos *tokens* que estão sendo retirados pelas funções listadas acima. *Octave* é mais uma destas que irá retirar da lista de *tokens* a altura da nota.

```
Tkn :== [(Ti 5),(Ts "("),(Ts "m"),(Ts ")"),(Ts ","),(Ts "C"),(Ts "#"),
        (Ti 5),(Ts "("),(Ts 'sm"),(Ts "d"),(Ts "d"),(Ts ")"),
        (Ts ","),(Ts "B"),(Ti 5),(Ts "("),(Ts "sc"),(Ts ')"),
        (Ts ")"),(Ts ";")
       ];

Start = Octave (OK(Nil, tkn)),
```

A saída é:

```
(0K ((Oc 5),[(Ts "("),(Ts "m"),(Ts ")"),(Ts ","),(Ts "C"),(Ts "#"),
        (Ti 5),(Ts "("),(Ts "sm'),(Ts "d"),(Ts "d"),(Ts '9"),
        (Ts ","),(Ts "B"), (Ti 5),(Ts "("),(Ts "sc"),(Ts ")"),
        (Ts ")"),(Ts ";")
       ]))
```

4.1.11) *Duration*:

As funções *Duration, basRhy, basicRhy, AllRhy e findPointValue* resolvem em conjunto o trecho de gramática: *TEMPO -> (b|sb|m|sm|c|sc|f|sf|q|sq)+ . (". ")* . [0-9]** e *RITMO-> "(" . TEMPO . ")"*

```
Duration (OK(_, [Ts x, Ts ". ", Ts ". ", Ts ". ", Ts ". ", Ts ")":rest]))
        | basRhy x = OK(Dr (findPointValue (AllRhy x 0.0) 4), rest);
Duration (OK(_, [Ts x, Ts ". ", Ts ". ", Ts ". ", Ts ")":rest]))
        | basRhy x = OK(Dr (findPointValue (AllRhy x 0.0) 3), rest);
Duration (OK(_, [Ts x, Ts ". ", Ts ". ", Ts ")":rest]))
        | basRhy x = OK(Dr (findPointValue (AllRhy x 0.0) 2), rest);
Duration (OK(_, [Ts x, Ts ". ", Ts ")" :rest]))
        | basRhy x = OK(Dr (findPointValue (AllRhy x 0.0) 1), rest);
Duration (OK(_, [Ts x, Ts ")":rest]))
        | basRhy x = OK(Dr (findPointValue (AllRhy x 0.0) 0), rest);
Duration (OK(_, [Ts x, Ti n, Ts ")":rest]))
```

```
        | basRhy x = OK(Dr (findPointValue (AllRhy x (toReal n)) 0), rest);
Duration (OK(_, [Ts x, Ts ".", Ti n, Ts ")":rest]))
        | basRhy x = OK(Dr (findPointValue (AllRhy x (toReal n)) 1), rest);
Duration = Fail
```

A função *Duration* acha a duração da nota. A *string "d"* simboliza os pontos (*dots*) da nota e este trecho reconhecerá uma nota com até quatro pontos. A expressão *(AllRhy x n)* reconhece se uma nota faz parte ou não de uma quiáltera e *(findPointValue y rest)* reconhece se a nota é ou não pontuada.

```
tkn :== [(Ts "m"),(Ts ")"),(Ts ","),(Ts "C"),(Ts "#"),
        (Ti 5),(Ts "("),(Ts "sm"),(Ts "p"),(Ts "p"),(Ts ")"),
        (Ts ","),
        (Ts "B"),(Ti 5),(Ts "("),(Ts "sc"),(Is ")"),(Is ")"),(Ts ";")
        ];
```

Repare que o resultado acha o valor real da nota e já elimina os parênteses e a vírgula anterior à próxima nota.

A saída é:

```
(OK ((Dr 2),[(Ts ","),(Ts "C"),(Ts "#"),(Ti 5),(Ts "("),(Ts "sm"),
        (Ts "p"),(Ts "p"),(Ts ")"),(Ts ","),(Ts "B"),(Ti 5),
        (Ts "("),(Ts "sc"),(Ts ")"),(Ts ")"),(Ts ","),
        ]))
```

4.1.12) *basRhy*:

```
basl(hyx = isMember x ["b", "sb", "m", "sm", "c", "sc", "f",
                       "sf", "q", "sq"];
```

Esta função devolve um *booleano* baseado em x, ou seja, se o ritmo x está na lista. Ela reconhece somente ritmos básicos.

Start=basicRhy "se"; *devolve=> True*

4.1.13) *findPointValue*:

```
findPointValue:: Real Int -> Real;
findPointValue s i = return s i 0 0.0;
where
    }
        return s I j ac
                | I < j = ac;
                | otherwise = return (s/2.0) i (j+1) (s+ac);
        }
```

Esta função divide o valor "s", que é do tipo real, i vezes, sendo que, i é a quantidade de pontos na nota.

A função ainda calcula o valor de notas pontuadas. "s+ac" soma sempre metade do valor da duração para cada ponto.

```
Start=findPointValue 1.0 2;
```

Calcula o valor de uma semínima com dois pontos. A resposta é: 1.75

4.1.14) *AllRhy*:

```
AllRhy:: {#Char} Real -> Real;
AllRhy r n
        | n == 0.0 = basicRhy r;
        | otherwise = 2.0*(basicRhy r)/n;
```

Esta função encontra o valor (número real) de qualquer nota. Se n=0 então a nota não é quiáltera, portanto, só a função basicRhy se encarrega da tarefa, contudo, se n <> 0 trata-se de uma quiáltera, portanto, o valor deve ser dividido pela duração referência.

Start=AllRhy "se" 3.0;

devolve: 0.0833333, ou seja, o valor de semicolcheia de uma das três notas da quiáltera.

4.1.15) *basicRhy*:

```
basicRhy:: {#Char} -> Real;
basicRhy br
    | br ==          "b"  =   8.0;
    | br ==          "sb" =   4.0;
    | br ==          "m"  =   2.0;
    | br ==          "sm" =   1.0;
    | br ==          "c"  =   0.5;
    | br ==          "sc" =   0.25;
    | br ==          "f"  =   0.125;
    | br ==          "sf" =   0.0625;
    | br ==          "q"  =   0.03 125;
    | br ==          "sq" =   0.015625;
    | otherwise ==            0.0;
```

Esta função encontra o valor em real do ritmo básico de uma nota. O ritmo básico é definido como aquele que não está pontuado nem faz parte de uma quiáltera. Como a relação entre os ritmos das notas é constante sobre qualquer circunstância colocou-se , no código, o valor real e constante supondo a semínima=1.

Start=basicRhy "sc"; devolve: 0.25

NOTAALTURA -> [A-G]. (#|##|&|&|&|%) . [0-9]+*

4.1.16) *Name*:

```
Name (OK(_, [Ts x, Ts "#", Ts "#":rest]))
        | noteName x = OK(Nm (x+++"##"), rest);
Name (OK(_, [Ts x, Ts "#":rest]))
        | noteName x = OK(Nm (x+++"#"), rest);
Name (OK(_, [Ts x, Ts "&", Ts "&":rest]))
```

```
        | noteName x = OK(Nm (x+++"&&"), rest);
Name (OK(_, [Ts x, Ts "&":rest]))
        | noteName x = OK(Nm (x+++"&"), rest);
Name (OK(_, [Ts x, Ts "%":rest]))
        | noteName x = OK(Nm (x+++"%"), rest);
Name (OK(, [Ts x:rest]))
        | noteName x = OK(Nm x, rest);
Name_ = abort ("Erro na rotina:   Name ");
```

A função *Name* reconhece a junção dos acidentes musicais: sustenidos(#), bemóis(&) etc, se houver. Esta função, basicamente, junta dois *tokens string* Ts num só. A função devolve uma lista com o resto dos *tokens*. Repare que o tipo *Notation* possui todos os tipos listados na lista de *tokens*. Quando é necessário agrupar algum tipo, constrói-se uma função como *Name* para devolver no primeiro elemento da tupla, tal como *OK*, o tipo encontrado e no segundo elemento o resto dos *tokens*. Esta função reconhece &&,&, % (aqui usado como becuadro), #, ##.

```
tkn :== [(Ts "B"),(Ti 5),(Ts "("),(Ts "m"),
        (Ts ")"),Ts ",",(Ts "C"),(Ts "#"),(Ts "#"),(Ti 5),(Ts "("),
        (Ts "sm"),(Ts ",",(Ts ",",Ts ")"),(Ts ",",(Ts "B"),
        (Ti 5),(Ts "("),(Ts "sc"),(Ts ")"),(Ts ",",
        ];

Start = Name (OK(Nil, tkn));
```

A saída é:

```
(0K ((Nm "B"),[(Ti 5),(Ts "("),(Ts "m"),(Ts ")"),(Ts ",",(Ts "C"),
        (Ts "#"),(Ti 5),(Ts "("),(Ts "sm"),(Ts "."),(Ts "."),
        (Ts "."),(Ts ")"),(Ts ",",(Ts "B"),(Ti 5),(Ts ")"),
        (Ts "sc"),(Ts ")"),(Ts ")"),(Ts ";")
        ]))
```

4.1.17) *noteName*:

noteName x = TsMember x ["A", "B", "C", "D", "E", "F", "G", "P"];

Esta função devolve "*true*" se x é algum elemento da lista, ou seja, reconhece o nome da nota.

4.2) O MÓDULO *Sytx0899*

Este módulo usa ferramentas do módulo *lexMidi0899*. A figura 5 mostra a relação das funções no módulo.

T:

T (OK(_,)) = True;
T _ = False;

Simplesmente verifica se o parâmetro de T é do tipo *Result*.

R:

R (OK(_, xs)) = xs;
R Fail= abort "Unknown error!";

Esta função devolve a lista de *tokens* ainda não tratada no segundo elemento da tupla.

F:

F (OK(s, _)) = s;
F Fail= abort "Unknown error!";

Esta função diz o seguinte: Sendo do tipo *Result* capture o primeiro elemento da tupla.

FF:

FF (OK(Nt s, _)= s;
FF Fail= abort "Unknown error!";

Esta função devolve somente a nota, sem o *flag Nt*, do primeiro elemento da tupla *OK*.

MEC:

MEC (OK(Sr s, _)) s;
MEC Fail= abort "Unknown error!";

Esta função devolve somente a *string*, sem o *flag Sr*, do primeiro elemento da tupla *OK*.

É importante notar que: O meta evento acima é "00 FF 58 4 2 2 18 8", todos os números devem estar no formato decimal e não no formato *hexa*, portanto escrevemos: 0 FF 88 4 2 2 24 8. A função *Tokens* está no modulo *Tokens2l*.

Tokens "quebra" a *string lst em tokens e parsit* transforma a seqüência de *tokens* numa lista de notas no formato *([C[Notation]])*.

Se:

```
lst = "VOICE " +++ "1 - (B4 (c), C5 (c), D5 (c), E5 (c), F5 (m));" +++
"2 - (D5 (c 3), E5 (e 3), F5 (c 3), G5 (sm), A5 (m));" +++
"3 - (D5 (c. 3), E5 (sc 3), F5 (c 3), G5 (0 FF 88 4 2 2 24 8. sm));" +++
"FINALVOJCE " +++ " VOICE " +++
"1 - (B&4 (c), C&5 (c), D&5 (c), E&5 (c), F&5 (m)); " +++
"2 - (D&5 (c 3), E&5 (c 3), F&5 (c 3), G&5 (sm), A&S (m));" +++
"3 - (D&5 (c. 3), E&5 (sc 3), F&5 (c 3), G&5 (0 FF 88 4 2 2 24 8. sm));" +++
"FINALVOICE ";
Start = parsit (Tokens lst);
```

Devolve:

```
[
(C [(M[("", "B",4,.5),("", "C",5,.5),("", "D",5,.5),
      ("", "F",5,2)])
```

```
        ])
    (C [(M[("", "B",5,.3333333),("", "E",5,.3333333),("", "F",5,.3333333),
        ("", "G",5,5),("", "A"5,2)])
        ])
    (C [(M
    [("", "D",5,.5),("", "E",5,.1666667),("", "F",5,.3333333),("", "F",5,.3333333),
        (" ÿX          ", "])
        ])
    (C [(M[("", "Db",4,.5),("", "Cb",5,5),("", "Db",5,.5),("", "Eb",5,.5),
        ("", "Fb",5,2)])
        ]),
    (C [(M[("", "Db",5,.3333333),("", "Eb",5,.3333333),("", "Fb",5, 3333333),
        ("", "Gb",5,1),("", "Ab",5,2)])
        ])
    (C [(M[("", "Db",5,.5),("", "Eb",5,.1666667),("", "Fb",5,.3333333),
        (" ÿX          ", "Gb",5,1)])
        ])
]
```

A primeira posição da tupla na lista de notas é para Midi eventos inclusive o de mudança de canal. Como o resultado de *parsit* é uma lista contendo listas, onde cada uma traz as nota de determinada voz, pode-se colocar nesta estrutura infinitas listas descrevendo infinitas vozes musicais.

4.2.1) *parsit*:

Esta função faz a chamada a *Score*.

```
tkn     [(Ti l), (Ts "-"), (Ts "("),(Ts "["),(Ts "ME"),(Ts "("),
                (Ti 0),(Ts "FF"),(Ti 58),(Ti 4),(Ti 2),
                (Ti 2),(Ts ")"),(Ts ","),(Ts "ME"),
                (Ts "("),(Ti 0),(Ts "FF"),(Ti 2),
                (Ts "F"),(Ti 0),(Ts ")"),(Ts "]"),
                (Ts "B"),(Ti 5),(Ts "("),(Ts "m"),
                (Ts ")"),(Ts ","),(Ts "C"),(Ts "#"),(Ti 5),
```

```
                (Ts "(",(Ts "sm"),(Ts ". "),(Ts ". "),
                (Ts ")"),(Ts ", "),(Ts "["),(Ts "ME"),
                (Ts "("),(Ti 0),(Ts "FF"),(Ti 58),(Ti 4),
                (Ti 2),(Ti 2),(Ts ")"),(Ts "]"),
                (Ts "B"),(Ti 5),(Ts "("),(Ts "sc"),
                (Ts ")"),(Ts ")"),(Ts ";"),
    (Ti 2),(Ts "-"),(Ts "("),(Ts "["),(Ts "ME"),(Ts ")"),
                (Ti 0),(Ts "FF"),(Ti 58),(Ti 4),(Ti 2),
                (Ti 2),(Ts ")"),(Ts ", "),(Ts "ME"),
                (Ts "("),(Ti 0),(Ts "FF"),(Ti 2),
                (Ts "F"),(Ti 0),(Ts ")"),(Ts "]"),
                (Ts "B"),(Ti 5),(Ts "("),(Ts "m"),
                (Ts ")"),(Ts ", "),(Ts "C"),(Ts "#"),(Ti 5),
                (Ts "("),(Ts "sm"),(Ts ". "),(Ts ". "),
                (Ts "("),(Ti 0),(Ts "FF"),(Ti 58),(Ti 4),
                (Ti 2),(Ti 2),(Ts ")"),(Ts "]"),
                (Ts "B"),(Ti 5),(Ts "("),(Ts "sc"),
                (Ts ")"),(Ts ")"),(Ts ";")
                ];

 Start=parsit tkn;

 Devolve:

[  (C [(M [(" ÿ±23' ÿ/'", "B",5,2),(" ", "C#",5,1.75(,(" ÿ±23' ", "B",5,.25)])
   ]),
 (C [(M [(" ÿ±23' ÿ/'", "B",5,2),(" ", "C#",5,1.75(,(" ÿ±23' ", "B",5,.25)])
   ])
]
```

Repare que na saída se tem uma lista que é composta por tipos C

```
parsit :: [Tk] -> [Notation],
parsit xs = Score xs;
```

4.2.2) *Score*:

A função *Score* faz a varredura da partitura aplicando *RdLine* a cada compasso. A expressão: ***(cl, xs) = getTks(RdLine (OK(NTi, tks)));*** coloca em "cl" um compasso e em "*more*" os outros compassos ainda não tratados.

Se Start = Score tkn

Devolve:

[(C[(M [notas]),(M [notas]),etc]), etc]

```
Score [] = []
Score tks = [cl : more]
where {
      (cl, xs) = getlks(RdLine (OK(Nil, tks)));
      more= Score xs
      };
```

As *funções RdLine, getlks, compass, melody e addCompass* trabalham sob o trecho de gramática: *COMPASSO -> INTEIRO. "-". "(" . NOTAS . ")"*

4.2.3) *getTks*:

Esta função transforma o tipo definido pelo usuário OK numa tupla simples.

```
getlks (OK(tl, xs)) = (t1, xs);
getlks Fail= (Nil, []);
```

4.2.4) *RdLine*:

Esta função retira da lista de *tokens* o identificador de compasso.

Se:

```
tkn :== [(Ti 2),(Ts "-"),(Ts "("),(Ts "D"),(Ti 5),(Ts "("),
                    (Ts "c"),(Ti 3),(Ts ")"),(Ts ","),
                        (Ts "E"),(Ti 5),....];
E:
lst = (OK(Nil,tkn)); Start=RdLine lst;
Start=Rdline lst;

Devolve:

[(Ts "("),(Ts "D"),(Ti 5),(Ts "("),(Ts "c"),(Ti 3),(Ts ")"),(Ts "."),
(Ts "E"),(Ti 5),....]

RdLine (OK(x, [Ti i, Ts "-": rest])) = compass (OK(x, rest));
RdLine ok = compass ok;
RdLine Fail = ErrorReport " RdLine " "    "3;
```

4.2.5) *compass*:

A função separa os compassos da música trazendo já estruturados todos os elementos que os compõe tais como, *meta-evento*, melodia e ritmo numa tupla. O Separador escolhido para a distinção de compassos foi ";". A função chamada *melody* separa a linha melódica. Se *restOfCompass* é verdadeiro então *ml* que possui os *tokens* que ainda restam no compasso é passado como parâmetro na chamada recursiva.

```
compass (OK(_, [Ts ";":rest])) = OK( C [], rest);
compass ok
        #               m l = melody ok;
           restOfCompass = compass m l | T restOfCompass = addCompass m
                                                                    l
restOfCompass;
compass _ = Fail;

Se:
tkn
        [(Ti 1),(Ts "-"),(Ts "("),(Ts "ME"),(Ts "("),(Ti 0),(Ts "FF"),
```

```
    (Ti 58),(Ti 4),(Ti 2),(Ti 2),(Ts ")"),(Ts ",",(Ts "ME"),
    (Ts "(",(Ti 0),(Ts "FF"),(Ti 2),(Ts "F"),(Ti 0),(Ts ")"),
  (Ts ",",(Ts "B"),(Ti 5),(Ts "(",(Ts "m"),(Ts ")"),
  (Ts ",",(Ts "C"),(Ts "#"),(Ti 5),(Ts "(",(Ts "sm"),
  (Ts "d"),(Ts "d"),(Ts ")"),(Ts ",",(Ts "B"),(Ti 5),(Ts "(",
  (Ts "sc"),(Ts ")"),(Ts ")"),(Ts ";"),
  ];

  Start = Compass (OK(Nil,tkn));
```

A saída é:

```
(OK ((Nr 1),[(Ts "(",(Ts "ME"),(Ts "(",(Ti 0),(Ts "FF"),(Ti 58),
      (Ti 4),(Ti 2),(Ti 2),(Ts ")"),(Ts ",",(Ts "ME"),
      (Ts "(",(Ti 0),(Ts "FF"),(Ti 2),(Ts "F"),(Ti 0),
      (Ts ")"),(Ts ",",(Ts "B"),(Ti 5),(Ts "(",(Ts "m"),
      (Ts ")"),(Ts ",",(Ts "C"),(Ts "#"),(Ti 5),(Ts "(",
      (Ts "sm"),(Ts "d"),(Ts "d"),(Ts ")"),(Ts ",",(Ts "B"),
      (Ti 5),(Ts "Q"),(Ts "sc"),(Ts ")"),(Ts ")"),(Ts ";")
      ]))
```

4.2.6) *melody*:

A função *melody* tem como propósito retirar da lista de *tokens* o abre parêntese e reconhecer as notas de um compasso.

```
melody ok
      # paren= openParen ok;
        m=notes paren | T m = m;
melody _ = Fail;
```

4.2.7) *addCompass*:

Esta função tem o mesmo comportamento das *adds* anteriores, ela junta os

elementos *M(...)* numa nova lista *([C [M (...), M (...)]])*, a lista de compassos.

addCompass (OK(ml, _)) (OK(C ns, rest)) = OK(C [ml :ns], rest);
addCompass _ _ = abort "Syntatic error! ";

As funções notes e *addNote* resolvem o seguinte trecho de gramática: *NOTAS-> EVENTOS . NOTA . MAISNOTA.*

4.2.8) *notes*:

O segredo para se entender o código de notes é compreender como funciona o *let (#)* e perceber que cada linha de cada trecho de ***notes*** desempenha papel fundamental na leitura da estrutura de entrada e na produção da estrutura de saída. Veja a lista ***tkn***, deve-se tratar quatro casos básicos:

1°) Quando se está para finalizar o compasso e se encontra a última nota, um fecha parêntese aparece, e só ai, posposto à última nota. Assim no trecho (1) ***lastNote= note ok "";*** pega a última nota, isto é feito pela função ***note.*** O segundo parâmetro desta função é uma *string* vazia porque este trecho resolve o problema de não se ter um *meta-evento* antecedendo a nota e ser a última. A função ***rest = closeParen lastNote | T rest = OK(M 1FF lastNotej, R rest);*** é aquela que serve ao mesmo tempo de controle e para retirar o fecha parêntese, ainda é feita urna verificação se *rest* está no formato *T.* Caso seja afirmativo a lista do tipo *M* é preenchida pelo resultado de *FF* e *R* devolve o restante dos *tokens* ainda não tratados.

2°) Quando existem, para além da posição atual da lista de *tokens*, mais notas, a expressão ***c = comma nl;*** caracteriza a presença destas notas e este trecho (2) só será selecionado por isto, porque há urna coincidência. Ao se retirar elementos da lista de *tokens* a lista vai diminuindo e à medida que os *tokens* vão sendo retirados outros tomam seu lugar, sem aumentar a lista, como dissemos, e foi dito está seqüência de tokens deve ser conhecida do trecho do programa por a ordem que eles aparecem dever ser completamente previsível.

3°) Quando se tratar da nota final da melodia', e ela for antecedida de *meta-eventos* o trecho (3) se encarregará de agrupar os eventos e colocar as notas no formato apropriado.

4°) Finalmente, o trecho (4) está supondo que existe um volume de notas em seqüência

todas com *meta-eventos*.

Combinados os trechos, eles resolvem todas as possibilidades: "nota sem *meta-evento*" seguida de "nota com *meta-evento*", "nota com *meta-evento*" seguida de "nota sem *meta-evento*" etc. As combinações possíveis são àquelas dos números de trechos. Se:

```
tkn:= =[(Ts "["),(Ts "ME"),(Ts "("),(Ti 0),(Ts "FF"),(Ti 58),(Ti 4),
        (Ti 2),(Ti 2),(Ts ")"),(Ts ","),(Ts "ME"),(Ts "("),(Ti 0),
        (Ts "FF"),(Ti 2),(Ts "F"),(Ti 0),(Ts ")"),(Ts "]"),(Ts "B"),
        (Ti 5),(Ts "("),(Ts "m"),(Ts ")"),(Ts ","),(Ts "C"),(Ts "#"),
        (Ti 5),(Ts "("),(Ts "sm"),(Ts "d"),(Ts "d"),(Ts ")"),(Ts ","),
        (Ts "["),(Ts "ME"),(Ts "("),(Ti 0),(Ts "FF"),(Ti 58),(Ti 4),
        (Ti 2),(Ti 2),(Ts ")"),(Ts "]"),(Ts "B"),(Ti 5),(Ts "("),
        (Ts "sc"),(Ts ")"),(Ts ")"),(Ts ";")
        ];

ok=(OK(Nil,tkn));
Start=notes ok;
```

A saída é:

```
(OK ((M [(" ÿ±23' ÿ/!", "B5",2),("", "C#5",1 .75),(" ÿ±23'", "B5",.25)]),
     [(Ts ";")]))
```

A lista com flag *M* (melodia) foi preenchida com as notas da melodia e restou na lista de *tokens* apenas o marcador de fim de compasso ";".

Código:

```
notes Fail = Fail;

notes ok (1)
  # lastNote= note ok "";
      rest = closeParen lastNote | T rest = OK(M [FF lastNote], R rest);
notes ok (2)
```

```
            # nl = note ok
               c =  comma nl,
    restOfNotes = notes c T restOfNotes addNote nl restOfNotes;

    notes ok (3)
    #          b = openBra ok;
             me = MetaLventList b;
      lastNote = note me (joinMEs (F me));
          rest = closeParen lastNote | T rest = OK(M [FF lastNote], R rest);

  notes ok (4)
        #  ob = openBra ok;
           me = MetaEventList ob;
           n1 = note me (joinMEs (F me));
            c = comma nl;
 restOfNotes = c | T restOfNotes = addNote nl restOfNotes notes;

 notes _ Fail;
```

4.2.9) *AddNote*:

AddNote junta duas notas. Como exemplificado abaixo pega-se o primeiro elemento da estrutura "OK".

Ex:

```
Start = addNote (OK(Nt ("B5",l.725), [(Ts "qcoisa l ")]))
            (OK(M [ ("A5",2.0)], [(Ts "qcoisa2")] ))
```

A Saída é:

```
(OK ((M [("B5", 1.725),("A5",2)]),[(Ts "qcoisa")]))

addNote (OK(Nt n1, _)) (OK(M ns, rest)) = OK(M [nl :ns, rest);
addNote _ _ = abort "Syntatic error!";
```

As funções mkNote, notOkME, whoNext e note resolvem a linha de gramática:
NOTA -> NOTAALTURA . RITMO

A função *note* devolve no formato: *OK(Nt(Eventos,NomeNota,Oitava,duracao), rest). Name, Octave, openParen e Duration esta em lexMidi0899*

4.2.10) *note*:

```
note ok
    #  nm = Name ok;
    octave = Octave nm;
        op = openParen octave;
        me = whoNext op;
         d = Duration me | T d = mkNote (me, nm, octave, d);
note _      = abort( " Erro na rotina: note ");
```

4.2.11) *whoNext*:

```
whoNext op
        | okME op MidiEventStr op;
        | otherwise = notOkME op;
```

4.1.12) *mkNote*:

A função ***mkNote*** é a primeira de uma série que começa a montar a lista de notas com suas alturas e ritmos. O que ela faz é simplesmente juntar o nome da nota à sua oitava e ligar ao ritmo.

A chamada é:

```
Start = mkNote ( OK((Nm "B"), [Ts "qcoisal "]), OK((Oc 8),[Ts "qcoisa2"]),
        OK((Dr 1.725), [Ts "qcoisa3"]))
```

A saída é:

```
(OK ((Nt ("B8", 1.725)), [(Ts "qq3")]))

mkNote (OK(Nm nm, _), OK(Oc oc, _), OK(Dr n, rest)) = OK(Nt(nt, n), rest)
where {
        nt = nm+++(toString oc);
      };
```

4.2.13) *okME*:

```
okME (OK(x, [Ti n: rest])) = True;

okME_ = False;
```

4.2.14) *notOkME*:

```
notOkME (OK(x,y))=(OK( Sr "",y)),
```

As funções *MidiEventStr, MidiEventLTst, joinMEs e addME* resolvem a linha de gramática*: EVENTOS-> "[" . EVENTO . MAISEVENTO. "]" ε*

4.1.15) *MidiEventStr*:

Esta função devolve: *OK((Sr "MidiEventosBlocoString"), rest)*

```
MidiEventStr ok = meStr;
where
      {
      meLTst = MidiEventLTst ok,
      meStr joinMlEs meLTst;
      };
```

4.2.16) *joinMEs*:

```
joinMEs (OK(ME lst, rest)) = OK( (Sr (foldr (+++) lst)), rest);
```

Esta função junta os vários *meta-eventos* num só bloco *string*.

Se Tst= ["Cala", "mus"]
Start=joinlMEs (ME lst) = foldr (+++) "" lst,

Devolve: "Calamus"

4.1.17) *MidiEventList*:

Os midi-eventos têm tamanho fixo. A ordem das funções é rigorosamente esta. *Comma, finalPoint e MidiEvent* estão no módulo *lexMidi0899.*

```
MidiEventList ok
#       lastME = MidiEvent ok; (1)
        fi = finalPoint lastME | fp = 0K (ME [MEC lastME], R fp),

MidiEventLTst ok
#       me = MidiEvent ok; (2)
      c = comma me;
  restOfME = MidiEventList c | restOfME = addME me restOfME;

MidiEventLTst_ = abort("Erro na rotina MidiEventLTst ");
```

Esta função reconhece e recupera da lista de *tokens* todos os *midi-eventos* que ocorrem antes de uma nota, caso eles existam.

O trecho de código (2) acima recupera primeiro "*me*" que é do tipo *(OK(Sr ÿ+^{23}'",[(Ts ")"),(Ts ", "):resto_dos_tokens]))*. Já *c* recupera: *(OK(Nil,[(Ts ", "):resto_dos_tokens]))*. O trecho final faz uma chamada recursiva a *MidiEventList* porque este trecho supõe que existe outro *midi-evento* na lista. A expressão *T restOfME* garante que se esteja manipulando o tipo *(OK(...))*. A expressão *addME* irá colocar numa lista cada elemento da lista de *midi-eventos* já devidamente formatado começando por "*me*". O processo todo só pára quando o trecho de código (1) é satisfeito. Ele diz o seguinte: em *lastME* recupera-se algo parecido com "*me*", e em "*fp*", um ponto final diz que não há mais *midi-eventos*. No trecho (2) tem-se uma curiosidade: *restOfME* não foi resolvido ainda e é passado como parâmetro para *addME* que apenas constrói uma lista com os midi-eventos, como isto é possível?

O trecho *"restOfME= MidiEventLTst c"* obtém um novo "*me*" quer a lista possua mais de um evento ou seja o último. Caso o midi-evento seja o último o trecho (1) é executado.

```
Se tkn:= = [(Ts "ME"),(Ts "("),(Ti 0),(Ts "FF"),(Ti 58),
        (Ti 4),(Ti 2),(Ti 2),(Ts ")"),(Ts "."),(Ts "B"),(Ti 5),
        (Ts "("),(Ts "m"),(Ts ")"),(Ts ","),(Ts "C"),(Ts "#"),
        (Ti ),(Ts "("),(Ts "sm"),(Ts "d"),(Ts "d"),(Ts ")"),
        (Ts ")"),(Ts ";"),
       ];

Start = MidiEventList (OK(Nil, tkn));

Devolve:

(OK ((ME "[" y+23' "]),[(Ts "B"),(Ti 5),(Ts "("),(Ts "m"),(Ts ")"),
        (Ts ","),(Ts "C"),(Ts "#"),(Ti 5),(Ts "("),
        (Ts "sm"),(Ts "d"),(Ts "d"),(Ts ")"),(Ts ","),
        (Ts "B"),(Ti 5),(Ts "("),(Ts "sc'),(Ts ")"),
        (Ts ")"),(Ts ";")
       ]))
```

4.2.18) *addME*:

```
addME (OK(Sr nl, _)) (OK(ME ns, rest)) = OK(ME [nl :ns], rest);
        addME_ _ = abort "Syntatic error!";
```

Esta função unifica dois tipos num só os *Srs e MEs* passam a ser somente *ME* que agrupa todos os midi-eventos que antecedem uma nota.

4.3) O MODULO *Midi0899*

4.3.1) *HdTrack*:

```
:: Hdlrack =
```

```
{
timeSignature   :: Int,
formula         :: Int,
ticks           :: Int,
nFusas          :: Int,
keySignature    :: Int,
acidente        :: int, /* sf/*
mode            :: Int, /* mi */
tempo           :: Int,
channel         :: Int,
instrument      :: Int
};
```

No formato 1, como já se disse, são necessários, quase sempre, dois *tracks*: um com a configuração básica e outro com as notas. A declaração de tipo definida pelo usuário, *Hdtrack*, contém o cabeçalho básico desta configuração.

4.3.2)Trecho do programa contendo tuplas:

```
fst2Tupla (x,y) = x;
sec2Tupla (x,y) = Y;
fst4Tupla (x,y,w,z) = x;
sec4lupla (x,y,w,z) = y;
thr4lupla (x,y,w,z) = w;
for4lupla (x,y,w,z) = z;
fif5Tupla (a,b,c,d,e) = e;
red5To4Tupla(a,b,c,d,e) = (a,b,c,d);
```

Estas funções, cujos parâmetros são tuplas, desempenha a tarefa de retirar algum elemento (ou vários) destas tuplas.

4.3.3) *notesMidi*:

Esta função constrói uma lista única (retirando qualquer marcação de compasso) com todas as notas da música. Após, procede-se à verificação das pausas antes das notas e finalmente junta todas as notas já com suas alturas e durações.

```
notesMidi:: [Notation] Int -> {#Char};
```

```
notesMidi nl ppq = return;
where {
        tnl = allNoteslnOneLst nl;
        p = (pauses tnl O "" []ppq);
        ns = map (rhyAndNote ppq) p;
    return = foldr (+++) "" ns;
    };
```

4.3.4) *allNoteslnOneLst*:

Esta função irá colocar numa lista apenas, todas as notas de uma música. A lista já não tem aqui os *flags* de tipo.

```
allNoteslnOneLst:: [Notation] -> [(String,String,Int,Real)];
allNoteslnOneLst allC = flatten (prepareNotes allC);
```

4.3.5) *prepareNotes*:

Esta função mapeia *oneCompass* em uma lista do tipo *Notation* o que irá produzir uma lista sem tipo.

```
prepareNotes:: [Notation] -> [[(String,String,Int,Real)]];
prepareNotes allC = map oneCompass allC;
```

4.3.6) *oneCompass*:

```
oneCompass (C [(N lst)]) = lst;
```

4.3.7) *pauses*:

Esta função detecta e resolve os problemas das pausas antes das notas. Para o *MIDI* as pausas são caracterizadas unicamente por silêncios e não têm qualquer cerceamento rítmico, assim o que a função faz é reunir todas as pausas, quando há mais de uma, num único bloco.

```
pauses nl ad me ac2 ppq
        | length nl == 0 = ac2;
        | (onePBool (hd(nl))) = pauses (tl(nl)) (ac 1 +(PpqNote (hd(nl)) ppq))
                        (me+++(fst4Tupla (hd(nl)))) ac2 ppq;
        | otherwTse = pauses (tl(nl)) O me  (ac2 ++ [(me+++(fst4lupla (hd(nl))),
                                                sec4lupla (hd(nl)),
                                                thr4lupla (hd(nl)),
                                                for4Tupla (hd(nl)),
                                                adl)]) ppq,
```

4.3.8) *rhyAndNote*:

Esta função constrói a string que representa uma nota no formato *MIDI* (uma das possibilidades). O modelo é o seguinte: *meta-evento* (se existir) + valor da(s) pausa(s) da nota em *hexa* + $0x90 + nota em *hexa* + volume da nota (que aqui é máximo) + duração da nota em *hexa* + volume da nota em questão com valor 0 (nota desligada).

```
rhyAndNote:: Int (String,String,Int,Real,Int) -> String;
rhyAndNote ppq tuplaNote = return;
where {
  (me,note, hi, nd,bfn) = tuplaNote;
                  np= if (bfn >0)
                          { toChar((hi7f bfn) + 128), toChar(lo7f bfn) }
                  r = red5To4Tupla(tuplaNote),
                  ndi = PpqNote r ppq;
              ndppq = {toChar((hi7f ndi) + 128), toChar(lo7f ndi)},
                  n = (NotaMidi note hi);
    return = me+++np +++ { $0x90,n, $0x7F } +++ ndppq +++ { $0x80,n, $0 }
    }
```

4.3.9) *NotaMidi*:

Na função *NotaMidi* estão listadas todas as notas da notação musical ocidental. Ela devolve o correspondente ao valor *midi* da nota em inteiro. Este valor inteiro é possível porque a relação entre as notas é constante. A relação entre as notas também pressupõe o que se conhece como enarmônicos.

Ex:

Start = NotaMidi "A##" 6

Devolve: 53

```
NotaMidi:: String Int -> Char; NotaMidi nota altura
        | (isMember nota [ "B#", "C", "Dbb"]) = $( 12* altura);
        | (isMember nota [ "B##","C#", "Db" ]) = $((12* altura)+1);
        | (isMember nota [ "CH#","D", "Ebb"]) = $((12* altura)+2);
        | (isMember nota [ "D#", "Eb","Fbb"]) = $((12* altura)+3);
        | (isMember nota [ "D##","E", "Fb"]) = $((12* altura)+4);
        | (isMember nota [ "E#", "F", "Gbb"]) = $((12* altura)+5);
        | (isMember nota [ "E##","F#", "Gb"]) = $((12* altura)+6);
        | (isMember nota [ "F##","G", "Abb"]) = $((12* altura)+7);
        | (isMember nota [ "G#", "Ab"])  = $((12* altura)+8);
        | (isMember nota [ "G##","A", "Bbb"]) = $((12* altura)+9);
        | (isMember nota [ "A#", "Cbb", "Bb" ]) = $((12* altura)+ 10);
        | (isMember nota [ "A##","B", "Cb"]) = $((12* altura)+1 1);
        | otherwise =
```

4.3.10) *hdrChk*:

Esta função tem como parâmetros de entrada *"format notracks e dívision"*, que são inteiros. A função devolve uma string. Esta função é que é verdadeiramente o cabeçalho do arquivo *midi*. Concatenou-se as variáveis, com valores constantes. A expressão *"{'M', 'T', 'h','d'}"* +++ está dizendo para concatenar o vetor de caracteres *{'M', 'T', 'h', 'd'}* com o que vem a seguir: *"+++"*. O que vem a seguir é o tamanho, em *hexa*, de "*format*", "*notracks*" e "*division*" que ocupam 6 *bytes*. No *midi* o tamanho é pré-fixado.

```
hdrChk:: Int Int Int -> String;
hdrChk format notracks divTsion =

        {'M', 'T', 'h', 'd'} +++ //concatena string
        {$0, $0, $0, $6} +++
        { $0, $format } +++
```

{hi notracks, lo notracks} +++

{hi division, lo division}; // ppq

4.3.11) *TrkChunk*:

Esta função tem como parâmetro de entrada *Hdtrack*, que já é em si mesmo uma coleção de tipos. A função devolve uma *string*. O vetor *{'M', 'T', 'r', 'k'}* é o *flag* que determina onde começa um track. *{$0, $0, $0, $0x19}* é o tamanho do *track* a partir deste ponto. *(three 0 t.timeSignature)* usa a função *three* que está declarada na cláusula *where*. O que esta função faz é simplesmente formatar um número inteiro, transformando-o em *hexa*, e devolvendo-o no formato de caractere (1 *byte*). Este número se apresenta como sendo chamado de dentro de um *record* pelo ponto *"t. ..."*. Os *"ticks"* são a unidade de tempo do metrônomo. *"t.nfusas"* diz que se tem *n* fusas por unidade de tempo. *"t.keySignature"* informa que, a seguir, virá a armadura de clave e por isso tem-se *"{$2, $t.accidents, $t.mode) +++"*. *"$2"* quer dizer que a seguir virão dois bytes. O primeiro byte é *"$t.accidents"* que define a armadura de clave. O segundo byte é *"$t.mode"* que define o modo maior ou menor da música. *"{ $0, $0xFF, $0x5 1 } +++"* quer dizer que o primeiro caractere do vetor é um *delta-time*, o segundo e terceiro informam que, a seguir, será definido o tempo (comando de *"set tempo"*). *"{ $3, Lo t.tempo, hi t.tempo, lo t.tempo} +++"* aqui se vê que *"$3"* declara que a seguir virão três bytes, o resto do vetor diz que se deve formatar o tempo segundo três bytes de caracteres. *"{$0, $0xFF, $0x2F, $0)"* finaliza o bloco de *track.*. *"nt"* é puramente as notas já escritas como *strings*. *"t"* é um *record* com dados do cabeçalho do *track*.

```
trkChunk :: Hdlrack String -> String;
trkChunk t nt = th+++lentrk+++resttrack+++n
where {
      th={'M', 't', 'r', 'k'};
        lt = (size restlrack) + (size n);
        resttrack=(three 0 t.timeSignature) +++
              (three 4 t.formula) +++
              {$ t.ticks, $t.nFusas} +++
              (three 0 t.keySignature) +++
              {$2, $t.accidents, $t.mode} +++
```

```
        $0, $0xFF, $0x5 1 } +++
        {$3, Lo t.tempo, hi t.tempo, lo t.tempo} +++
        ($0, $t.channel,$t.instrument };
    n = nt +++ ($0, $0xFF,$0x2F, $0);
    lentrk = {Hi lt, Lo lt, hi lt, lo lt};

    three i s = {$i, hi s, lo s)
    };
```

4.4) O MÓDULO *gerll99*

Este módulo é o que contém a função *gerMidi*. O trabalho desta função é transformar um arquivo que contém uma música no formato texto num arquivo *MIDI*. As constantes abaixo impõem que o *ppq* deve ser 400 em *hexa*, que se usará o formato 1 e dois *tracks*. A constante *chk* é um *Record* contendo o *setup* inicial de uma música. Os campos do *Record* poderão ser alterados e serão, como se verá, quando for apresentado o editor musical.

```
    ppq :== 0x400;
  format :== 1;
  nolracks :== 2;
chk = {
      timeSignature = 0xFF58,
      formula = 0x0402,
      ticks = 0x18,
      nFusas = 0x8,
      keySignature = 0xrFF59,
      accidents = 0,
      mode = 0,
      tempo = 0x98968,
      channel = 0xC0,
      Instrument = 0
      }
```

4.4.1) *gerMidi*:

Supondo que *ln* é uma lista com um número finito vozes, pode-se sofisticar a duas funções abaixo: *v1* e *v2* e torná-las capazes de escrever uma grade de orquestra. As funções *v1* e *v2* recebem as listas contendo as notas das vozes 1 e 2 respectivamente. Estas vozes inicialmente no formato texto são transformadas em *voice1* e *voice2* que são o resultado da transformação das vozes no seu equivalente *MIDI*. Finalmente o resultado de *gerMidi* conterá o texto *MIDI* a ser inserido num arquivo.

```
gerMidi:: String -> String;
gerMidi score
# notesLst = (parsit(Iokens score)); // devolve uma lista do tipo [C [N (..)]...]
    v1 = (hd(notesLst));
    v2 = (hd(tl(notesLst)));
    voice1 = notesMidi vi ppq;
  voice2 = notesMidi v2 ppq;
= (hdrChk format nolracks ppq)+++
(IrkChunk chk voice1) +++ (IrkChunk chk voice2);
```

O fato é ilustrado com o exemplo a seguir:

```
score =
"VOICE " +++
"1 - (B4 (c), C5 (c), D5 (c), E5 (c), F5 (m));" +++
"2- (D5 (c 3), E5 (c 3), F5 (c 3), G5 (sm), A5 (m));" +++
"3- (D5 (c. 3), E5 (sc 3), F5 (c 3), G5 (0 FF 88 4 2 2 24 8. sm));" +++
" FINALVOICE " +++
" VOICE "
"1 - (B&4 (c), C&5 (c), D&5 (c), E&5 (c), F&5 (m));" +++
"2 - ( D&5 (c 3), E&5 (c 3), F&5 (c 3), G&5 (sm), A&5 (m));" +++
"3 - ( D&5 (c. 3) , E&5 (sc 3), F&5 (c 3), G&5 (0 FF 88 4 2 2 24 8. sm));" +++
"FINALVOICE ";
```

```
Start= gerMidi score;
```

Irá devolver uma longa string de caracteres no formato MIDI.

CAPÍTULO V

IMPLEMENTAÇÃO DE UM EDITOR MUSICAL SIMPLES

Este capítulo irá tratar do editor musical feito para intermediar a entrada musical e a produção de um arquivo *MIDI*. Para a aproximação mais precisa da forma musical é que existe o editor. Ele é a ligação com o mundo musical e o texto produzido é a estrutura intermediária.

5.1) Módulo *ConvertToMidi*:

Este é um dos módulos de apoio, usado para transformar uma lista produzida pelo editor de partitura e contendo uma estrutura própria para interpretação de notas em *MIDI*.

5.1.1) *ToStrManyVoices*:

O parâmetro *nOfc* é o número de compassos na voz. Repare que o objetivo desta função é transformar *listaNt* num texto (o caminho inverso do conversor apresentado no capitulo IV).

listaNt = [(Br (20,20),A(5,1,l)), (Co (21,21),B (3,1,2)), (Co (10,10), C (3,1,2)),
(SM (15,1 5),D(5, 1,2)), (Mi (20,20),A(5, 1,2)),
(Br (20,20),A(5,2,1)), (Co (21,21),B (3,2,2)), (Co (10,10), C (3,2,2)),
(SM (l5,15),D(5,2,2)), (Mi (20,2O),A(5,2,2))];
A estrutura exigida aqui é: (Duração (x,y), nota (altura, voz, compasso)). Se Start = ToStrManyVoices listaNt 2; Devolve:

"VOICE 1-(A5 (b)),2-(B3 (c),C3 (c),D5 (sm),A5 (m)); FINALVOICE
VOICE 1-(A5 (b));2-(B3 (c),C3 (c),D5 (sm),A5 (m)); FINALVOICE"

ToStrManyVoices:: [(Ritmos,Notas)] Int -> { #Char};

```
ToStrManyVoices listaNt nOfc
# svoice = SeparateVoice lTstaNt [] [];
  ScoreStr svoice nOfc "";
```

5.1.2) *SeparateVoice*:

Esta função vai separar as notas que pertencem ao pentagrama 1 das do pentagrama 2. Aqui se está supondo haver apenas dois pentagramas (normalmente piano), e é claro, pode existir, teoricamente, um número ilimitado de vozes por pentagrama.

Se listaNt = [(Br (20,20),A(5,1,1)), (Co (21,21),B (3,1,2)), (Co (10,10), C (3,1,2)), (SM (15,15),D(5,1,2)), (Nu (20,20),A(5,l,2)), (Br (20,20),A(5,2,1)), (Co (21,21),B (3,2,2)), (Co (10,10), C (3,2,2)), (SM (15,1 5),D(5,2,2)), (Mi (20,2O),A(5,2,2))]; e Start = SeparateVoice lTstaNt [] []' Devolve:

```
[
[ ((Br (20,20)),(A (5,1,1 ))),((Co (21,21 )),(B (3,1,2))),
   ((Co (10, 10)),(C (3,1 ,2))),((SM (15,1 5)),(D (5,1,2))),
   ((MI (20,20)),(A (5,1,2)))
 ],
[ ((Br (20,20)),(A (5,2,1))),
   ((Co (21,2 1)),(B (3,2,2))),((Co (10,1 0)),(C (3,2,2))),
   ((SM (15,15)),(D (5,2,2))),((Mi (2O,20)),(A (5,2,2)))
 ]
]
```

```
SeparateVoice :: [(Ritmos,Notas)] [(Ritmos,Notas)] [(Ritmos,Notas)] ->
[[(Ritmos,Notas)]];
SeparateVoice lista0r l1 l2
        | length(lTstaOr) == 0 = [l1l]++[l2];
        | (whatVoice (hd(listaOr)) ) == 1 = SeparateVoice (tl(lTstaOr)) (Append l1
[(hd(listaOr))]) l2;
        | (whatVoice (hd(listaOr)) ) == 2 = SeparateVoice (tl(listaOr)) 11 (Append l2
[(hd(listaOr))]);
```

5.1.3) *Append*:

Esta função apenas junta duas listas.

```
Append [] s = s;
Append [x:r] s = [x: Append r s];
```

5.1.4) *ScoreStr*:

A saída de *SeparateVoice* é a entrada desta função. Quando a função *ToStrManyVoices* pega uma lista com as notas e seus ritmos ela divide sua tarefa entre outras duas funções: *SeparateVoice* e *ScoreStr*. A primeira vasculha, a definição da nota na lista, e descobre a que voz ela pertence. A segunda, supondo já se ter empregado *SeparateVoice*, pega cada um dos elementos da lista transformada (cada elemento é uma lista contendo somente uma voz) e faz as devidas transformações para string.

```
Se listaNt =[
[ ((Br (20,20)),(A (5,1,1))),((Co (21,21 )),(B (3,1,2))),
   ((Co (10, 10)),(C (3, 1,2))),((SM (15,1 5)),(D (5,1,2))),
   ((Mi (20,20)),(A (5,1,2)))
 ],
[ ((Br (20,20)),(A (5,2, 1))),
   ((Co (21,21 )),(B (3,2,2))),((Co (10,1 0)),(C (3,2,2))),
   ((SM (15, 15)),(D (5,2,2))),((lVli (20,20)),(A (5,2,2)))
 ]
  ]; e Start=ScoreStr listaNt 2 ""; Devolve:

"VOICE 1 -(A5 (b));2-(B3 (c),C3 (c),D5 (sm),A5 (m)); FINALVOICE
VOICE 1 -(A5 (b));2-(B3 (c),C3 (c),D5 (sm),A5 (m)); FINALVOICE"

ScoreStr orList nOfc strc
        | length(orList) == 0 = strc;
        | otherwise = ScoreStr (tl(orList)) nOfc (strc+++(ToStrOneVoice (hd(orLTst))
nOfc));
```

5.1.5) *ToStrOneVoice*:

A lista *Lst* descreve toda partitura com suas *n* vozes. Isto se deve ao fato do *MIDI* não considerar a divisão por compasso, mas o preenchimento da unidade de

compasso por notas. Esta função devolve a tradução em *string* de uma voz da partitura escrita na lista.

Se listaNt = [(Br (20,20),A(5,1,1)),(Co (21,21),B (3,1,2)), (Co (10,10), C (3,1,2)), (SM (15,15),D(5,1,2)), (Mi (20,20),A(5,1,2))]; e Start = ToStrOneVoice listaNt 2; Devolve: "VOICE 1-(A5 (b));2-(B3 (c),C3 (c),D5 (sm),A5 (m)); FINALVOICE"

5.1.6) *SeparateCompass*:

Esta função vai separar as notas que pertencem a cada um dos compassos. Na devolução a lista exterior representa uma voz e as listas interiores, cada compasso.

Se: *listaNt = [(Br (20,20),A(5,1,1)), (Co (21,21),B (3,1,2)), (Co (10,10), C (3,1,2)), (SM (1S,15),D(S,1,2)), (Mi (20,20),A(5,1,2))]; e Start = SeparateCompass listaNt 2 1 []*

Devolve:

```
[
  [((Br (20,20)),(A (5,1,1)))],
  [((Co (21 ,21)),(B (3, 1,2))),((Co (10,10)),(C (3,1,2))),
  ((SM (15,15)),(D (5,1,2))),((Mi (20,20)),(A (5,1,2)))]
]
```

```
SeparateCompass oneVoice nOfc count ac
        | dount == (n0fc+1) = ac;
        | otherwise = SeparateCompass oneVoice nOfc (count+1) (Append ac
[takeCompassN oneVoice count []]);
```

5.1.7) *takeCompassN*:

Esta função separa todas as notas pertencentes ao compasso "n"

Se: listaNt = [(Br (20,20),A(5;1,1)), (Co (21,21),B (3,1,2)), (Co (10,10), C (3,1,2)), (SM (15,15),D(5,1,2)), (Mi (20,20),A(5,1,2))]; e Start = takeCompassN listaNt 2 [] Devolve:

```
[
  ((Co (21,21)),(B (3,1,2))),
  ((Co (10,10)),(C (3,1,2))),
  ((SM (15,15)),(D (5,1,2))),
  ((Mi (20,20)),(A (5,1,2)))
```

```
]
```

Ou seja, todas as notas do compasso 2.

```
takeCompassN orList n ac
        | length(orList) == 0 = ad;
        | (takeCompass (hd(orLTst)) ) == n = takeCompassN (tl(orLTst)) n (Append ac
[(hd(orLTst))]);
        | otherwTse = takeCompassN (tl(orLTst)) n ad;
```

5.1.8) *ToStrAllCompass*:

A lista *Lst* descreve apenas uma voz da partitura. Quem produz *Lst* é *SeparateCompass*.

Se: Lst =[[((Br (20,20)),(A (5,1,1)))], [((Co (21,2 1)),(B (5,1,2))),((Co (10,10)),(C (5,1,2))), ((SM (15,15)),(D (5,1,2))),((N'Ti (20,20)),(A (4,1,2)))]]; e Start = IoStrAllCompass Lst Devolve: " VOICE 1-(A5 (b));2-(B5 (c),C5 (c),D5 (sm),A4 (m)); FINALVOICE "

```
ToStrAllCompass orLTst strc
        | length(orLTst) == 0 = " VOICE " +++ strc +++ "FINALVOICE";
        | otherwTse = ToStrAllCompass (tl(orList)) (strc+++(oneCompass
hd(orList))));
```

5.1.9) *oneCompass*:

O parâmetro *Lst* contém todas as notas do compasso e somente de um. Como o exemplo esclarece é desta função o trabalho de traduzir cada nota com sua duração numa outra estrutura, uma *string*, levando-se em consideração de qual compasso se trata simbolizado pelo trecho de *string* "2- ..." , no exemplo abaixo:

Lst =[((Co (21,21)),(B (5,1,2))),((Co (10,10)),(C (5,1,2))), ((SM (15,15)),(D (5,1,2))),((Mi (20,20)),(A (4,1,2)))]; e Start = oneCompass Lst Devolve: "2-(B5 (c),C5 (c),D5 (sm),A4 (m));"

```
oneCompass orList
#       hdn = (hd(orList));
```

```
    idxIntOfc = takeCompass (hdn);
    idxStrOfc = toString(idxIntOfc) +++ "-("
    notes = oneCompassSub orList "";
= idxStrOfc +++ (notes%(0,size(notes)-2)) +++ ");";
```

5.1.10) *oneCompassSub*:

Esta função faz uma tradução direta de uma estrutura derivada da imagem no caso *Lst* numa outra estrutura, uma *string*, sem levar em consideração de qual compasso se trata.

Se: Lst [((Co (21,21)),(B (5,1,2))),((Co (10,10)),(C (5,1,2))), ((SM (15,15)),(D (5,1,2))),((Mi (20,20)),(A (4,1,2)))]; e Start = oneCompassSub Lst "" Devolve: "B5 (c),C5 (c),D5 (sm),A4 (m),"

```
oneCompassSub oneScoreLst pent
        | length(oneScoreLst) == 0 = pent;
        | otherwTse = oneCompassSub (tl(oneScoreLst)) (pent+++(assemblyNoteStr
(hd(oneScoreLst)))+++", ");
```

5.1.11) *assemblyNoteStr*:

Esta função desempenha a tarefa de obter o trecho de texto que descreve a nota.

```
assemblyNoteStr elem
#       ntStr=NotaStr elem;
        rtStr=RitmoStr elem;
= ntStr +++ rtStr;
```

5.1.12) *NotaStr*:

Esta função constrói a partir de uma estrutura de lista um trecho de *string* que representa a nota com sua altura.

```
NotaStr (_,A (x,y,z)) = "A"+++toString(x);
NotaStr (_,B (x,y,z)) "B"+++toString(x);
NotaStr (_,Cc (x,y,z)) = "C "+++toString(x);
```

NotaStr (_,D (x,y,z)) = "D"+++toString(x);
NotaStr (_,M (x,y,z)) = "E"+++toString(x);
NotaStr (_,F (x,y,z)) "F"+++toString(x);
NotaStr (_,G (x,y,z)) "G"+++toString(x);

NotaStr (_,Asus (x,y,z)) = "A#"+++toString(x);
NotaStr (_,Bsus (x,y,z)) = "B#"+++toString(x);
NotaStr (_,Csus (x,y,z)) "C#"+++toString(x);
NotaStr (_,Dsus (x,y,z)) = "D#"+++toString(x);
NotaStr (_,Msus (x,y,z)) = "E#"+++toString(x);
NotaStr (_,Fsus (x,y,z)) = "F4"+++to String(x);
NotaStr (_,Gsus (x,y,z)) = "G4"+++toString(x);

NotaStr (_,Ab (x,y,z)) = "Ab"+++toString(x);
NotaStr (_,Bb (x,y,z)) = "Bb"+++toString(x);
NotaStr (_,Cb (x,y,z)) = "Cb"+++toString(x);
NotaStr (,Db (x,y,z)) = "Db"+++toString(x);
NotaStr (,Mb (x,y,z)) = "Eb"+++toString(x);
NotaStr (,Fb (x,y,z)) = "Fb"+++toString(x);
NotaStr (,Gb (x,y,z)) = "Gb"+++toString(x);

5.1.13) RitmoStr:

Quando as informações na partitura precisam ser salvas é feita a transformação do símbolo de duração e altura da nota no pentagrama num formato texto. Esta função traduz o ritmo da nota, exemplo, *Br(breve) em (b)* que é parte da estrutura estabelecida no capítulo IV. O mesmo se dá no item 5.1.12 onde aquele conjunto de funções obtém o nome da nota.

RitmoStr (Br _,_) = "(b)";
RitmoStr (SB _,_) = "(sb)";
RitmoStr (Mi _,_) = "(m)"
RitmoStr (SM _,_) = "(sm)";
Ritmo Str (Co _,_) = "(c)"
Ritmo Str (SC _,_) = "(sc)";

```
RitmoStr (Fu _,_) = "(f)";
RitmoStr (SF _,_) = "(sf)";
RitmoStr (PB _,_) = "(b)";
RitmoStr (PSB _,_) = "(sb)";
RitmoStr (PMi _,_) = "(m)";
RitmoStr (PSM _,_) = "(sm)";
RitmoStr (PCo _,_) = "(c)";
RitmoStr (PSC _,_) = "(sc)";
RitmoStr (PFu _,_) = "(f)";
Ritmo Str (PSF _,_) = "(si)";
RitmoStr (BeBr _,_) = "(b)";
RitmoStr (BeSB _,_) = "(sb)";
RitmoStr (BeMi _,_) = "(m)";
RitmoStr (BeSM _,_) = "(sm)";
RitmoStr (BeCo _,_) = "(c)";
RitmoStr (BeSC _,_) = "(sc)";
RitmoStr (BeFu _,_) = "(f)";
RitmoStr (BeSF _,_) = "(sf)";
RitmoStr (SusBr _,_) = "(b)";
RitmoStr (SusSB _,_) = "(sb)";
RitmoStr (SusMi _,_) = "(m)";
RitmoStr (SusSM _,_) = "(sm)";
RitmoStr (SusCo _,_) = "(c)";
RitmoStr (SusSC _,_) = "(sc)";
RitmoStr (SusFu _,_) = "(f)";
RitmoStr (SusSF _,_) = "(sf)";
```

5.2) Módulo *SaveScore*:

Módulo de conversão da estrutura em lista para um trecho *string* que representa as notas com seus ritmos e notas.

5.2.1) *lstStrOfNotes*:

Esta função transforma a estrutura de uma lista em uma *string* de mesma forma.

Se listaNt = [(Br (20,20),A(5,1,1)), (Co (21,21),B (3,1,2)), (Co (10,10), Cc (3,1,2)), (SM (15,1 5),D(5,1,2)),(Mi (20,20),A(5,1,2)), (Br (20,20),A(5,2,1)), (Co (21,21),B (3,2,2)), (Co (10,10), Cc (3,2,2)), (SM (15,15),D(5,2,2)), (Mi (20,20),A(5,2,2))]; e Start = lstStrOfNotes listaNt ""; Devolve: "(Br (20,20),A(5,1,1)),(Co (21,21),B(3, 1,2)),(Co (10,10),Cc(3, 1,2)), (SM (15,1 5),D(5, 1 ,2)),(Mi (20,2O),A(5, 1,2)), (Br (20,20),A(5,2, 1)),(Co(21 ,21),B(3,2,2)),(Co(10, 10),Cc(3,2,2)),(SM (15,15),D(5,2,2)),(Mi(20,20),A(5, 2,2))"

```
lstStrOfNotes :: [(Ritmos,Notas)] {#Char} -> {#Char};
lstStr0fNotes [] str = (str%(0,size(str)-2));
lstStrOfNotes [s:r] str = lstStrOfNotes r (rhyStr (s) +++ NoteStr (s) +++ str );
```

5.2.2) *rhyStr*:

Quando se vai salvar, em arquivo, a partitura optou-se pela estrutura representada em *Lst* do item 5.2.3. As duas funções abaixo apenas gravam a dita estrutura numa *string*.

```
rhyStr((Br (x,y) ,_)= "(Br (" +++ $x +++ "," +++ $y +++ "), ";
rhyStr((SB (x,y) ,_)= "(SB (" +++ $x +++ "," +++ $y +++ "), ";
rhyStr((Mi (x,y) ,_)= "(Mi (" +++ $x +++ "," +++$y+++ "), ";
rhyStr((SM (x,y) ,_)= "(SM (" +++ +++++ "," +++ $y +++ "), ";
rhyStr((Co (x,y) ,_)= "(Co (" +++ $x +++ "," +++ $y +++ "), ";
rhyStr((SC (x,y) ,_)= "(SC (" +++ $x +++ "," +++ $y +++ "), ";
rhyStr((Fu (x,y) ,_)= "(Fu (" +++ $x +++ "," +++ $y +++ "), ";
rhyStr((SF (x,y) ,_)= "(SF (" +++ $x +++ "," +++$y+++"), ";

rhyStr((SF (x,y) ,_)= "(SF (" +++ $x +++ "," +++$y+++"), ";
rhyStr((PB (x,y) ,_)= "(PB (" +++ $x +++ "," +++ $y +++ "), ";
rhyStr((PSB (x,y) ,_)= "(PSB (" +++ $x +++ "," +++ $y +++ "), ";
rhyStr((PMi (x,y) ,_)= "(PMi (" +++ $x +++ "," +++ $y +++ "), ";
rhyStr((PSM (x,y) ,_)= "(PSM (" +++ $x +++ "," +++ $y +++ "), ";
rhyStr((PCo (x,y) ,_)= "(PCo (" +++ +++++ "," +++ $y +++ "), ";
rhyStr((PSC (x,y) ,_)= "(PSC (" +++ $x +++ "," +++ $y +++ "), ";
rhyStr((PFu (x,y) ,_)= "(PFu (" +++ +++++ "," +++ $y +++ "), ";
```

```
rhyStr((PSF (x,y) ,_)= "(PSF (" +++ $x +++ "," +++ $y +++ "), ";

rhyStr((BeBr (x,y) ,_)= "(BeBr (" +++ +++++ "," +++ $y +++ "), ";
rhyStr((BeSB (x,y) ,_)= "(BeSB (" +++ +++++ "," +++ $y +++ "), ";
rhyStr((BeMi (x,y) ,_) "(BeMi (" +++ $x +++ "," +++ $y +++ "), ";
rhyStr((BeSM (x,y) ,_)= "(BeSM (" +++ $x +++ "," +++ $y +++ "), ";
rhyStr((BeCo (x,y) ,_)= "(BeCo (" +++ $x +++ "," +++ $y +++ "), ";
rhyStr((BeSC (x,y) ,_)= "(BeSC (" +++ +++++ "," +++ $y +++ "), ";
rhyStr((BeFu (x,y) ,_)= "(BeFu (" +++ $x +++ "," +++ $y +++ "), ";
rhyStr((BeSF (x,y) ,_)= "(BeSF (" +++ $x +++ "," +++ $y +++ "), ";

rhyStr((SusBr (x,y) ,_)= "(SusBr (" +++ $x +++ "," +++ $y +++ "), ";
rhyStr((SusSB (x,y) ,_)= "(SusSB (" +++ $x +++ "," +++ $y +++ "), ";
rhyStr((SusMi (x,y) ,_)= "(SusMi (" +++ $x +++ "," +++ $y +++ "), ";
rhyStr((SusSM (x,y) ,_)= "(SusSM (" +++ $x +++ "," +++ $y +++ "), ";
rhyStr((SusCo (x,y) ,_)= "(SusCo (" +++ $x +++ "," +++ +++++ "), ";
rhyStr((SusSC (x,y) ,))= "(SusSC (" +++ $x +++ "," +++ $y +++ "), ";
rhyStr((SusFu (x,y) ,_)= "(SusFu (" +++ +++++ "," +++ $y +++ "), ";
rhyStr((SusSF (x,y) ,_)= "(SusSF (" +++ $x +++ "," +++ $y +++ "), ";
rhyStr — = abort "rhyStr falhou!";
```

5.2.3) *NoteStr*:

```
NoteStr (_,A (x,y,z)) = "A("+++ $ x +++", "+++ $ y+++","+++ $ z+++ ")), ";
NoteStr (_,B (x,y,z)) = "B("+++ $ x +++","+++ $y +++","+++ $ z ")), ";
NoteStr (_,Cc (x,y,z)) = "Cc("+++ $ x +++", "+++ $y +++","+++ $ z ")), ";
NoteStr (_,D (x,y,z)) = "D("+++ $ x +++", "+++ $y +++","+++ $ z ")), ";
NoteStr (_,M (x,y,z)) = "M("+++ $ x +++", "+++ $y +++","+++ $ z ")), ";
NoteStr (_,F (x,y,z)) = "F("+++ $ x +++", "+++ $y +++","+++ $ z ")), ";
NoteStr (_,G (x,y,z)) = "G("+++ $ x +++", "+++ $y +++","+++ $ z ")), ";

NoteStr (_,Asus (x,y,z)) ="Asus("+++ $ x +++","+++ $ y +++", "+++ $ z ")), ";
NoteStr (_,Bsus (x,y,z)) = "Bsus("+++ $ x +++","+++ $ y +++", "+++ $ z ")), ";
NoteStr (_,Csus (x,y,z)) = "Csus("+++ $ x +++","+++ $ y +++", "+++ $ z ")), ";
```

NoteStr (_,Dsus (x,y,z)) = "Dsus("+++ $ x +++", "+++ $ y+++", "+++ $ z ")), ";
NoteStr (_,Msus (x,y,z)) = "Msus("+-i-+ $ x +++", "+++ $ y +++", "+++ $ z ")), ";
NoteStr (_,Fsus (x,y,z)) = "Fsus("+++ $ x+++", "+++ $ y +++", "+++ $ z ")), ";
NoteStr (_,Gsus (x,y,z)) = "Gsus("+++ $ x +++", "+++ $ y +++", "+++ $ z ")), ";

NoteStr (_,Ab (x,y,z)) = "Ab("+++ $ x +++", "+++ $ y +++", "+++ $ z ")), ";
NoteStr (_,Bb (x,y,z)) = "Bb("+++ $ x +++", "+++ $ y +++", "+++ $ z ")), ";
NoteStr (_,Cb (x,y,z)) = "Cb("+++ $ x+++", "+++ $ y +++", "+++ $ z ")), ";
NoteStr (_,Db (x,y,z)) = "Db("+++ $ x +++", "+++ $ y +++", "+++ $ z ")), ";
NoteStr (_,Mb (x,y,z)) = "Mb("+++ $ x +++", "+++ $ y +++", "+++ $ z ")), ";
NoteStr (_,Fb (x,y,z)) = "Fb("+++ $ x+++", "+++ $ y +++", "+++ $ z ")), ";
NoteStr (_,Gb (x,y,z)) = "Gb("+++ $ x +++", "+++ $ y +++", "+++ $ z ")), ";

Se: listaNt = [(Br (20,20),A(5,1,1)), (Co (21,21),B (3,1,2)), (Co (10,10), Cc (3,1,2)), (SM (15,15),D(5,1,2)), (Mi (20,20),A(5,1,2)), (Br (20,20),A(5,2,1)), (Co (21,21),B (3,2,2)), (Co (10,10), Cc (3,2,2)), (SM (15,15),D(5,2,2)), (Mi (20,20),A(5,2,2))]; e Start = lstStr0fNotes listaNt ""; Devolve: "(Br (20,20),A(5,1,1)),(Co (21,21),B(3, 1,2)),(Co (10,10),Cc(3,1,2)), (SM (15,1 5),D(5, 1 ,2)),(Mi (2O,20),A(5, 1,2)), (Br (20,20),A(5,2, 1)),(Co (21,21),B(3,2,2)),(Co(10,1 O),Cc(3,2,2)), (SM (15,1 5),D(5,2,2)),(Mi (20,20), A(5,2,2))"

5.3) Módulo *ReadScoreTxt*:

Este módulo tem sua representação gráfica na figura 9.

5.3.1) *readScoreTxt*:

Esta função traduz uma string para a estrutura usada para interpretar notas na partitura.

Se notas = "INICIO "+++"(SM (89,75),M(5,1,1)), (SM (126,70),F(5,1,1))," +++ "(SM (158,65),G(5, 1,1)), (SM (198,60),A(5,1,1))," +++"(SM (278,5 5),B(5, 1,2)), (SM (308,50),Cc(6,1,2))," +++"(SM (345,45),D(6,1,2)), (SM (383,40),M(6, 1,2))," +++ "(SM (87,155),G(3,2, 1)), (SM (116,150),A(3,2, 1))," +++ "(SM (148, 145),B(4,2,1)), (SM (181,140),Cc(4,2,1))," +++ "(SM (276,1 35),D(4,2,2)), (SM (312, 130),M(4,2,2))," +++ "(SM (346,125),F(4,2,2)), (SM (382,12O),G(4,2,2)) " +++

```
"FIM"; e Start = readScorelxt notas; Devolve: [ ((SM (89,75)),(M (5,1,1))),((SM
(126,70)),(F (5,1,1))), ((SM (158,65)),(G (5,1,1))),((SM (198,60)),(A (5,1,1))), ((SM
(278,55)),(B (5, 1,2))),((SM (308,50)),(Cc (6,1,2))), ((SM (345,45)),(D (6,1,2))),((SM
(383,40)),(M (6,1,2))), ((SM (87,155)),(G (3,2,1))),((SM (116,1 50)),(A (3,2,1))), ((SM
(148,145)),(B (4,2, 1))),((SM (181,140)),(Cc (4,2,1))), ((SM (276, 135)),(D
(4,2,2))),((SM (312,130)),(M (4,2,2))), ((SM (346,1 25)),(F (4,2,2))),((SM (382, 1
20)),(G (4,2,2)))
]
readScorelxt :: String -> [(Ritmos,Notas)];
readScorelxt readStr
#       tk = Iokens readStr;
= readlext tk;
```

5.3.2) *readText*:

Esta função faz a primeira filtragem reconhecendo se a *string* é válida, marcada pela presença da *string* "INÍCIO".

```
readlext :: [Tk] -> [(Ritmos,Notas)];
readlext read
#       mi = iniReading (READ (read,(NOT,NO),[]));
(READ (x,y,z)) = readloList ini;
=z;
```

5.3.3) *iniReading*:

Esta função reconhece a palavra que dá início a um bloco de notas que caracteriza uma voz na partitura. A função *fim* faz a mesma coisa com a palavra que designa fim do texto.

```
iniReading (READ ([Ts "INICIO":r],x,y)) = (READ (r,x,y));
iniReading _ = abort("Erro na funcao iniReading");
```

5.3.4) *readToList*:

Esta função é responsável por reconhecer cada bloco que descreve uma nota numa coleção de *tokens*.

```
Se lst = [ (Ts "("),(Ts "SM"),(Ts "("),(Ti 346),(Ts ","),(Ti 125),(Ts ")"),(Ts ","), (Ts
"F"),(Ts "("),(Ti 4),(Ts ","),(Ti 2),(Ts ","),(Ti 2),(Ts ")"),(Ts ")"),(Ts ","), (Ts
"Ç'),(Ts "SM"),(Ts "("),(Ti 382),(Ts ","),(Ti 120),(Ts ")'9,(Ts ","), (Ts "G"),(Ts
"Ç'),(Ti 4),(Ts ","),(Ti 2),(Ts ","),(Ti 2),(Ts ")"),(Ts ")"), (Ts "FIM") ]; e Start =
readToLTst (READ (lst,(NOT,NO),[])); Devolve (READ ([(Ts "("),(Ts "SM"),(Ts
"("),(Ti 382),(Ts ","),(Ti 120),(Ts ")"),(Ts ","), (Ts "G"),(Ts "("),(Ti 4),(Ts ",'9,(Ti
2),(Ts ","),(Ti 2),(Ts ")"),(Ts ")"),(Ts "FTIVL")], ((SM (346,125)),(F (4,2,2))), [((SM
(346, 1 25)),(F (4,2,2))),((SMI (382, 1 20)),(G (4,2,2)))])

readToList read
#       lastRhy = readRhy read;
        lastNote = readNotes lastRhy;
  (READ (x,y,z)) = lastNote;
            rest = fim lastNote IRESI rest = READ (x,y,[y:z]);
readToList read
#           rh = readRhy read;
            rn = readNotes rh;
            c = comma rn;
    restOfNotes = readToList c | TREST restOfNotes = addNote c restOfNotes,
readToList _ = abort ("Erro na função readToList");
```

5.3.5) readRhy:

Esta função reconhece da lista de tokens aqueles que caracterizam a duração de uma nota.

```
readRhy :: Reader -> Reader;
readRhy (READ ([Ts "(",Ts "B",Ts "r",Ts "(",Ti i,Ts ",",Ti j,Ts ")",Ts ',":rs],
xs,ys)) = (READ (rs,(Br (i,j),NO),ys));
readRhy (READ ([Ts "(",Ts "SB",Ts "(",Ti i, Ts ",", Ti j, Ts ")", Ts ",":rs],
      xs,ys)) = (READ (rs,(SB (i,j),NO),ys));
readRhy (READ ([Ts "(",Ts "Nl" Ts "i",Ts "(",Ti i, Ts ",",Ti j,Ts ")',Ts ",":rs],
      xs,ys)) = (READ (rs,(Mi (i,j),NO),ys));
readRhy (READ ([Ts "(",Ts "SM",Ts "Q',Ti i, Ts ",", Ti j, Ts ")", Ts ",":rs],
      xs,ys)) = (READ (rs,(SM (i,j),NO),ys)),
readRhy (READ ([Ts "(",Ts "C",Ts "o",Ts "(",Ti i,Ts ",",Ti j,Ts ")",Ts ")':rs],
      xs,ys)) = (READ (rs,(Co (i,j),NO),ys));
```

```
readRhy (READ (LTs "(",Ts "SC",Ts "(",Ti i, Ts ",", Tij, Ts ")", Ts ", ':rs],
        xs,ys)) = (READ (rs,(SC (i,j),NO),ys));
readRhy (READ ([Ts "(",Ts 'T", Ts "u" ,Ts "(",Ti i, Ts "," ,Ti j,Ts )",Ts ", ":rs],
        xs,ys)) = (READ (rs,(Fu (i,j),NO),ys));
readRhy (READ ([Ts "(',Ts "SF",Ts ~'(",Ti i, Ts ",", Ti j, Ts ")", Ts ",":rs],
        xs,ys)) = (READ (rs,(SF (i,j),NO),ys)),
readRhy (READ ([Ts "(",Ts "PB",Ts "Ç',Ti i, Ts "1', Ti j, Ts "), Ts ", ":rs],
        xs,ys)) = (READ (rs,(PIB (i,j),NO),ys));
readRhy (READ ([Ts "(",Ts "PSB",Ts "Ç',Ti i, Ts "2', Ti j, Ts ")", Ts ", ":rs],
        xs,ys)) (READ (rs,(PSB (i,j),NO),ys));
readRhy (READ ([Ts "(",Ts "PM",Ts "i", Ts "(",Ti i, Ts ",", Tij, Ts ")", Ts , ":rs],
        xs,ys)) = (READ (rs,(PMi (i,j),NO),ys));
readRhy (READ ([Ts "(",Ts "PSM",Ts "(",Ti i, Ts ",", Ti j, Ts ")", Ts ", ":rs],
        xs,ys)) = (READ (rs,(PSM (i,j),NO),ys));
readRhy (READ ([Ts "(",Ts "PC",Ts "o", Ts "(",Ti i, Ts ",", Tij, Ts ")", Ts ", ":rs],
        xs,ys)) (READ (rs,(PCo (i,j),NO),ys));
readR.hy (READ ([Ts "(",Ts "PSC",Ts "(",Ti i, Ts ",", Tij, Ts ")", Ts ", ":rs],
        xs,ys)) (READ (rs,(PSC (i,j),NO),ys));
readRhy (READ ([Ts "(",Ts "PF",Ts "u", Ts "(",Ti i, Ts ",", Tij, Ts ")", Ts ", ":rs],
        xs,ys)) = (READ (rs,(PFu (i,j),NO),ys));
readRhy (READ ([Ts "("Ts "PSF",Ts "(",Ti i, Ts ",", Tij, Ts ")", Ts ", ":rs],
        xs,ys)) (READ (rs,(PSF (i,j),NO),ys));
readRhy (READ ([Ts "(",Ts "B",Ts "e", Ts "B", Ts "r", Ts "(",Ti i, Ts ",", Ti j, Ts
")", Ts ", ":rs],
        xs,ys)) = (READ (rs,(BeBr (i,j),NO),ys));
readlRhy (READ ([Ts "(",Ts "B",Ts "e", Ts "SB", Ts "(",Ti i, Ts ",", Ti j, Ts ")",
Ts '', '' :rs],
        xs,ys)) = (READ (rs,(BeSB (i,j),NO),ys));
readRhy (READ ([Ts "(",Ts "B",Ts "e", Ts "~4", Ts "i", Ts "(",Ti i, Ts ",", Ti j, Ts
")", Ts ", ":rs],
        xs,ys)) = (READ (rs,(BeMi (i,j),NO),ys)),
readRhy (READ ([Ts "(",Ts "B",Ts "e", Ts "SM", Ts "(",Ti i, Ts ",", Ti j, Ts ")",
Ts '', '' :rs],
        xs,ys)) = (READ (rs,(BeSM (i,j),NO),ys));
```

```
readRhy (READ ([Ts "(",Ts "B",Ts "e", Ts "C", Ts "o", Ts "(",Ti i, Ts ",", Ti j, Ts
")", Ts ",",:rs],
        xs,ys)) = (READ (rs,(BeCo (i,j),NO),ys));
readRhy (READ ([Ts "(",Ts "B",Ts "e", Ts "SC", Ts "(",Ti i, Ts ",", Ti j, Ts ")", Ts
",",:rs],
        xs,ys)) = (READ (rs,(BeSC (i,j),NO),ys));
readRhy (READ ([Ts "(",Ts "B",Ts "e", Ts "F", Ts "u", Ts "(",Ti i, Ts ",", Ti j, Ts
")", Ts ",",:rs],
        xs,ys)) = (READ (rs,(BeFu (i,j),NO),ys));
readRhy (READ ([Ts "(",Ts "B",Ts "e", Ts "SF", Ts "(",Ti i, Ts ",", Tij, Ts ")", Ts
",",:rs],
        xs,ys)) (READ (rs,(BeSF (i,j),NO),ys));
readRhy (READ ([Ts "(",Ts "S",Ts "u", Ts "s", Ts "B", Ts "r", Ts "(",Ti i, Ts ",", Ti
j, Ts ")", Ts ",",:rs],
        xs,ys)) = (READ (rs,(SusBr (i,j),NO),ys));
readRhy (READ ([Ts "(",Ts "S",Ts "u", Ts "s", Ts "SB", Ts "(",Ti i, Ts ",", Tij, Ts
")", Ts ",",:rs],
        xs,ys)) = (READ (rs,(SusSB (i,j),NO),ys));
readRhy (READ ([Ts "(",Ts "S",Ts "u", Ts "s", Ts "M", Ts "i", Ts "(",Ti i, Ts ",", Ti
j, Ts ")",Ts ",",:rs],
        xs,ys)) (READ (rs,(SusMi (i,j),NO),ys));
readRhy (READ ([Ts "(",Ts "S",Ts "u", Ts "s", Ts "SM", Ts "(",Ti i, Ts ",", Ti j, Ts
")", Ts ",",:rs],
        xs,ys)) = (READ (rs,(SusSM (i,j),NO),ys));
readlRhy (READ ([Ts "(",Ts "S",Ts "u", Ts "s", Ts "C", Ts "o", Ts "(",Ti i, Ts ",",
Tij, Ts ")", Ts ",",:rs],
        xs,ys)) = (READ (rs,(SusCo (i,j),NO),ys)),
readRhy (READ ([Ts "(",Ts "S",Ts "u", Ts "s", Ts "SC", Ts "(",Ti i, Ts ",", Tij, Ts
")", Ts ",",:rs],
        xs,ys)) = (READ (rs,(SusSC (i,j),NO),ys));
readRhy (READ ([Ts "(",Ts "S",Ts "u", Ts "s", Ts "F", Ts "u", Ts "(",Ti 1, Ts ",",
Ti j, Ts ")", Ts ",",:rs],
        xs,ys)) = (READ (rs,(SusFu (i,j),NO),ys));
readRhy (READ ([Ts "(",Ts "S",Ts "u", Ts "s", Ts "SF", Ts "(",Ti i, Ts ",", Ti j, Ts
```

```
")", Ts ",":rs],
        xs,ys)) — (READ (rs,(SusSF (i,j),NO),ys)),
readRhy — = abort ("Erro na funcao readRhy");
```

5.3.6) *readNotes*:

Esta função reconhece da lista de tokens aqueles que caracterizam a altura da nota.

```
readNotes Reader -> Reader;
readlNotes (READ ([Ts "A", Ts "(", Tii, Ts ",", Ti j, Ts ",", Ti k, Ts ")", Ts ")":rs],
    (x,y),xs)) (READ (rs,(x,A (i,j,k)),xs));
readlNotes (READ ([Ts "B", Ts "(", Tii, Ts ",", Ti j, Ts ",", Ti k, Ts ")", Ts ")":rs],
    x,y),xs)) = (READ (rs,(x,B (i,j,k)),xs));
readNotes (READ ([Ts "C", Ts "e", Ts "(", Ti i, Ts ",", Ti j, Ts",", Tik, Ts ")", Ts
")":rs],
    (x,y),xs)) = (READ (rs,(x,Cc (i,j,k)),xs));
readNotes (READ ([Ts "D", Ts "(", Tii, Ts ",", Tij, Ts ",", Ti k, Ts ")", Ts ")":rs],
    (x,y),xs)) = (READ (rs,(x,D (i,j,k)),xs));
readNotes (READ ([Ts "M", Ts "(", Ti i, Ts ",", Tij, Ts ",", Ti k, Ts ")", Ts ")":rs],
    x,y),xs)) = (READ (rs,(x,M (i,j,k)),xs));
readNotes (READ ([Ts "F", Ts "(", Tii, Ts ",", Tij, Ts ",", Ti k, Ts ")", Ts ")":rs],
    (x,y),xs)) = (READ (rs,(x,F (i,j,k)),xs));
readlNotes (READ ([Ts "G", Ts "(", Ti i, Ts ",", Ti j, Ts ",", Ti k, Ts ")", Ts ")":rs],
    (x,y),xs)) = (READ (rs,(x,G (i,j,k)),xs));

readNotes (READ ([Ts "A", Ts "sus", Ts "(", Ti i, Ts ",", Ti j, Ts ",", Ti k, Ts ")", Ts
")":rs],
    (x,y),xs)) = (READ (rs,(x,Asus (i,j,k)),xs));
readNotes (READ ([Ts "B" Ts "sus", Ts "(", Ti i, Ts ",", Tij, Ts ",", Ti k, Ts ")", Ts
")":rs],
    (x,y),xs)) = (READ (rs,(x,Bsus (i,j,k)),xs)),
readNotes (READ ([Ts "C", Ts "sus", Ts "(", Ti i, Ts ",", Ti j, Ts ",", Ti k, Ts ")", Ts
")":rs],
    (x,y),xs)) = (READ (rs,(x,Csus (i,j,k)),xs));
readNotes (READ ([Ts "D", Ts "sus", Ts "(", Ti i, Ts ",", Ti j, Ts ",", Ti k, Ts ")", Ts
```

```
")":rs],
        (x,y),xs)) = (READ (rs,(x,Dsus (i,j,k)),xs));
readNotes (READ ([Ts "M", Ts "sus", Ts "(", Ti i, Ts ",", Tij, Ts ",", Ti k, Ts ")", Ts
")":rs],
        (x,y),xs)) = (READ (rs,(x,Msus (i,j,k)),xs));
readNotes (READ ([Ts "F", Ts "sus", Ts "(", Ti i, Ts ",", Ti j, Ts ",", Ti k, Ts ")", Ts
")":rs],
        (x,y),xs)) = (READ (rs,(x,Fsus (i,j,k)),xs));
readNotes (READ ([Ts "G", Ts "sus", Ts "(", Ti i, Ts ",", Ti j, Ts ",", Ti k, Ts ")", Ts
")":rs],
        (x,y),xs)) = (READ (rs,(x,Gsus (i,j,k)),xs));

readlNotes (READ ([Ts "A", Ts "b", Ts "(", Ti i, Ts ",", Tij, Ts ",", Ti k, Ts ")", Ts
")":rs],
        (x,y),xs)) (READ (rs,(x,Ab (i,j,k)),xs));
readNotes (READ ([Ts "B", Ts "b", Ts "(", Ti i, Ts ",", Tij, Ts ",", Ti k, Ts ")", Ts
")":rs],
        (x,y),xs)) = (READ (rs,(x,Bb (i,j,k)),xs));
readNotes (RE.AD ([Ts "C", Ts "b", Ts "(", Tii, Ts ",", Tij, Ts ",", Ti k, Ts ")", Ts
")":rs],
        (x,y),xs)) = (READ (rs,(x,Cb (i,j,k)),xs));
readNotes (RE.AD ([Ts "D", Ts "b", Ts "(", Tii, Ts ",", Tij, Ts ",", Ti k, Ts ")", Ts
")":rs],
        (x,y),xs)) = (READ (rs,(x,Db (i,j,k)),xs)),
readNotes (RE.AD ([Ts "M", Ts "b", Ts "(", Tii, Ts ",", Tij, Ts ",", Ti k, Ts ")", Ts Ts
")":rs],
        (x,y),xs)) = (READ (rs,(x,MIb (i,j,k)),xs));
readNotes (RE.AD ([Ts "F", Ts "b", Ts "(", Tii, Ts ",", Tij, Ts ",", Ti k, Ts ")", Ts
")":rs],
        (x,y),xs)) (READ (rs,(x,Fb (i,j,k)),xs));
readNotes (RE.AD ([Ts "G", Ts "b", Ts "(", Tii, Ts ",", Tij, Ts ",", Ti k, Ts ")", Ts
")":rs],
        (x,y),xs)) (READ (rs,(x,Gb (i,j,k)),xs));
```

5.3.7) *fim*:

fim (READ ([(Ts "FIM")],x,y)) = (READ ([],x,y));

fim = Fl //abort("Erro na funcao fim ");

5.3.8) addNote:

Esta função junta um elemento a uma lista.

addNote (READ (a,b,c)) (READ (,,lst)) = (READ (a,b,[b:lst]));

addNote = abort ("Erro na rotina addNote");

5.4) Módulo *ScoreEdit*:

Este módulo é o principal. Ele controla o editor e o *MIDI*. Parte do projeto está grafado nas figuras 10 e 11. A listagem completa do programa está em anexo. Abordar-se-á aqui apenas alguns aspectos.

A declaração de tipo que vem a seguir irá controlar a ação levada a efeito pelo usuário quando clicar em algum ponto do pentagrama.

```
:: Action = NOTHING | DEL | Breve | SBreve | Minima | SMinima | Colcheia |
                SColcheia | Fusa | Sfusa |
                PBreve | PSBreve | PMinima | PSMinima | Pcolcheia |
                PSColcheia | Pfusa | PSFusa | Bemol        | Sustenido;
```

O mecanismo *World* do *Clean* usa em seu bojo a função *Tools* que irá introduzir as ferramentas gráficas para inserção de notas.

```
Start :: *World -> *World
Start iniworld = snd(StartIO Tools inistate [] iniworld);
where {
        inistate = { g = [], stt= (NOTHING, NOTHING)};
    };
```

5.4.1) *Tools*:

Esta função tem subordinadas as três funções principais deste módulo: *dialSys*, *Wnd* e *ddd*. Juntas, elas controlam a inserção de notas, a transformação do paradigma gráfico no paradigma adotado nesta dissertação como padrão para escrita de notas em

texto e produzem um arquivo *MIDI*.

Tools= [DialogSystem[dialSys], WindowSystem[Wnd]]++ddd;

5.4.2) *Wnd*:

Esta função cria uma janela com um *ID*, posição, título, tamanho e preenchimento. É este preenchimento que são as notas e pentagramas.

```
Wnd=
FixedWindow
Windowld // nro da janela.
(winX, winY) // pos da janela.
"Score Editor" //titulo da janela.
WindowRect //tamanho da janela.
UpdFunction // funcao da janela.
[GoAway Quit, Mouse Able Markit];
```

5.4.3) *UpdFunction*:

Esta função desenha ou redesenha as linhas dos pentagramas e coloca nelas as notas (*mkDrawing*).

```
UpdFunction area s=:{g} =
    (s, EraseRect ++ Score ++ (mkDrawing g));
```

5.4.3.1) *EraseRect*:

Esta função apaga toda janela principal da partitura, linhas, notas, tudo!

```
EraseRect=
    [ SetPenColour BlackColour // ou (RGB 0.0 0.0 0.8)
    , EraseRectangle WindowRect
    ];
```

5.4.4) *Score*:

Esta função desenha os elementos básicos de um pentagrama: Armadura de clave, fórmula de compasso, claves etc.

```
Score = (twoStaffs (htwoStaffs 3 0) (htwoStaffs 3 1)) ++
    (twoStaffs (htwoStaffs 3 2) (htwoStaffs 3 3)) ++
    Clef"&" (gPos 3 0) ++ Clef"?" (fPos 3 1) ++
    timeSignature "4" "4" tsgPosx (htwoStaffs 3 0) (tsgPosy2 3 0) ++
    timeSignature "4" "4" tsgPosx (htwoStaffs 3 1) (tsgPosy2 3 1) ++
    Clef "&" (gPos 3 2) ++ Clef "?" (fPos 3 3) ++
    timeSignature "4" "4" tsgPosx (htwoStaffs 3 2) (tsgPosy2 3 2) ++
    timeSignature "4" "4" tsgPosx (htwoStaffs 3 3) (tsgPosy2 3 3);
```

5.4.5) *mkDrawing*:

O parâmetro "*note*" é do tipo: *(Ritmos, Notas)*, por exemplo: *(Co (cl,c2), A(al,a2,a3))* sendo "*Co*" o ritmo da nota *(cl,c2)* as coordenadas onde se deve colocar a nota (fonte). O parâmetro *a1* é a altura da nota A, *a2* é 1 ou 2 dependendo se a nota está na clave de G ou F e *a3* é o número do compasso onde a nota aparece.

```
mkDrawing []=[];
mkDrawing [note: r]=
        [   SetFont      font2
        ,   MovePenTo    (getPos note) // devolve tipo "Point"
        ,   SetPenMode   OrMode
        ,   DrawString   (NoteSym note) // devolve a string ritmo da nota.
        :   mkDrawing r
        ];
```

5.4.6) *ddd*:

A função *ddd* controla as três ações de manipulação de arquivos do editor: salvar arquivos, salvar arquivo *Midi*, recuperar arquivo de notas (que não o *MIDI*).

```
ddd= [MenuSystem[Files, Dialogo]];
where {
```

5.4.7) sv:

Este trecho salva o formato texto de notas. Este formato é o mesmo que irá ser

usado para a conversão *MIDI*.

```
svinfo s=:{g} io
# txt= "INICIO "+++ (lstStrOfNotes (g) "") +++ " FIM";
 (ok, fn, s, io)=SelectOutputFile "" "" s io;
= writit fn txt s io;
```

5.4.8) *svMidi*:

Este trecho gera um arquivo *MIDI* baseado no estado atual do sistema que possui a lista estrutural de notas (não é texto).

```
svMidi info s=:{g} io
# txt = gerMidi (ToStrManyVoices (g) 2); // GERMIDI
(ok,fn,s,io)=SelectOutputFile "" "" s io;
= writit fn txt s io;
```

5.4.9) *ld*:

Este trecho lê de um arquivo texto para o formato adotado que possui as notas.

```
ld info s io
# (ok, fn, s, io)= SelectInputFile s io;
((open,file),io)= accFiles (fopen2 fn FReadbext) io;
        (txt, file)= freads file 200000;
            (c,io)= accFiles (fclose file) io;
              rs= readScoreTxt txt;
= ( {g=rs,st= (NOTHING,NOTHING)}, DrawInActiveWindow (ReadFont rs ) io);
```

5.4.10) *Files*:

Esta função cria um menu *pull down*

```
Files= PullDownMenu 0 "File" Able [Bye];
```

A função *save* é uma opção do menu que salva a partitura em disco.

```
save= Menultem 0 "Salvar" (Key 'S') Able sv;
```

A função *load* é uma opção do menu que abre um arquivo txt.

```
load= Menultem 1 "Abrir" (Key 'A') Able (ld);
```

A função Smidi é uma opção do menu que salva a partitura no formato MIDI.

Smidi= Menultem 2 "Salvar Em Midi" (Key 'M') Able (svMidi);

5.4.11) *Bye*:

Quando acionada a opção "*Quit*" do menu pull down o editor é encerrado.

Bye =Menultem 3 "Quit" (Key 'Q') Able Quit;

5.4.12) *Quit*:

Quit s io= (s, QuitIO io);

5.4.13) Dialogo:

As funções dos itens 5.4.13 a 5.4.16 criam uma janela com botões que desempenham a mesma tarefa das opções do menu *pull down*: Salvar arquivo, salvar midi, abrir arquivo.

Dialogo= PullDownMenu 0 "FileOperation" Able [dl];

5.4.14) *dl*:

dl= Menultem 0 "Notes and Rests"(Key 'N') Able (fn 1);

5.4.15) *fn*:

fn 1 s io= (s, OpenDialog Notes_and_Rests io);

5.4.16) *Notes_and_Rests:*

```
Notes_and_Rests= CommandDialog Notes_and_RestsID "Salvar como" []0
        [DialogButton 3 Left " Salvar" Able (sv),
                DialogButton 4 (Rightbo 3)" Abrir" Able (ld),
                DialogButton 6 (Rightbo 4)" Midi "Able (svMidi),
                Editlext 5 (Below 3)(MM (toReal 50)) 2 ""];
   g i info s io= (s, Beep io)
};
```

5.4.17) *writit*:

Os parâmetros de entrada são: "*nm*" que é o nome do arquivo em questão, "*txt*"

que é a "variável" cujo conteúdo será escrito no arquivo "*nm*". A operação já está definida e é "*FWriteText*". O parâmetro global "*io*" vai sendo modificado ao longo das chamadas sucessivas.

```
writit nm txt s io
# ((open,file),io)= accFiles (fopen2 nm FWritebText) io;
| not open = (s,Beep io);
# file=fwrites txt file;
(c,io)=accFiles (fclose file) io;

= (s,io);
```

5.4.18) *ReadFont*:

Esta função plota na janela principal o pentagrama completo e as notas.

```
ReadFont bs =
        [EraseRectangle WindowRect,
        SetPenColour BlackColour] ++ Score ++ (mkDrawing bs);
```

5.4.19) *dialSys*:

Esta função cria uma janela com as figuras das durações das notas. Controla ainda os eventos de click do mouse para a inserção de uma nota ou pausa no pentagrama.

```
dialSys= CommandDialog Dialog_id "Notes & Rests" [dlgPos] 0 bbb;
```

5.4.20) *bbb*:

Cada um dos elementos da lista de *bbb* plota uma figura musical na janela "*Notes & Rests*" em forma de botão.

```
bbb=
    [ DialogButton quitID Left "Quit" Able Closit
    ,DialogButton delID (Rightbo quitID) "Del" Able deleteNote
```

```
,DialogButton saveID (Below quitID) "Save" Able Closit //saveFile
,DialogButton midiID (Rightbo saveID) "Midi" Able Closit //midiFile
,CalcButton bID (Below saveID) "W"          font3 1 drawBreve
,CalcButton sbID (Rightbo bID) "w"          font3 1 drawSBreve
,Calcflutton mTiD (Below bID) "h"           font 2 drawMinima
,CalcButton smJD (Rightbo mTiD) "q"         font 2 drawSMinima
,CalcButton coID (Below mTiD) "e"           font 2 drawColcheia
,CalcButton scID (Rightbo coID) "x"         font 2 drawSColcheia
,CalcButton fuID (Below coID) "q"           font 3 drawFusa
,CalcButton sfID (Rightbo fuID) "q"         font 4 drawSFusa
,CalcButton pbID (Below fuID) "a" font 1 drawPBreve
,CalcButton psblD (RightTo pbJD) "î" font 1 drawPSBreve
,CalcButton psmlD (Below pbID) "Î" font3 1 drawPSMinima
,CalcButton pcolD (Rightlo psmlD) "ä" font2 1 drawPColcheia
,CalcButton psclD (Below psmLD) "Ä" font3 1 drawPSColcheia
,CalcButton pfulD (RightTo psclfl) "" font 1 drawPFusa
,CalcButton psflD (Below psclD) "ô" font 1 drawPSFusa
,CalcButton bemollD (Rightlo sbID) "b" font3 1 bemois
,CalcButton suslD (Below bemollD) "#" font3 1 sustenidos
,CalcButton beqlD (Below suslD) "n" font3 1 bequadros
]
where {
    [quitID, delID, saveID,midilD,bID, sbID,miID, smID, coID, scID,fuID, sfID,
        pbID,psbID, pmiID,psmlD, pcoID,pscID,pfulID,psfID,bemolID,
        susID,beqID:ids]= [Button_id...];

        };
```

As funções de 5.4.20.01 a 5.4.20.20 controlam o estado do programa reconhecendo qual nota foi escolhida na janela "*Notes & Rests*". Esta nota é escrita na janela da partitura e colocada numa lista de notas.

5.4.20.01) *drawBreve*:

```
drawBreve info s=: {g} io=
        ({s&stt= (Breve,NOTHING)}
        , DrawInActiveWindow (ReadFont g) io
        );
```

5.4.20.02) *drawSBreve*

```
drawSBreve info s=: {g} io=
        ({ s&stt= (SBreve,NOTHING) }
        , DrawInActiveWindow (ReadFont g) io
        );
```

5.4.20.03) *drawMinima*:

```
drawMinima info s=: {g} io=
        ({s&stt= (Minima,NOTHING)}
        DrawlnActiveWindow (ReadFont g) io
```

5.4.20.04) *drawSMinima*:

```
drawSMinima info s=: {g} io=
        ({ s&stt= (SMinima,NOTHING) }
        , DrawlnActiveWindow (ReadFont g ) io
        );
```

5.4.20.05) *drawColcheia*:

```
drawColcheia info s=:{g} io=
        ( { s&stt (Colcheia,NOTHING) }
        , DrawlnActiveWindow (ReadFont g) io
        );
```

5.4.20.06) *drawSColcheia*:

```
drawSColcheia info s=:{g} io=
```

```
({ s&stt (SColcheia,NOTHING) }
, DrawInActiveWindow (ReadFont g) io
);
```

5.4.20.07) *drawFusa*:

```
drawFusa info s=: {g} io=
        ({ s&stt= (Fusa,NOTHING) io=
        , DrawlnActiveWindow (ReadFont g) io
        );
```

5.4.20.08) *drawSFusa*:

```
drawSFusa info s=:{g} io=
        ({ s&stt= (SFusa,NOTHING) }
        , DrawlnActiveWindow (ReadFont g) io
        );
```

5.4.20.09) *drawPBreve*:

```
drawPBreve info s=: {g} io=
        ({ s&stt= (PBreve,NOTHING) }
        , DrawlnActiveWindow (ReadFont g) io
        );
```

5.4.20.10) *drawPSBreve*:

```
drawPSBreve info s=: {g} io=
        ({ s&stt= (PSBreve,NOTHING) }
        , DrawlnActiveWindow (ReadFont g ) io
        );
```

5.4.20.11) *drawPSMinima*:

```
drawPSMinima info s=: {g} io=
        ({ s&stt= (PSMinima,NOTHING) }
        , DrawlnActiveWindow (ReadFont g) io
        );
```

5.4.20.12) *drawPColcheia*:

```
drawPColcheia info s=: {g} io=
        ({ s&stt= (PColcheia,NOTHING) }
        , DrawlnActiveWindow (ReadFont g ) io
        );
```

5.4.20.13) *drawPSColcheia*:

```
drawPSColcheia info s=: {g} io=
        ({ s&stt= (PSColcheia,NOTI-TING)}
        , DrawInActiveWind.ow (ReadFont g ) io
        );
```

5.4.20.14) *drawPFusa*:

```
drawPFusa info s=:{g} io=
        ({ s&stt= (PFusa,NOTHING) }
        , DrawInActiveWindow (ReadFont g ) io
        );
```

5.4.20.15) *drawPSFusa*:

```
drawPSFusa info s:{g} io=
        ({ s&stt= (PSFusa,NOTHJNG) }
        , DrawlnActiveWindow (ReadFont g ) io
        );
```

5.4.20.16) bemóis:

```
bemois info s:{g,stt} io
#  (x,y):s.stt =
       ( { s&stt(x,Bemol) }
       , DrawInActiveWindow (ReadFont g) io
       );
```

5.4.20.17) sustenidos:

```
sustenidos info s=:{g,stt} io
# (x,y):s.stt =
       ({ s&stt=(x, Sustenido) }
       , DrawInActiveWindow (ReadFont g) io
       );
```

5.4.20.18) bequadros:

```
bequadros info s=: { g, stt } io
#      (x,y)=:s.stt =
       ({ s&stt= (x,NOTHTING) }
       , DrawInActiveWindow (ReadFont g) io
       );
```

5.4.20.19) *deleteNote*:

```
deleteNote info s=:{g} io=
       ({ s&stt= (DEL,NOIHLNG)}
       , DrawInActiveWindow (ReadFont g) io
       );
```

5.4.21) *CalcButton*:

O trabalho desta função é criar uma *DialogBox* com botões em determinada posição com uma *string* que seja seu nome e que aqui designa qual a duração da nota. Esta função cria quatro modelos diferentes de botões.

```
CalcButton :: DialogItemId ItemPos ItemTitle Font Int (ButtonFunction *State IO) ->
                                        DialogItem *State IO;
CalcButton id pos title font type bfunc
        = DialogIconButton id pos buttonDomain (look type) Able bfunc;
where
        {
        buttonWidth = 30;

        buttonHeight = 30;
        buttonDomain = ((0,0),(buttonWidth,buttonHeight));
        bhf   = (buttonHeight-fontLen);
        x     = (buttonWidth-FontStringWidth title font)/2-1;
        look:: Int SelectState -> [DrawFunction];
        look 1
                =[      SetFont         font
                ,       FillRectangle shadowrect
                ,       SetPenColour WhiteColour //ou (RGB 0.5 0.5 0.5)
                ,       FillRectangle buttonrect
                ,       SetPenColour BlackColour
                ,       DrawRectangle       buttonrect
                ,       MovePenTo           (x,bhf)
        ,       SetPenMode          OrMode
,       DrawString          title
,
];
        look 2
                = [             SetFont         font
                ,       FillRectangle shadowrect
                ,       SetPenColour WhiteColour // ou (RGB 0.5 0.5 0.5)
                ,       FillRectangle buttonrect
                ,       SetPenColour BlackColour
                ,       DrawRectangle       buttonrect
                ,       MovePenbo   (x,bhf+5)
                ,       SetPenMode  OrMode
                ,       DrawString  title
```

```
        ];

look 3
        =[              SetFont         font
        ,               FillRectangle shadowrect
        ,               SetPenColour WhiteColour //ou (RGB 0.5 0.5 0.5)
        ,               FillRectangle buttonrect
        ,               SetPenColour BlackColour
                        ,               DrawRectangle   buttonrect
        ,               MovePenTo       (x,bhf±5)
        ,               SetPenMode      OrMode
        ,               DrawString      title
        ,               MovePenTo       (x+(x/2)-1,x/2)
        ,   DrawString "r"
        ,   MovePenlo (x+(x/2)-1,x+2)
        ,   DrawString "K"
        ];

look 4_
        =[          SetFont         font
        ,           FillRectangle shadowrect
        ,           SetPenColour WhiteColour //ou (RGB 0.5 0.5 0.5)
        ,           FillRectangle buttonrect
        ,           SetPenColour BlackColour
        ,           DrawRectangle       buttonrect
        ,           MovePenbo           (x,bhf+5)
        ,           SetPenMode          OrMode
        ,           DrawString          title
        ,           MovePenbo           (x+(x/2)-1,x/2)
        ,   DrawString    "r"
        ,   MovePenTo     (x+(x/2)-1,x+2)
        ,   DrawString    "r"
        ];

shadowrect buttonrect = ((2,2), (buttonWidth,buttonHeight));
```

```
bottonrect          = ((0,0),(buttonWidth-2,buttonHeight-2));

};
```

5.4.22) *Markit*:

Na função abaixo acontece o seguinte: quando se aperta algum dos botões que caracteriza uma nota há uma mudança no estado "*stt*" determinando qual nota foi escolhida, veja as funções *drawSBreve*, *drawMinima*, etc. então quando se clica na partitura o estado atual de "*stt*" é comparado com as opções abaixo e então a função apropriada executa seu trabalho. O parâmetro *pMouse* é a posição do *click* na tela e há um grupo de três estados possíveis ao se escolher uma nota: bequadro, bemol e sustenido. O parâmetro do tipo *Point* que aparece nas funções *addBreve* e suas irmãs é: *(posx, posy, Nota)*, ou seja, idêntico a função *Prx*. Estas funções abaixo montam uma lista do tipo: *(Ritmos,Notas)* exemplo: *(SB(15,45),FSus (5,1,12))*

```
Markit (pMouse, ButtonDown, _) s =: { stt} io =
            case stt of {
  (NOTHING,NOTHING)   -> (s,io);
  (DEL,NOTHING)  -> delNote (pMouse) s io;
  (Breve,NOTHING) -> addBreve (NotePosition "Bequadro" pMouse) s lo;
  (SBreve,NOTHING) -> addSBreve       (NotePosition "Bequadro" pMouse) s io;
  (Minima,NOTHING) -> addMinima (NotePosition "Bequadro" pMouse) s io;
  (SMinima,NOTHING) -> addSMinima (NotePosition "Bequadro" pMouse) s io;
  (Colcheia,NOTHING) -> addColcheia (NotePosition "Bequadro" pMouse) s io;
  (SColcheia,NOTHING) -> addSColcheia (NotePosition "Bequadro" pMouse) s io;
  (Fusa,NOTHING) -> addFusa   (NotePosition "Bequadro" pMouse) s io;
  (SFusa,NOTHING) -> addSFusa (NotePosition "Bequadro" pMouse) s io;
  (PBreve,NOTHING) -> addPBreve (NotePosition "Bequadro" pMouse) S iO;
  (PSBreve,NOTHING) -> addPSBreve (NotePosition "Bequadro" pMouse) s io;
  (PMinima,NOIHLNG) -> addPMinima (NotePosition "Bequadro" pMouse) s io;
  (PSMinima,NOIHJNG) -> addPSMinima (NotePosition "Bequadro" pMouse) s io;
  (PColcheia,NOILILNG) -> addPColcheia (NotePosition "Bequadro" pMouse) s io;
  (PSColcheia,NOTHING)-> addPSColcheia (NotePosition "Bequadro" pMouse)s io;
  (PFusa,NOIHLNG) -> addPFusa (NotePosition "Bequadro" pMouse) s io;
```

```
(PSFusa,NOTHING) -> addPSFusa (NotePosition "Bequadro" pMouse) s io;
(Breve,Bemol)      -> addbBreve (NotePosition "Bemol" pMouse) s io;
(SBreve,Bemol)     -> addbSBreve (NotePosition "Bemol" pMouse) s io;
(Minima,Bemol)     -> addbMinima (NotePosition "Bemol" pMouse) s io;
(SMinima,Bemol)    -> addbSMinima (NotePosition "Bemol" pMouse) s io;
(Colcheia,Bemol)   -> addbColcheia (NotePosition "Bemol" pMouse) s io;
(SColcheia,Bemol)  -> addbSColcheia (NotePosition "Bemol" pMouse) s io;
(Fusa,Bemol)       -> addbFusa  (NotePosition "Bemol" pMouse) s io;
(SFusa,Bemol)      -> addbSFusa (NotePosition "Bemol" pMouse) s io;
(Breve,Sustenido)  -> addSusBreve (NotePosition "Sustenido" pMouse) s io;
(SBreve,Sustenido)-> addSusSBreve (NotePosition "Sustenido" pMouse) s io;
(Minima,Sustenido) -> addSusMinima (NotePosition "Sustenido" pMouse)s io;
(SMinima,Sustenido) -> addSusSMinima (NotePosition "Sustenido" pMouse)s io;
(Colcheia,Sustenido) -> addSusColcheia (NotePosition "Sustenido" pMouse)s io;
(SColcheia, Sustenido) -> addSusSColcheia (NotePosition "Sustenido" pMouse)s io;
(Fusa,Sustenido)   -> addSusFusa (NotePosition "Sustenido" pMouse) s io;
(SFusa,Sustenido)  -> addSusSFusa (NotePosition "Sustenido" pMouse) s io;
otherwise -> (s,io)
};

Markit _ s io = (s,io);
```

5.4.23) *NotePosition*:

Esta função determina de qual nota se trata a partir do *click* do mouse e sua posição num compasso e num pentagrama.

```
NotePosition acd p:(x,y) = Prx acd p rPos wSfgrid;
where {
        Y[Pos
        | y <= (htwoStaffs 3 1 )-(betweenStaff/2) =
                (whatCompass 1 0 x (htwoStaffs 3 0));
        | y <= (htwoStaffs 3 2)-(betweenStaff/2) && y > (htwoStaffs 3 1)-
(betweenStaff/2) =
                (whatCompass 2 O x (htwoStaffs 3 1));
```

```
        | y <= (htwoStaffs 3 3)-(betweenStaff/2) && y > (htwoStaffs 3 2)-
(betweenStaff/2) =
                (whatCompass 1 3 x (htwoStaffs 3 2));
        | y <= (htwoStaffs 3 4)-(betweenStaff/2) && y > (htwoStaffs 3 3)-
(betweenStaff/2) =
                (whatCompass 2 3 x (htwoStaffs 3 3));
        | otherwise = abort ("Erro na rotina NotePosition ");
    rPos = tupla_3_3(YPos);
    wSf = tupla_3_12(YPos);
    };
```

5.4.23.01) *whatCompass*:

A função *whatCompass* devolve *(v,c,y)* onde: "*v*" indica em qual voz a nota está. "*c*" indica o compasso na referida voz e "*y*" é a posição "*y*" da nota.Quem é a nota é determinado por *NotePosition.*

```
whatCompass v c x y
        | x <= wW/3 (v,c+1,y);
        | x > wW/3 && x < 2*wW/3 = (v,c+2,y);
        | x > 2*wW/3 = (v,c+3,y);
```

5.4.24) *Prx*:

Prx devolve a tupla *(posx, posy, Nota)* que descreve a posição de uma nota. Exemplo*: (15,45,FSus (5,1,12))*, quer dizer que *(15,45)* é a posição da nota *FSus(5,1,12)* sendo que 5 é a oitava da nota, 1 é a voz (clave de G) e 12 é o compasso em que a nota está. A função *nG6B4* e suas irmãs detectam qual pentagrama se está usando, já que na mesma posição relativa as notas num e noutro são diferentes, por exemplo, o terceiro espaço do pentagrama de baixo para cima tem-se uma nota que na clave de sol é a nota do5 e na clave de fá é a nota mi3.

```
Prx :: String Point Int (Int,Int) Int -> (!Int,!Jnt,!Notas);
Prx acd (x,y) step whatStaffg
        | y ~ ! (step) && (acd == "Bequadro") = (x,(step), (nG6B4 whatStafi));
```

```
        | y ~ ! (step+g) && (acd == 'Bequadro") = (x,(step+g), (nM6G4 whatStafl));
        | y ~! (step+2*g) && (acd == "Bequadro") = (x,(step+2*g), (nC6M4
whatStaff));
        | y ~! (step+3*g) && (acd =="Bequadro") = (x,(step+3*g), (nA5C4
        whatStaff));
        | y ~! (step+4*g) && (acd =="Bequadro") = (x,(step+4*g), (nF5A3
        whatStafl));
        | y ~! (step+5*g) && (acd =="Bequadro") = (x,(step+4*g), (nD5F3
        whatStafl)); etc.
```

5.4.24.01) *~!*:

Este operador calcula se dois valores se encontram próximos de um valor *h*. Como se pode ver o operador é infixo com prioridade 5.

```
(~!) infix 5:: !Int !Int -> Bool
(~!) yl y2 = yl-y2 < h && y2-yl < h where {h=grid/5};
```

5.4.25) *delNote*:

Esta função apaga uma nota na partitura, retira-a da lista e redesenha a partitura sem ela.

```
delNote point s io=
        let! {nug= eraseNote point s.g}
        in ({ s&g= nug, stt=(NOTHING,NOTHING)}
            , DrawInActiveWindow (ReadFont nug) io
            ),
```

5.4.25.01) *eraseNote*:

Esta função elimina uma nota da lista de notas mantida pelo estado atual do programa.

```
eraseNote p []= [];
eraseNote p [n:r] | p ~=~ getPos(n) = r;
```

```
eraseNote p [n:r] = [n: eraseNote p r];
```

5.4.25.02) ***~=~*:**

Esta função verifica a proximidade de dois pontos, assim se se deseja apagar uma nota não é necessário clicar exatamente no mesmo ponto que deu origem a nota mas num ponto próximo e ela será apagada.

```
(~=~) infix 5 :: !Point !Point -> Bool;

(~=~) (xl,yl) (x2,y2) = xl-x2 <h && x2-xl <h && yl-y2 <h && y2-yl <h where
{h=grid};
```

5.4.26) ***ordVoicAndCompass*:**

Esta função ordena primeiro os compassos depois as vozes. Ao ordernar os compassos, emparelham-se as vozes lado a lado, depois ordena-se as vozes.

```
ordVoicAndCompass orList
#       qsv = qsortCompass orList;
=       qsortVoice qsv;
```

5.4.26.01) ***qsortCompass*:**

Esta função ordena a lista de notas levando-se em consideração qual compasso a nota está. Usou-se o algoritmo da bolha.

```
qsortCompass:: [(Ritmos,Notas)] -> [(Ritmos,Notas)];
qsortCompass [] = [];
qsortCompass [a:xs] = qsortCompass [x \\ x<-xs | x !!< a] ++
                      [a] ++ qsortCompass [x\\x<-xs | not (x !!< a)];
```

5.4.26.02) ***qsortVoice*:**

O mesmo que o anterior, considerando-se a voz como critério de ordenação.

```
qsortVoice:: [(Ritmos,Notas)] -> [(Ritmos,Notas)];
qsortVoice [] = [];
```

```
qsortVoice [a:xs] = qsortVoice [x || x<-xs | x !< a] ++
                    [a] ++ qsortVoice [x||x<-xs | not (x !< a)];
```

5.4.26.03) ***!<:***

Este operador determina se a voz dentro da declaração de tipo *(Ritmos,Notas)* é menor que a outra.

```
(!<) infix 5 :: ¡ (Ritmos, Notas) !(Ritmos, Notas) -> Bool;
(!<)yl y2
# voicl whatVoice yl;
  voic2 = whatVoice y2;
=  if(voicl < voic2) True False;
```

5.4.26.04) ***!!<:***

Este operador determina se o compasso dentro da declaração de tipo *(Ritmos,Notas)* é menor que outro.

```
(!!<) infix 5 :: ! (Ritmos, Notas) ! (Ritmos, Notas) -> Bool;
(!!<)yl y2
#       voe1 = takeCompass y1;
        voc2 = takeCompass y2;
= if(vocl < voc2) True False;
```

5.4.27) ***fopen2:***

Esta função recebe três parâmetros de entrada a saber: *fileName* que é o nome do arquivo. *mode* que é o tipo de operação que se vai fazer com o arquivo, leitura ou escrita, *files* que é a parte do mundo (*world*) pertinente a arquivos que está sendo manipulada no momento. O que a função devolve é: "*ok*" que é um booleano que diz se a operação foi bem sucedida. "*file*" que é o arquivo modificado pela operação *mode* e *files2* que é a parte do mundo relativa a arquivos agora modificada pela função.

```
fopen2 fileName mode files :== ((ok,file),files2);
where {
        (ok,file,files2) = fopen fileName mode files
```

```
};
```

5.4.28) *fontTpl, font, font2, font3, font4*

Estas funções são para o controle das fontes, usadas no programa.

```
fontTpl (k,f)= f;
font = fontTpl( SelectFont "Petrucci" [] fontLen);
font2 = fontTpl( SelectFont "Petrucci" [] (2*fontLen));
font3 = fontTpl( SelectFont "Petrucci" [] 16);
font4 = fontTpl( SelectFont "Petrucci" [] (3*fontLen));
```

CAPÍTULO VI

COMO USAR O EDITOR MUSICAL SIMPLES

6.1) Aspectos gerais:

O editor de partituras simples é composto de três objetos estruturais que são: Um menu *pull-down*, uma janela com ferramentas de manipulação de notas e a partitura localizada na janela principal

O menu que possui apenas duas opções *File/Quit (CtrI+Q)* e *FileOperation/ Notes And Rests (Ctrl+N),* Esta última opção abrirá a janela *"Notes &. Rests"* que possui as ferramentas necessárias para se introduzir notas na partitura.

A janela *"Notes &. Rests"* possui uma coleção de botões que serve para se introduzir urna figura musical na partitura, ou para sair do sistema, salvar arquivo MIDI, salvar arquivo texto ou ainda apagar urna nota.

6.2) **Utilizando o editor:**

Tomando a figura 13 pode-se ver a janela *"Notes &. Rests"* cada um dos botões possui uma função. O botão nomeado como *"Quit"* irá fechar o aplicativo e é preciso tornar cuidado! Deve-se salvar a partitura antes de sair. O botão *"Del"* irá apagar uma nota que por ventura foi colocada errada na partitura.. Para isto basta apertar este botão e clicar nas proximidades da nota em questão. Como já foi dito, há um conjunto de funções e operadores definidos pelo usuário, que cuidam para que seja válido um conjunto de pontos em se falando de uma nota. Qualquer nota, ao ser apagada, irá desaparecer da lista de notas e qualquer nota inserida em qualquer lugar de um pentagrama, irá aparecer na lista que possui a estrutura ordenada destas notas.

Para se inserir uma nota na partitura basta clicar no botão correspondente à figura desejada e clicar novamente no ponto onde se deseja inseri-la no pentagrama. Ao se clicar no botão, o estado do programa é alterado e a nota escolhida é ritmicamente interpretada. Ao se clicar num dos dois pentagramas, o ponto de inserção da nota é analisado e, através dele, a nota é nomeada, finalizando o processo. A nota então passa a fazer parte de uma lista. Esta lista não é ordenada. A seqüência correta das notas é, na verdade, interpretada por um dos índices da estrutura de sua formação. Esta forma de

construção permite serem inseridas e apagadas notas em qualquer lugar de qualquer um dos pentagramas. Uma nota apagada é retirada da lista, não importando o lugar onde ela esteja nesta lista, no entanto, uma nota é sempre inserida no final da lista.
Ha uma limitação para a inserção de notas ou pausa, é que ela deve ser feita sempre no limite dos pentagramas, ou seja, não há linhas complementares.

Outra limitação é que, apesar do MIDI aceitar a alteração em tempo de execução, qualquer um dos parâmetros musicais, tais como mudança de compasso, de armadura, de instrumento etc, o editor impõe o modelo rítmico 4 por 4.

Pelo que foi dito, em parágrafo anterior, a seqüência de inserção das notas é irrelevante, portanto, pode-se inserir notas num ou noutro pentagrama até ser preenchido ritmicamente o compasso. Cuidado, porém, com o valor destas notas porque a editor não faz verificação se a soma total dos tempos das notas está dentro da unidade de compasso.

CAPÍTULO VII

CONCLUSÃO

O objetivo do trabalho foi criar uma forma de se escrever urna partitura musical num formato intermediário entre o musical e o MIDI. Isto tem uma vantagem imediata: a de não exigir de uma equipe formada por músicos e engenheiros vastos conhecimentos da interface MIDI destes, nem vastos conhecimento de música daqueles. Ao se trabalhar num projeto comum a curva de aprendizado poderia ser minimizada.

O editor é muito inferior, em recursos, em relação ao MIDI. Algumas modificações dever ser feitas para diminuir esta distância tais como se fazer uma distribuição-espaço-de-pentagrama, por valor de nota. A verificação da soma dos tempos individuais que deve ser igual à unidade de compasso, se o valor ultrapassou deve-se gerar mensagens de aviso e não permitir o avanço na inserção de notas. Algumas notas como apogiaturas, não possuem valor real e, portanto, não podem entrar na conta. Fica a cargo do executante perceber quanto e onde retirar tempo de notas para não comprometer a unidade de compasso. Permitir a inserção de notas complementares. Promover a ligação das hastes de notas que podem e, freqüentemente, devem ser ligadas, mas permitir também que possam ser desligadas (quando se tratar de urna partitura de canto, por exemplo).

CAPÍTULO VIII

BIBLIOGRAFIA

LACERDA, OSWALDO. Compêndio de teoria elementar da música. Ricordi brasileira, 1966

ZAMACOIS. JOAQUIN. Teoria de la musica. Barcelona. Labor, 1983

ZAMACOIS, JOAQUIN. Tratado de Armonia. Barcelona, Labor, 1984

MENEZES, PAULO BLAUTH. Linguagens Formais e Autômatos. Porto alegre. Sagra Luzzato, 1998

RATTON, MIGUEL BALLOUSSIER. Midi guia básico de referência, Rio de Janeiro, Campus, 1992

IMA THE INTERNATIONAL MIDI ASSOCIATION. MIDI especification 1.0. California.

CLEAN STAFF. Functional Programming in Clean, Netherland, 1999

APÊNDICE A

Módulo LexMidi0899

```
implementation module lexMidi0899;

import StdEnv,Tokens21;

:: Notation=
        C [Notation] | VOICE [Notation] |
                        Nt (String,String, Int, Real) |
        Nm String | Oc Int  | Dr Real |
        Sr String | Nr Int  |
        N [(String,String, Int,Real)] |
        ME [String]  |
        Nil;

:: Result = OK(Notation, [Tk]) | Fail;

/* Gerenciamento do erro no Clean */

ErrorReport  syntaxError  xs e =abort (syntaxError+++position)
        where
                {       x= xs+++"\n";
                        position = (x%(0, e)) +++ "<-Here->"+++(x%(e+1,size x-1))
                }

$ = toChar;

/* ----------------------finalPoint----------------------------*/
finalPoint :: Result -> Result;
finalPoint (OK(_,[Ts ".":rest])) = OK(Nil,rest);
finalPoint xs = Fail;
```

```
/* ------------------------openParen-------------------------*/
openParen :: Result -> Result;
openParen (OK(_, [Ts "(":rest])) = OK(Nil, rest);
openParen xs= Fail;

/* ------------------------closeParen-------------------------*/
closeParen :: Result -> Result;
closeParen (OK(_,  [Ts ")":rest])) = OK(Nil, rest);
closeParen xs= Fail;

/* ------------------------openBra-------------------------*/
openBra :: Result -> Result;
openBra (OK(_,  [Ts "[":rest])) = OK(Nil, rest);
openBra xs= Fail;

/*------------------------closeBra-------------------------*/
closeBra :: Result -> Result;
closeBra (OK(x,  [Ts "]":rest])) = OK(x,rest);
closeBra xs= Fail;

/* ------------------------comma-------------------------*/
comma :: Result -> Result;
comma (OK(_, [Ts ",":rest])) = OK(Nil, rest);
comma _ = Fail;
comma Fail = ErrorReport " comma  " "     " 3;

/* ------------------------semiColon-------------------------*/
semiColon :: Result -> Result;
semiColon (OK(_,  [Ts ";":rest])) = OK(Nil, rest);
semiColon xs= Fail;

/*---- Duration - basRhy - AllRhy - findPointValue - basicRhy ----------*/

Duration :: Result -> Result;
```

```
    Duration (OK(_, [Ts x, Ts "d", Ts "d", Ts "d", Ts "d", Ts ")":rest]))
            | basRhy x = OK(Dr  (findPointValue (AllRhy x 0.0) 4), rest);
    Duration (OK(_, [Ts x, Ts "d", Ts "d", Ts "d", Ts ")":rest]))
            | basRhy x =  OK(Dr (findPointValue (AllRhy x 0.0) 3), rest);
    Duration (OK(_, [Ts x, Ts "d", Ts "d", Ts ")" :rest]))
            | basRhy x = OK(Dr (findPointValue (AllRhy x 0.0) 2), rest);
    Duration (OK(_, [Ts x, Ts "d", Ts ")" :rest]))
            | basRhy x = OK(Dr (findPointValue (AllRhy x 0.0) 1), rest);
    Duration (OK(_, [Ts x, Ts ")":rest]))
            | basRhy x = OK(Dr (findPointValue (AllRhy x 0.0) 0), rest);
    Duration (OK(_, [Ts x, Ti n, Ts ")":rest]))
            | basRhy x = OK(Dr (findPointValue (AllRhy x (toReal n)) 0), rest);
    Duration (OK(_, [Ts x, Ts "d", Ti n, Ts ")":rest]))
            | basRhy x = OK(Dr (findPointValue (AllRhy x (toReal n)) 1), rest);
    Duration _= Fail;

    basRhy x = isMember x ["b" ,"sb", "m", "sm", "c",  "sc", "f", "sf","q","sq"];

    AllRhy :: {#Char} Real -> Real; // Trata quialteras.
    AllRhy r n
            |n == 0.0   = basicRhy r;
            |otherwise =    2.0*(basicRhy r)/n;

    findPointValue :: Real Int  -> Real; // Trata notas pontuadas.
    findPointValue s i = return s i 0 0.0;
    where
      {
          return s i j ac
                  |i<j= ac; // divide o valor "s", que eh do tipo real,  i vezes.
                  |otherwise = return (s/2.0) i (j+1) (s+ac);
          };

basicRhy ::  {#Char}   -> Real;
```

```
basicRhy br
        |br ==   "b"     = 8.0;
        |br ==   "sb"    = 4.0;
        |br ==   "m"     = 2.0;
        |br ==   "sm"    = 1.0;
        |br ==   "c"     = 0.5;
        |br ==   "sc"    = 0.25;
        |br ==   "f"     = 0.125;
        |br ==   "sf"    = 0.0625;
        |br ==   "q"     = 0.03125;
        |br ==   "sq"    = 0.015625;
        |otherwise       = 0.0;


/* ------------------------------ MidiEvent -------------------------- */
MidiEvent :: Result -> Result;
MidiEvent (OK(_, [Ti a0,  Ts "FF", Ti a1, Ti a2, Ti a3, Ti a4,Ti a5,Ti a6: rest]))
        = OK(Sr  ({$ a0, $ 0xFF, $ a1, $ a2, $ a3,$ a4,$ a5,$ a6 } ), rest);
MidiEvent (OK(_, [Ti b0,  Ts "FF", Ti b1, Ti b2, Ti b3, Ti b4: rest]))
        = OK(Sr ({$ b0, $ 0xFF, $ b1,$ b2,$ b3,$ b4}), rest);
MidiEvent (OK(_, [Ti b0,  Ts "FF", Ti 2, Ts "F", Ti  b: rest]))
        = OK(Sr ({$ b0,$ 0xFF,$ 0x2F,$ b} ), rest);
MidiEvent (OK(_, [Ti b0,  Ts "C", Ti c: rest]))
        = OK(Sr ({$ b0,$ 0xC,$ c} ), rest);
MidiEvent _= abort (" Erro na rotina   MidiEvent ");


/*  ---------------------- Name - noteName - Octave ----------------------*/
noteName :: String -> Bool;
noteName x = isMember x ["A", "B", "C", "D",  "E", "F", "G", "P"];

Name :: Result -> Result;
Name (OK(_, [Ts x, Ts "#", Ts "#":rest]))
        | noteName x = OK(Nm (x+++"##"), rest);
```

```
Name (OK(_, [Ts x, Ts "#":rest]))
        | noteName x =  OK(Nm (x+++"#"), rest);
Name (OK(_, [Ts x, Ts "b", Ts "b":rest]))
        | noteName x = OK(Nm (x+++"bb"), rest);
Name (OK(_, [Ts x, Ts "b":rest]))
        | noteName x = OK(Nm (x+++"b"), rest);
Name (OK(_, [Ts x, Ts "%":rest]))
        | noteName x = OK(Nm (x+++"%"), rest);
Name (OK(_, [Ts x:rest]))
        | noteName x = OK(Nm x, rest);
Name _= abort ( "Erro na rotina:     Name ");

Octave :: Result -> Result;
Octave (OK(_, [Ti i:rest])) = OK(Oc i, rest);
Octave _ = abort ("Erro na rotina Octave");
```

APÊNDICE B

Módulo Sytx0899

```
implementation module Sytx0899;

import StdEnv,lexMidi0899;


ErrorReport  syntaxError  xs e =abort (syntaxError+++position)
        where
                {       x= xs+++"\n";
                        position = (x%(0, e)) +++ "<-Here->"+++(x%(e+1,size x-1))
                }

$ = toChar;

/* Simplesmente verifica se o parametro de T eh do tipo OK */

T  (OK(_,_)) = True;
T  _ = False;

R (OK(_, xs))= xs;
R Fail= abort "Unknown error!";

F (OK(s, _))= s;
F Fail= abort "Unknown error!";

FF (OK(Nt s, _))= s;
FF Fail= abort "Unknown error!";

MEC (OK( Sr s, _ )) =s;
MEC Fail= abort "Unknown error!";
```

```
/*----------------------------------parsit---------------------------------*/
parsit :: [Tk] -> [[Notation]];
parsit xs = Score  xs;

/*----------------------------------Score----------------------------------*/
Score [] = [];
Score tks = [c1:more]
where {
        (c1, xs) = getTks(Voice (OK(Nil, tks)));
           more = Score xs
    };

/*----------------------------------Voice----------------------------------*/
Voice ok
#     v = takeVoice ok;
=   Avoice v;

takeVoice (OK (x,[Ts "VOICE", Ti i, Ts "-":rest])) = OK(x,rest);
takeVoice _ = abort ("Erro na funcao takeVoice");


/*----------------------------------Avoice---------------------------------*/
Avoice ok
#           lastCp = compass ok;
      fvoice = finalVoice lastCp | T fvoice = OK (VOICE [fV lastCp], R fvoice);

Avoice ok
#      cp = compass ok;
      rl = RdLine cp;
 restOfCp = Avoice  rl | T restOfCp = addCp cp restOfCp;

Avoice _ = abort("Erro na rotina Avoice");
```

```
finalVoice :: Result -> Result;
finalVoice (OK(_,[Ts "FINALVOICE":rest])) = OK (Nil,rest);
finalVoice xs = Fail;

fV (OK(m,n)) = m;
fV Fail = abort ("Erro na funcao  fV");

addCp (OK (m,_)) (OK (VOICE ns,rest)) = OK( VOICE [m:ns],rest);
addCp _ _ = abort ("Erro no modulo addCp");

/* ----------RdLine---------- */

RdLine (OK(x , [Ti i, Ts "-": rest])) = (OK(x, rest));
RdLine Fail = ErrorReport " RdLine " "     " 3;

/*----------getTks----------*/
getTks (OK (VOICE t1, xs)) = (t1, xs);
getTks Fail= abort ("Erro no modulo getTks");

/*----------compass----------*/

compass (OK(_, [Ts ";":rest])) = OK( C [], rest);
compass ok // MAISCOMPASSO -> ";" COMPASSO
        #       m1 = melody ok;
        restOfCompass = compass m1 | T restOfCompass = addCompass m1
restOfCompass;
compass _       = abort ("Erro na rotina   compass");
/*----------melody----------*/

melody ok
        #       paren = openParen ok; // openParen esta em lexMidi0899.
        m = notes paren | T  m = m;
melody _     = abort ("Erro na rotina  melody");
```

```
                /* melody so retira  o token  (Ts "(" ) */

 /*---------------------------------addCompass-------------------------------*/

 addCompass (OK(m1, _))  (OK(C ns, rest))  = OK(C [m1:ns], rest);
 addCompass _ _  = abort ( "Erro na rotina addCompass ");

 /* ---------------------------- notes - addNote ----------------------------*/

 notes ok
        #  lastNote = note ok;
            rest = closeParen lastNote | T rest=  OK(N [FF lastNote], R rest);

 notes ok // MAISNOTAS -> , NOTAS
        #       n1 = note ok ;
         c  = comma n1;
        restOfNotes = notes c |  T restOfNotes = addNote n1 restOfNotes;

 notes _        = abort ("Erro na rotina  notes");

 /* --------------------- mkNote - notOkME - whoNext - note ---------------------*/
 note  ok
        #   nm = Name ok; // Name, Octave, openParen e Duration esta em lexMidi0899
    octave = Octave nm;
                op = openParen octave;
       me = whoNext op;  // MIDIEVENTO-> (EVENTOS . TEMPO)
            d = Duration  me | T d = mkNote (me, nm,  octave,  d);
 note _     = abort( "  Erro na rotina:   note");

 whoNext op
        |  okME op = MidiEventStr op;
        | otherwise = notOkME op;

 mkNote (OK(Sr s,_), OK(Nm nm, _), OK(Oc oc, _), OK(Dr n, rest))=
```

```
                    OK(Nt(s,nm,oc,n), rest);

 okME (OK(x,[Ti n: rest])) = True;
 okME _ = False;

 notOkME (OK(x,y))=(OK(Sr "",y));

 /* ----------MidiEventStr----------*/

 MidiEventStr ok = meStr;
 where
    {
      meList  = MidiEventList ok;
      meStr  = joinMEs meList;
    };
 /*  Devolve  => OK( (Sr  "MidiEventosBlocoString" ), rest)     */
 /*----------MidiEventList----------
   os midi-eventos tem tamanho fixo.
   A ordem das funcoes abaixo eh rigorosamente esta.
   comma, finalPoint e MidiEvente esta no modulo lexMidi0899
 */
 MidiEventList ok
   #   lastME = MidiEvent ok;
       fp = finalPoint lastME | T fp  = OK (ME [MEC lastME], R fp);

 MidiEventList ok
       #     me  = MidiEvent ok;   // MAISEVENTOS-> "," EVENTO
       c  = comma me;
    restOfME = MidiEventList c | T restOfME = addME me restOfME;

 MidiEventList _   = abort("Erro na rotina    MidiEventList");

 /* nao se pode colocar este trecho antes do outros ou sempre
 ocorrera Fail.   Produz =>  (OK(ME  ["String","String", ...] ,   [restOfTokens]))   */
```

```
/*-------------------------------joinMEs-----------------------------------*/

joinMEs (OK(ME lst, rest)) = OK( (Sr (foldr (+++) "" lst)), rest)  ;

// junta os varios midi-eventos num so bloco string.

/*-------------------------------addME-------------------------------------*/

addME (OK(Sr n1, _)) (OK(ME ns, rest))  = OK(ME [n1:ns], rest);
addME _ _ = abort "Erro na rotina      addME";
```

APÊNDICE C

Módulo Midi0899

```
implementation module Midi0899;

import StdEnv, Sytx0899;

:: HdTrack =
        {
         timeSignature :: Int,
         formula       :: Int,
         ticks         :: Int,
         nFusas        :: Int,
         keySignature  :: Int,
         accidents     :: Int,  /* sf */
         mode          :: Int,  /* mi */
         tempo         :: Int,
    channel      :: Int,
    instrument   :: Int
        };

$ = toChar; // sinonimo
/* ----------------------------Manipulacao de Tuplas-------------------------*/

fst2Tupla  (x,y)      = x;
sec2Tupla  (x,y)      = y;
fst4Tupla  (x,y,w,z)  = x;
sec4Tupla  (x,y,w,z)  = y;
thr4Tupla  (x,y,w,z)  = w;
for4Tupla  (x,y,w,z)  = z;
fif5Tupla  (a,b,c,d,e) = e;
red5To4Tupla(a,b,c,d,e) = (a,b,c,d);
```

```
/* ----------------------------------AppendT-----------------------------------*/
AppendT  []    s = s;
AppendT  [x:r] s = [x: AppendT r s];

/* ----------------------------------notes-------------------------------------*/

notesMidi :: [Notation] Int -> {#Char};
notesMidi nl ppq = return;
where {
        tnl = allNotesInOneLst nl;
         p = (pauses tnl 0 "" [] ppq);
        ns =  map (rhyAndNote ppq)  p;
     return = foldr (+++)  "" ns;
    };

/*-------------------------prepareNotes-allNotesInOneLst----------------------*/

allNotesInOneLst :: [Notation] -> [(String,String,Int,Real)];
allNotesInOneLst  allC = flatten (prepareNotes allC);

prepareNotes :: [Notation] -> [[(String,String,Int,Real)]];
prepareNotes allC = map oneCompass allC;

/* ----------------------------------oneCompass--------------------------------*/

oneCompass (C [(N lst)]) = lst;

/* ----------------------------------pauses------------------------------------*/
pauses nl ac1 me ac2 ppq
        | length nl == 0 = ac2;
        | (onePBool (hd(nl))) =  pauses (tl(nl)) (ac1 +(PpqNote  (hd(nl)) ppq ))
                                 (me+++(fst4Tupla (hd(nl))))  ac2 ppq;
        | otherwise = pauses (tl(nl)) 0 me (ac2 ++ [(me+++(fst4Tupla (hd(nl))),
```

```
                                    sec4Tupla (hd(nl)),

                                    thr4Tupla (hd(nl)),
                                    for4Tupla (hd(nl)),
                                    ac1)]) ppq;


onePBool (_,"P",_,_)=True;
onePBool _ = False;

/* ----------------------------------rhyAndNote--------------------------------*/

rhyAndNote :: Int (String,String,Int,Real,Int)  -> String;
rhyAndNote ppq tuplaNote   = return;
where {
   (me,note, hi, nd,bfn) = tuplaNote;
                np= if (bfn >0)
                        {toChar((hi7f bfn) +128), toChar(lo7f bfn)}
                        {$0};
                r = red5To4Tupla(tuplaNote);
               ndi =  PpqNote r ppq;
              ndppq = {toChar((hi7f ndi) + 128), toChar(lo7f ndi)};
                n = (NotaMidi note hi);
             return = me+++np +++ {$0x90,n,$0x7F} +++ ndppq +++ {$0x80,n,$0}
    };

/* ----------------------------------PpqNote-----------------------------------*/

PpqNote :: (String,String,Int,Real) Int -> Int;
PpqNote nota ppq = ppqnote
where  {
    noteval =  for4Tupla nota;
    ppqnote = toInt ( noteval *  (toReal ppq));
     };
```

```
/* ------------------------------Quebra de numero------------------------------*/
Hi x = toChar ((x/65536) / 256);
Lo x = toChar ((x/65536) mod 256);
hi x = toChar ((x mod 65536) / 256);
lo x = toChar ((x mod 65536) mod 256);

/* ----------------------------------hdrChk----------------------------------*/
hdrChk :: Int Int Int -> String;
hdrChk format notracks division =
    {'M','T','h','d'} +++ //concatena string
    {$0, $0, $0, $6} +++
    {$0, $format} +++
    {hi notracks, lo notracks} +++
    {hi division, lo division}; // ppq

/* ----------------------------------TrkChunk----------------------------------*/
/* nt eh puramente as notas ja escritas como strings, t eh um record com dados do
cabecalho do track */

TrkChunk :: HdTrack   String -> String;
TrkChunk t nt = th+++lentrk+++restTrack+++n
where {
        th= {'M','T','r','k'};
        lt = (size restTrack) + (size n);
        restTrack=(three 0 t.timeSignature) +++
                (three 4 t.formula) +++
                {$ t.ticks, $t.nFusas} +++
                (three 0 t.keySignature) +++

                {$2, $t.accidents, $t.mode} +++
                {$0, $0xFF, $0x51} +++
                {$3, Lo t.tempo, hi t.tempo, lo t.tempo} +++
                {$0, $t.channel,$t.instrument};
```

```
        n =  nt +++ {$0, $0xFF,$0x2F, $0};
             lentrk = {Hi lt, Lo lt, hi lt,  lo lt};

        three i s = {$ i, hi s, lo s}
    };

/* ----------------------------------NotaMidi----------------------------------*/

NotaMidi :: String Int -> Char;
NotaMidi nota altura
        | (isMember nota [ "B#", "C", "Dbb"])    = $( 12* altura);
        | (isMember nota [ "B##","C#", "Db" ])   = $((12* altura)+1);
        | (isMember nota [ "C##","D", "Ebb"])    = $((12* altura)+2);
        | (isMember nota [ "D#", "Eb","Fbb"])    = $((12* altura)+3);
        | (isMember nota [ "D##","E", "Fb"])     = $((12* altura)+4);
        | (isMember nota [ "E#", "F",  "Gbb"])   = $((12* altura)+5);
        | (isMember nota [ "E##","F#",  "Gb"])   = $((12* altura)+6);
        | (isMember nota [ "F##","G", "Abb"])    = $((12* altura)+7);
        | (isMember nota [ "G#", "Ab"])          = $((12* altura)+8);
        | (isMember nota [ "G##","A", "Bbb"])    = $((12* altura)+9);
        | (isMember nota [ "A#", "Cbb", "Bb" ])  = $((12* altura)+10);
        | (isMember nota [ "A##","B", "Cb"])     = $((12* altura)+11);
        | otherwise = $(0);


/* ----------------------------------hi7f-lo7f----------------------------------*/

hi7f x =  ((x mod  16384) / 128);//contador de 128s
lo7f x =  ((x mod  16384) mod 128);
```

APÊNDICE D

Módulo ger1199

```
implementation module ger1199;

import StdEnv, Midi0899;

/* x mod y eh a diferenca entre x e y sendo x>y */


      ppq :== 0x400;
    format :== 1;
  noTracks :== 2;

chk = {
      timeSignature = 0xFF58,
                formula = 0x0402,
                ticks = 0x18,
                nFusas = 0x8,
                keySignature = 0xFF59,
                accidents = 0,
                mode = 0,
                tempo = 0x98968,
                channel = 0xC0,
                instrument = 0
   };

gerMidi :: String -> String;
gerMidi score
# notesLst  = (parsit(Tokens score)); //  devolve uma lista do tipo [C [N (..)]...]
     v1   = (hd(notesLst));
     v2   = (hd(tl(notesLst)));
```

```
    voice1   = notesMidi v1 ppq;
    voice2   = notesMidi v2 ppq;
=   (hdrChk format noTracks  ppq)+++
    (TrkChunk chk voice1) +++ (TrkChunk chk voice2);
```

APÊNDICE E

Módulo ConvertToMidi

```
implementation module ConvertToMidi;

import StdEnv,deltaEventIO,TypeRhyNote;

Append []    s = s;
Append [x:r] s = [x: Append r s];

/*------------------------------ NoteSym -----------------------------*/

NoteSym :: (Ritmos,Notas) -> String;
NoteSym((Br  p,_))= "W";
NoteSym((SB  p,_))= "w";
NoteSym((Mi  p,_))= "h";
NoteSym((SM  p,_))= "q";
NoteSym((Co  p,_))= "e";
NoteSym((SC  p,_))= "x";
NoteSym((Fu  p,_))= " ";
NoteSym((SF  p,_))= " ";

NoteSym((PB  p,_))= "ã";
NoteSym((PSB p,_))= "î";
NoteSym((PMi p,_))= "î";
NoteSym((PSM p,_))= "Î";
NoteSym((PCo p,_))= "ä";
NoteSym((PSC p,_))= "Å";
NoteSym((PFu p,_))= "'";
NoteSym((PSF p,_))= "ô";

NoteSym((BeBr  p,_))= "bW";
```

```
NoteSym((BeSB  p,_))= "bw";
NoteSym((BeMi  p,_))= "bh";
NoteSym((BeSM  p,_))= "bq";
NoteSym((BeCo  p,_))= "be";
NoteSym((BeSC  p,_))= "bx";
NoteSym((BeFu  p,_))= "b ";
NoteSym((BeSF  p,_))= "b ";

NoteSym((SusBr  p,_))= "#W";
NoteSym((SusSB  p,_))= "#w";
NoteSym((SusMi  p,_))= "#h";
NoteSym((SusSM  p,_))= "#q";
NoteSym((SusCo  p,_))= "#e";
NoteSym((SusSC  p,_))= "#x";
NoteSym((SusFu  p,_))= "# ";
NoteSym((SusSF  p,_))= "# ";

NoteSym _ = abort "NoteSym falhou!";

/*---------------------------- RitmoStr ----------------------------*/

RitmoStr (Br _,_) = " (b)";
RitmoStr (SB _,_) = " (sb)";
RitmoStr (Mi _,_) = " (m)";
RitmoStr (SM _,_) = " (sm)";
RitmoStr (Co _,_) = " (c)";
RitmoStr (SC _,_) = " (sc)";
RitmoStr (Fu _,_) = " (f)";
RitmoStr (SF _,_) = " (sf)";
RitmoStr (PB _,_) = " (b)";
RitmoStr (PSB _,_) = " (sb)";
RitmoStr (PMi _,_) = " (m)";
RitmoStr (PSM _,_) = " (sm)";
RitmoStr (PCo _,_) = " (c)";
```

```
RitmoStr (PSC _,_) = " (sc)";
RitmoStr (PFu _,_) = " (f)";
RitmoStr (PSF _,_) = " (sf)";
RitmoStr (BeBr _,_) = " (b)";
RitmoStr (BeSB _,_) = " (sb)";
RitmoStr (BeMi _,_) = " (m)";
RitmoStr (BeSM _,_) = " (sm)";
RitmoStr (BeCo _,_) = " (c)";
RitmoStr (BeSC _,_) = " (sc)";
RitmoStr (BeFu _,_) = " (f)";
RitmoStr (BeSF _,_) = " (sf)";
RitmoStr (SusBr _,_) = " (b)";
RitmoStr (SusSB _,_) = " (sb)";
RitmoStr (SusMi _,_) = " (m)";
RitmoStr (SusSM _,_) = " (sm)";
RitmoStr (SusCo _,_) = " (c)";
RitmoStr (SusSC _,_) = " (sc)";
RitmoStr (SusFu _,_) = " (f)";
RitmoStr (SusSF _,_) = " (sf)";

/*------------------------------ NotaStr ----------------------------*/

NotaStr (_,A (x,y,z)) = "A"+++toString(x);
NotaStr (_,B (x,y,z)) = "B"+++toString(x);
NotaStr (_,Cc (x,y,z)) = "C"+++toString(x);
NotaStr (_,D (x,y,z)) = "D"+++toString(x);
NotaStr (_,M (x,y,z)) = "E"+++toString(x);
NotaStr (_,F (x,y,z)) = "F"+++toString(x);
NotaStr (_,G (x,y,z)) = "G"+++toString(x);

NotaStr (_,Asus (x,y,z)) = "A#"+++toString(x);
NotaStr (_,Bsus (x,y,z)) = "B#"+++toString(x);
NotaStr (_,Csus (x,y,z)) = "C#"+++toString(x);
NotaStr (_,Dsus (x,y,z)) = "D#"+++toString(x);
```

```
NotaStr (_,Msus (x,y,z)) = "E#"+++toString(x);
NotaStr (_,Fsus (x,y,z)) = "F#"+++toString(x);
NotaStr (_,Gsus (x,y,z)) = "G#"+++toString(x);

NotaStr (_,Ab (x,y,z)) = "Ab"+++toString(x);
NotaStr (_,Bb (x,y,z)) = "Bb"+++toString(x);
NotaStr (_,Cb (x,y,z)) = "Cb"+++toString(x);
NotaStr (_,Db (x,y,z)) = "Db"+++toString(x);
NotaStr (_,Mb (x,y,z)) = "Eb"+++toString(x);
NotaStr (_,Fb (x,y,z)) = "Fb"+++toString(x);
NotaStr (_,Gb (x,y,z)) = "Gb"+++toString(x);

/*------------------------------ ToStrManyVoices ----------------------------*/

ToStrManyVoices :: [(Ritmos,Notas)] Int -> {#Char};
ToStrManyVoices listaNt nOfc
# svoice = SeparateVoice listaNt [] [];
= ScoreStr svoice nOfc "";

/*------------------------------ ScoreStr ----------------------------*/

ScoreStr orList nOfc strc
        |length(orList) == 0 = strc;
        |otherwise = ScoreStr (tl(orList)) nOfc (strc+++(ToStrOneVoice (hd(orList))
nOfc));

/*------------------------------ SeparateVoice ----------------------------*/

SeparateVoice :: [(Ritmos,Notas)] [(Ritmos,Notas)] [(Ritmos,Notas)] ->
[[(Ritmos,Notas)]];
SeparateVoice listaOr l1 l2
        | length(listaOr) == 0 = [l1]++[l2];
        | (whatVoice (hd(listaOr)) ) == 1 = SeparateVoice (tl(listaOr)) (Append l1
[(hd(listaOr))]) l2;
```

```
        | (whatVoice (hd(listaOr)) ) == 2 = SeparateVoice (tl(listaOr)) l1 (Append l2
[(hd(listaOr))]);

/*---------------------------- ToStrOneVoice ---------------------------*/

ToStrOneVoice orList nOfc
#   oneVoiceLst = SeparateCompass orList nOfc 1 [];
=   (ToStrAllCompass  oneVoiceLst "");

/*---------------------------- SeparateCompass---------------------------*/

SeparateCompass oneVoice nOfc count ac
        | count == (nOfc+1) = ac;
        | otherwise = SeparateCompass oneVoice nOfc (count+1) (Append ac
[takeCompassN oneVoice count []]);

/*---------------------------- takeCompassN---------------------------*/
takeCompassN orList n ac
        | length(orList) == 0 = ac;
        | (takeCompass (hd(orList)) ) == n = takeCompassN (tl(orList)) n (Append ac
[(hd(orList))]);
        | otherwise = takeCompassN (tl(orList)) n ac;


/*---------------------------- ToStrAllCompass ---------------------------*/
ToStrAllCompass orList strc
        | length(orList) == 0 = " VOICE " +++ strc +++ " FINALVOICE ";
        | otherwise = ToStrAllCompass (tl(orList)) (strc+++(oneCompass (hd(orList)) ));

/*---------------------------- oneCompass ----------------------------*/

oneCompass orList
#     hdn   =  (hd(orList));
  idxIntOfc =  takeCompass (hdn);
```

```
  idxStrOfc =  toString(idxIntOfc) +++ "-(";
  notes     =  oneCompassSub orList "";
= idxStrOfc +++ (notes%(0,size(notes)-2)) +++ ");";

/*------------------------------ oneCompassSub ----------------------------*/

oneCompassSub oneScoreLst pent
        | length(oneScoreLst) == 0 = pent;
        | otherwise = oneCompassSub (tl(oneScoreLst)) (pent+++(assemblyNoteStr
(hd(oneScoreLst)))+++",");

assemblyNoteStr elem
#  ntStr=NotaStr elem;
        rtStr=RitmoStr elem;
=  ntStr +++ rtStr;
```

APÊNDICE F

Módulo SaveScore

```
implementation module SaveScore;

import StdEnv,TypeRhyNote;

$ = toString;

/*------------------------------ rhyStr --------------------------------*/

rhyStr((Br  (x,y) ,_))= "(Br (" +++ $x +++ "," +++ $y +++ "),";
rhyStr((SB  (x,y) ,_))= "(SB (" +++ $x +++ "," +++ $y +++ "),";
rhyStr((Mi  (x,y) ,_))= "(Mi (" +++ $x +++ "," +++ $y +++ "),";
rhyStr((SM  (x,y) ,_))= "(SM (" +++ $x +++ "," +++ $y +++ "),";
rhyStr((Co  (x,y) ,_))= "(Co (" +++ $x +++ "," +++ $y +++ "),";
rhyStr((SC  (x,y) ,_))= "(SC (" +++ $x +++ "," +++ $y +++ "),";
rhyStr((Fu  (x,y) ,_))= "(Fu (" +++ $x +++ "," +++ $y +++ "),";
rhyStr((SF  (x,y) ,_))= "(SF (" +++ $x +++ "," +++ $y +++ "),";

rhyStr((SF  (x,y) ,_))= "(SF (" +++ $x +++ "," +++ $y +++ "),";
rhyStr((PB  (x,y) ,_))= "(PB (" +++ $x +++ "," +++ $y +++ "),";
rhyStr((PSB (x,y) ,_))= "(PSB (" +++ $x +++ "," +++ $y +++ "),";
rhyStr((PMi (x,y) ,_))= "(PMi (" +++ $x +++ "," +++ $y +++ "),";
rhyStr((PSM (x,y) ,_))= "(PSM (" +++ $x +++ "," +++ $y +++ "),";
rhyStr((PCo (x,y) ,_))= "(PCo (" +++ $x +++ "," +++ $y +++ "),";
rhyStr((PSC (x,y) ,_))= "(PSC (" +++ $x +++ "," +++ $y +++ "),";
rhyStr((PFu (x,y) ,_))= "(PFu (" +++ $x +++ "," +++ $y +++ "),";
rhyStr((PSF (x,y) ,_))= "(PSF (" +++ $x +++ "," +++ $y +++ "),";

rhyStr((BeBr  (x,y) ,_))= "(BeBr (" +++ $x +++ "," +++ $y +++ "),";
rhyStr((BeSB  (x,y) ,_))= "(BeSB (" +++ $x +++ "," +++ $y +++ "),";
```

```
rhyStr((BeMi  (x,y) ,_))= "(BeMi (" +++ $x +++ "," +++ $y +++ "),";
rhyStr((BeSM  (x,y) ,_))= "(BeSM (" +++ $x +++ "," +++ $y +++ "),";
rhyStr((BeCo  (x,y) ,_))= "(BeCo (" +++ $x +++ "," +++ $y +++ "),";
rhyStr((BeSC  (x,y) ,_))= "(BeSC (" +++ $x +++ "," +++ $y +++ "),";
rhyStr((BeFu  (x,y) ,_))= "(BeFu (" +++ $x +++ "," +++ $y +++ "),";
rhyStr((BeSF  (x,y) ,_))= "(BeSF (" +++ $x +++ "," +++ $y +++ "),";

rhyStr((SusBr  (x,y) ,_))= "(SusBr (" +++ $x +++ "," +++ $y +++ "),";
rhyStr((SusSB  (x,y) ,_))= "(SusSB (" +++ $x +++ "," +++ $y +++ "),";
rhyStr((SusMi  (x,y) ,_))= "(SusMi (" +++ $x +++ "," +++ $y +++ "),";
rhyStr((SusSM  (x,y) ,_))= "(SusSM (" +++ $x +++ "," +++ $y +++ "),";
rhyStr((SusCo  (x,y) ,_))= "(SusCo (" +++ $x +++ "," +++ $y +++ "),";
rhyStr((SusSC  (x,y) ,_))= "(SusSC (" +++ $x +++ "," +++ $y +++ "),";
rhyStr((SusFu  (x,y) ,_))= "(SusFu (" +++ $x +++ "," +++ $y +++ "),";
rhyStr((SusSF  (x,y) ,_))= "(SusSF (" +++ $x +++ "," +++ $y +++ "),";

rhyStr _ = abort "rhyStr falhou!";

/*---------------------------- NoteStr ------------------------------*/

NoteStr (_,A (x,y,z))  = "A("+++ $ x +++","+++ $ y+++","+++ $ z +++")),";
NoteStr (_,B (x,y,z))  = "B("+++ $ x +++","+++ $y +++","+++ $ z +++")),";
NoteStr (_,Cc (x,y,z)) = "Cc("+++ $ x +++","+++ $y +++","+++ $ z +++")),";
NoteStr (_,D (x,y,z))  = "D("+++ $ x +++","+++ $y +++","+++ $ z +++")),";
NoteStr (_,M (x,y,z))  = "M("+++ $ x +++","+++ $y +++","+++ $ z +++")),";
NoteStr (_,F (x,y,z))  = "F("+++ $ x +++","+++ $y +++","+++ $ z +++")),";
NoteStr (_,G (x,y,z))  = "G("+++ $ x +++","+++ $y +++","+++ $ z +++")),";

NoteStr (_,Asus (x,y,z)) = "Asus("+++ $ x +++","+++ $ y +++","+++ $ z +++")),";
NoteStr (_,Bsus (x,y,z)) = "Bsus("+++ $ x +++","+++ $ y +++","+++ $ z +++")),";
NoteStr (_,Csus (x,y,z)) = "Csus("+++ $ x +++","+++ $ y +++","+++ $ z +++")),";
NoteStr (_,Dsus (x,y,z)) = "Dsus("+++ $ x +++","+++ $ y +++","+++ $ z +++")),";
NoteStr (_,Msus (x,y,z)) = "Msus("+++ $ x +++","+++ $ y +++","+++ $ z +++")),";
NoteStr (_,Fsus (x,y,z)) = "Fsus("+++ $ x +++","+++ $ y +++","+++ $ z +++")),";
```

```
NoteStr (_,Gsus (x,y,z)) = "Gsus("+++ $ x +++","+++ $ y +++","+++ $ z +++")),";

NoteStr (_,Ab (x,y,z)) = "Ab("+++ $ x +++","+++ $ y +++","+++ $ z +++")),";
NoteStr (_,Bb (x,y,z)) = "Bb("+++ $ x +++","+++ $ y +++","+++ $ z +++")),";
NoteStr (_,Cb (x,y,z)) = "Cb("+++ $ x +++","+++ $ y +++","+++ $ z +++")),";
NoteStr (_,Db (x,y,z)) = "Db("+++ $ x +++","+++ $ y +++","+++ $ z +++")),";
NoteStr (_,Mb (x,y,z)) = "Mb("+++ $ x +++","+++ $ y +++","+++ $ z +++")),";
NoteStr (_,Fb (x,y,z)) = "Fb("+++ $ x +++","+++ $ y +++","+++ $ z +++")),";
NoteStr (_,Gb (x,y,z)) = "Gb("+++ $ x +++","+++ $ y +++","+++ $ z +++")),";

lstStrOfNotes :: [(Ritmos,Notas)] {#Char} -> {#Char};
lstStrOfNotes [] str = (str%(0,size(str)-2));
lstStrOfNotes [s:r] str = lstStrOfNotes r (rhyStr (s) +++ NoteStr (s) +++ str );
```

APÊNDICE E

Módulo ReadScoreTxt

```
implementation module ReadScoreTxt;

import StdEnv, TypeRhyNote;

iniReading (READ ([Ts "INICIO":r],x,y)) = (READ (r,x,y));
iniReading _ = abort("Erro na funcao iniReading");

fim (READ ([(Ts "FIM")],x,y)) = (READ ([],x,y));
fim _ = Fl; //abort("Erro na funcao fim.");

comma (READ ([Ts ",":r],x,y)) = (READ (r,x,y));
comma _ = Fl; //abort("Erro na funcao comma");

TREST (READ (_,_,_)) = True;
TREST _              = False;

Append [] s = s;
Append [x:r] s = [x: Append r s];

/*---------------------------- readScoreTxt ----------------------------*/

readScoreTxt :: String -> [(Ritmos,Notas)];
readScoreTxt readStr
#               tk = Tokens readStr;
=       readText tk;

/*---------------------------- readText ----------------------------*/
readText :: [Tk] -> [(Ritmos,Notas)];
readText read
#               ini = iniReading (READ (read,(NOT,NO),[]));
```

```
    (READ (x,y,z)) = readToList ini;
= z;

/*--------------------------- readToList -------------------------*/

readToList read
#       lastRhy = readRhy read;
       lastNote = readNotes lastRhy;
  (READ (x,y,z)) = lastNote;
         rest = fim lastNote | TREST rest = READ (x,y,[y:z]) ;

readToList read
#              rh  = readRhy read;
          rn  = readNotes rh;
           c  = comma rn;
    restOfNotes = readToList c | TREST restOfNotes = addNote c restOfNotes;

readToList _  =  abort ("Erro na funcao readToList");

/*--------------------------- addNote ----------------------*/

addNote (READ (a,b,c)) (READ (_,_,lst)) = (READ (a,b,[b:lst]));
addNote _ _ = abort ("Erro na rotina addNote");

/*--------------------------- readRhy ----------------------*/
readRhy :: Reader -> Reader;
readRhy (READ ([Ts "(",Ts "B",Ts "r",Ts "(",Ti i,Ts "," ,Ti j,Ts ")",Ts ",":rs],
          xs,ys)) = (READ (rs,(Br (i,j),NO),ys));
readRhy (READ ([Ts "(",Ts "SB",Ts "(",Ti i, Ts ",", Ti j, Ts ")", Ts ",":rs],
          xs,ys)) = (READ (rs,(SB (i,j),NO),ys));
readRhy (READ ([Ts "(",Ts "M",Ts "i",Ts "(",Ti i,Ts "," ,Ti j,Ts ")",Ts ",":rs],
          xs,ys)) = (READ (rs,(Mi (i,j),NO),ys));
readRhy (READ ([Ts "(",Ts "SM",Ts "(",Ti i, Ts ",", Ti j, Ts ")", Ts ",":rs],
          xs,ys)) = (READ (rs,(SM (i,j),NO),ys));
```

```
readRhy (READ ([Ts "(",Ts "C",Ts "o",Ts "(",Ti i,Ts "," ,Ti j,Ts ")",Ts ",":rs],
        xs,ys)) = (READ (rs,(Co (i,j),NO),ys));
readRhy (READ ([Ts "(",Ts "SC",Ts "(",Ti i, Ts ",", Ti j, Ts ")", Ts ",":rs],
        xs,ys)) = (READ (rs,(SC (i,j),NO),ys));
readRhy (READ ([Ts "(",Ts "F",Ts "u",Ts "(",Ti i,Ts "," ,Ti j,Ts ")",Ts ",":rs],
        xs,ys)) = (READ (rs,(Fu (i,j),NO),ys));
readRhy (READ ([Ts "(",Ts "SF",Ts "(",Ti i, Ts ",", Ti j, Ts ")", Ts ",":rs],
        xs,ys)) = (READ (rs,(SF (i,j),NO),ys));
readRhy (READ ([Ts "(",Ts "PB",Ts "(",Ti i, Ts ",", Ti j, Ts ")", Ts ",":rs],
        xs,ys)) = (READ (rs,(PB (i,j),NO),ys));
readRhy (READ ([Ts "(",Ts "PSB",Ts "(",Ti i, Ts ",", Ti j, Ts ")", Ts ",":rs],
        xs,ys)) = (READ (rs,(PSB (i,j),NO),ys));
readRhy (READ ([Ts "(",Ts "PM",Ts "i", Ts "(",Ti i, Ts ",", Ti j, Ts ")", Ts ",":rs],
        xs,ys)) = (READ (rs,(PMi (i,j),NO),ys));
readRhy (READ ([Ts "(",Ts "PSM",Ts "(",Ti i, Ts ",", Ti j, Ts ")", Ts ",":rs],
        xs,ys)) = (READ (rs,(PSM (i,j),NO),ys));
readRhy (READ ([Ts "(",Ts "PC",Ts "o", Ts "(",Ti i, Ts ",", Ti j, Ts ")", Ts ",":rs],
        xs,ys)) = (READ (rs,(PCo (i,j),NO),ys));
readRhy (READ ([Ts "(",Ts "PSC",Ts "(",Ti i, Ts ",", Ti j, Ts ")", Ts ",":rs],
        xs,ys)) = (READ (rs,(PSC (i,j),NO),ys));
readRhy (READ ([Ts "(",Ts "PF",Ts "u", Ts "(",Ti i, Ts ",", Ti j, Ts ")", Ts ",":rs],
        xs,ys)) = (READ (rs,(PFu (i,j),NO),ys));
readRhy (READ ([Ts "(",Ts "PSF",Ts "(",Ti i, Ts ",", Ti j, Ts ")", Ts ",":rs],
        xs,ys)) = (READ (rs,(PSF (i,j),NO),ys));
readRhy (READ ([Ts "(",Ts "B",Ts "e", Ts "B", Ts "r", Ts "(",Ti i, Ts ",", Ti j, Ts ")",
Ts ",":rs],
        xs,ys)) = (READ (rs,(BeBr (i,j),NO),ys));
readRhy (READ ([Ts "(",Ts "B",Ts "e", Ts "SB", Ts "(",Ti i, Ts ",", Ti j, Ts ")", Ts
",":rs],
        xs,ys)) = (READ (rs,(BeSB (i,j),NO),ys));
readRhy (READ ([Ts "(",Ts "B",Ts "e", Ts "M", Ts "i", Ts "(",Ti i, Ts ",", Ti j, Ts ")",
Ts ",":rs],
        xs,ys)) = (READ (rs,(BeMi (i,j),NO),ys));
```

```
readRhy (READ ([Ts "(",Ts "B",Ts "e", Ts "SM", Ts "(",Ti i, Ts ",", Ti j, Ts ")", Ts
",":rs],
        xs,ys)) = (READ (rs,(BeSM (i,j),NO),ys));
readRhy (READ ([Ts "(",Ts "B",Ts "e", Ts "C", Ts "o", Ts "(",Ti i, Ts ",", Ti j, Ts ")",
Ts ",":rs],
        xs,ys)) = (READ (rs,(BeCo (i,j),NO),ys));
readRhy (READ ([Ts "(",Ts "B",Ts "e", Ts "SC", Ts "(",Ti i, Ts ",", Ti j, Ts ")", Ts
",":rs],
        xs,ys)) = (READ (rs,(BeSC (i,j),NO),ys));
readRhy (READ ([Ts "(",Ts "B",Ts "e", Ts "F", Ts "u", Ts "(",Ti i, Ts ",", Ti j, Ts ")",
Ts ",":rs],

        xs,ys)) = (READ (rs,(BeFu (i,j),NO),ys));
readRhy (READ ([Ts "(",Ts "B",Ts "e", Ts "SF", Ts "(",Ti i, Ts ",", Ti j, Ts ")", Ts
",":rs],
        xs,ys)) = (READ (rs,(BeSF (i,j),NO),ys));
readRhy (READ ([Ts "(",Ts "S",Ts "u", Ts "s", Ts "B", Ts "r", Ts "(",Ti i, Ts ",", Ti j,
Ts ")", Ts ",":rs],
        xs,ys)) = (READ (rs,(SusBr (i,j),NO),ys));
readRhy (READ ([Ts "(",Ts "S",Ts "u", Ts "s", Ts "SB", Ts "(",Ti i, Ts ",", Ti j, Ts ")",
Ts ",":rs],
        xs,ys)) = (READ (rs,(SusSB (i,j),NO),ys));
readRhy (READ ([Ts "(",Ts "S",Ts "u", Ts "s", Ts "M", Ts "i", Ts "(",Ti i, Ts ",", Ti j,
Ts ")", Ts ",":rs],
        xs,ys)) = (READ (rs,(SusMi (i,j),NO),ys));
readRhy (READ ([Ts "(",Ts "S",Ts "u", Ts "s", Ts "SM", Ts "(",Ti i, Ts ",", Ti j, Ts ")",
Ts ",":rs],
        xs,ys)) = (READ (rs,(SusSM (i,j),NO),ys));
readRhy (READ ([Ts "(",Ts "S",Ts "u", Ts "s", Ts "C", Ts "o", Ts "(",Ti i, Ts ",", Ti j,
Ts ")", Ts ",":rs],
        xs,ys)) = (READ (rs,(SusCo (i,j),NO),ys));
readRhy (READ ([Ts "(",Ts "S",Ts "u", Ts "s", Ts "SC", Ts "(",Ti i, Ts ",", Ti j, Ts ")",
Ts ",":rs],
        xs,ys)) = (READ (rs,(SusSC (i,j),NO),ys));
```

```
readRhy (READ ([Ts "(",Ts "S",Ts "u", Ts "s", Ts "F", Ts "u", Ts "(",Ti i, Ts ",", Ti j,
Ts ")", Ts ",":rs],
        xs,ys)) = (READ (rs,(SusFu (i,j),NO),ys));
readRhy (READ ([Ts "(",Ts "S",Ts "u", Ts "s", Ts "SF", Ts "(",Ti i, Ts ",", Ti j, Ts ")",
Ts ",":rs],
        xs,ys)) = (READ (rs,(SusSF (i,j),NO),ys));
readRhy _ = abort ("Erro na funcao readRhy");

/*---------------------------- readNotes -------------------------*/
readNotes :: Reader -> Reader;
readNotes (READ ([Ts "A", Ts "(", Ti i, Ts ",", Ti j, Ts ",", Ti k, Ts ")", Ts ")":rs],
      (x,y),xs)) = (READ (rs,(x,A (i,j,k)),xs));
readNotes (READ ([Ts "B", Ts "(", Ti i, Ts ",", Ti j, Ts ",", Ti k, Ts ")", Ts ")":rs],
      (x,y),xs)) = (READ (rs,(x,B (i,j,k)),xs));
readNotes (READ ([Ts "C", Ts "c", Ts "(", Ti i, Ts ",", Ti j, Ts ",", Ti k, Ts ")", Ts
")":rs],
      (x,y),xs)) = (READ (rs,(x,Cc (i,j,k)),xs));
readNotes (READ ([Ts "D", Ts "(", Ti i, Ts ",", Ti j, Ts ",", Ti k, Ts ")", Ts ")":rs],
      (x,y),xs)) = (READ (rs,(x,D (i,j,k)),xs));
readNotes (READ ([Ts "M", Ts "(", Ti i, Ts ",", Ti j, Ts ",", Ti k, Ts ")", Ts ")":rs],
      (x,y),xs)) = (READ (rs,(x,M (i,j,k)),xs));
readNotes (READ ([Ts "F", Ts "(", Ti i, Ts ",", Ti j, Ts ",", Ti k, Ts ")", Ts ")":rs],
      (x,y),xs)) = (READ (rs,(x,F (i,j,k)),xs));
readNotes (READ ([Ts "G", Ts "(", Ti i, Ts ",", Ti j, Ts ",", Ti k, Ts ")", Ts ")":rs],
      (x,y),xs)) = (READ (rs,(x,G (i,j,k)),xs));

readNotes (READ ([Ts "A", Ts "sus", Ts "(", Ti i, Ts ",", Ti j, Ts ",", Ti k, Ts ")", Ts
")":rs],
      (x,y),xs)) = (READ (rs,(x,Asus (i,j,k)),xs));
readNotes (READ ([Ts "B", Ts "sus", Ts "(", Ti i, Ts ",", Ti j, Ts ",", Ti k, Ts ")", Ts
")":rs],
      (x,y),xs)) = (READ (rs,(x,Bsus (i,j,k)),xs));
readNotes (READ ([Ts "C", Ts "sus", Ts "(", Ti i, Ts ",", Ti j, Ts ",", Ti k, Ts ")", Ts
")":rs],
```

```
        (x,y),xs)) = (READ (rs,(x,Csus (i,j,k)),xs));
    readNotes (READ ([Ts "D", Ts "sus", Ts "(", Ti i, Ts ",", Ti j, Ts ",", Ti k, Ts ")", Ts
    ")":rs],
        (x,y),xs)) = (READ (rs,(x,Dsus (i,j,k)),xs));
    readNotes (READ ([Ts "M", Ts "sus", Ts "(", Ti i, Ts ",", Ti j, Ts ",", Ti k, Ts ")", Ts
    ")":rs],
        (x,y),xs)) = (READ (rs,(x,Msus (i,j,k)),xs));
    readNotes (READ ([Ts "F", Ts "sus", Ts "(", Ti i, Ts ",", Ti j, Ts ",", Ti k, Ts ")", Ts
    ")":rs],
        (x,y),xs)) = (READ (rs,(x,Fsus (i,j,k)),xs));
    readNotes (READ ([Ts "G", Ts "sus", Ts "(", Ti i, Ts ",", Ti j, Ts ",", Ti k, Ts ")", Ts
    ")":rs],
        (x,y),xs)) = (READ (rs,(x,Gsus (i,j,k)),xs));

    readNotes (READ ([Ts "A", Ts "b", Ts "(", Ti i, Ts ",", Ti j, Ts ",", Ti k, Ts ")", Ts
    ")":rs],
        (x,y),xs)) = (READ (rs,(x,Ab (i,j,k)),xs));

    readNotes (READ ([Ts "B", Ts "b", Ts "(", Ti i, Ts ",", Ti j, Ts ",", Ti k, Ts ")", Ts
    ")":rs],
        (x,y),xs)) = (READ (rs,(x,Bb (i,j,k)),xs));
    readNotes (READ ([Ts "C", Ts "b", Ts "(", Ti i, Ts ",", Ti j, Ts ",", Ti k, Ts ")", Ts
    ")":rs],
        (x,y),xs)) = (READ (rs,(x,Cb (i,j,k)),xs));
    readNotes (READ ([Ts "D", Ts "b", Ts "(", Ti i, Ts ",", Ti j, Ts ",", Ti k, Ts ")", Ts
    ")":rs],
        (x,y),xs)) = (READ (rs,(x,Db (i,j,k)),xs));
    readNotes (READ ([Ts "M", Ts "b", Ts "(", Ti i, Ts ",", Ti j, Ts ",", Ti k, Ts ")", Ts
    ")":rs],
        (x,y),xs)) = (READ (rs,(x,Mb (i,j,k)),xs));
    readNotes (READ ([Ts "F", Ts "b", Ts "(", Ti i, Ts ",", Ti j, Ts ",", Ti k, Ts ")", Ts
    ")":rs],
        (x,y),xs)) = (READ (rs,(x,Fb (i,j,k)),xs));
```

```
readNotes (READ ([Ts "G", Ts "b", Ts "(", Ti i, Ts ",", Ti j, Ts ",", Ti k, Ts ")", Ts
")":rs],
    (x,y),xs)) = (READ (rs,(x,Gb (i,j,k)),xs));
```

APÊNDICE H

Módulo ScoreEdit

```
module ScoreEdit;

import StdEnv, intrface;
import deltaFileSelect, deltaEventIO, deltaWindow, deltaFont, deltaPicture,
    deltaDialog, deltaFileSelect, deltaIOState,deltaMenu;
import ConvertToMidi,ger1199,SaveScore,ReadScoreTxt;

:: Action= NOTHING | DEL | Breve | SBreve | Minima | SMinima | Colcheia |
            SColcheia | Fusa | SFusa |
              PBreve | PSBreve | PMinima | PSMinima | PColcheia |
            PSColcheia | PFusa | PSFusa | Bemol | Sustenido;

:: *State= { g :: ![(Ritmos, Notas)]
      , stt :: (!Action,!Action)
      };

:: *IO              :==     IOState *State;

fontLen = 12;
grid    = 10;

gScore  = 30; // inicio do primeiro pentagrama, G.
fScore  = gScore+8*grid; // inicio do segundo pentagrama, F.

Start :: *World -> *World;
Start iniworld = snd(StartIO Tools inistate [] iniworld);

where {
            inistate = { g= [], stt= (NOTHING,NOTHING)};
```

```
        };

    Tools= [DialogSystem[dialSys], WindowSystem[Wnd]]++ddd;

    Button_id  = 112;
    Edit_id    = 200;
    WindowId   = 114;
    Dialog_id  = 100;

    wWidth   = 20*grid;
    wHeight  = 5*grid;
    winX     = 0;
    winY     = 0;

    wW= 4*wWidth;
    wH= 7*wHeight;

    Notes_and_RestsID = 1100;

/*---------------------------- fopen2 -----------------------------*/

fopen2 fileName mode files :== ((ok,file),files2);
where {
        (ok,file,files2) = fopen fileName mode files
         };

/*---------------------------- clos -----------------------------*/

clos file files
# (ok,files)=fclose file files;
= files;

/*---------------------------- writit----------------------------*/
```

```
    writit nm txt s io
    # ((open,file),io)=      accFiles (fopen2 nm FWriteText) io;
     | not open = (s,Beep io);
    # file=fwrites txt file;
     (c,io)=accFiles (fclose file) io;
    = (s,io);

    /*------------------------ ddd--------------------------*/

    ddd= [MenuSystem[Files, Dialogo]];
    where {

     sv info s=:{g} io
     # txt= "INICIO "+++ (lstStrOfNotes (g) "") +++ " FIM";
      (ok, fn, s, io)=SelectOutputFile "" "" s io;
     = writit fn txt s io;

    svMidi info s=:{g} io
    # txt = gerMidi (ToStrManyVoices (g) 2); // GERMIDI
     (ok,fn,s,io)=SelectOutputFile "" "" s io;
    = writit fn txt s io;

   ld info s io
   #  (ok, fn, s, io)= SelectInputFile  s io;
    ((open,file),io)= accFiles (fopen2 fn FReadText) io;
       (txt, file)= freads file 200000;
          (c,io)= accFiles (fclose file) io;
             rs= readScoreTxt txt;
  = ( {g=rs,stt= (NOTHING,NOTHING)}, DrawInActiveWindow (ReadFont rs ) io);

 Files= PullDownMenu 0 "File" Able [Bye];
 save  =MenuItem 0 "Salvar" (Key 'S') Able sv;
 load  =MenuItem 1 "Abrir" (Key 'A') Able (ld);
 Smidi =MenuItem 2 "Salvar Em Midi" (Key 'M') Able (svMidi);
```

```
    Bye  =MenuItem 3 "Quit" (Key 'Q') Able Quit;
    Quit s io= (s, QuitIO io);
    Dialogo= PullDownMenu 0 "FileOperation" Able [d1];
    d1= MenuItem 0 "Notes and Rests"(Key 'N') Able (fn 1);
    fn 1 s io= (s, OpenDialog Notes_and_Rests io);
    Notes_and_Rests= CommandDialog Notes_and_RestsID "Salvar como" [] 0
     [DialogButton 3 Left " Salvar " Able (sv),
      DialogButton 4 (RightTo 3)" Abrir " Able (ld),
      DialogButton 6 (RightTo 4)" Midi " Able (svMidi),
      EditText 5 (Below 3)(MM (toReal 50)) 2 ""];
   g i info s io= (s, Beep io)
        };


 /*------------------------- ordVoicAndCompass--------------------------*/

 ordVoicAndCompass orList
 #   qsv = qsortCompass orList;
 =   qsortVoice qsv;

 /*------------------------- qsortVoice---------------------------*/

 qsortVoice :: [(Ritmos,Notas)] -> [(Ritmos,Notas)];
 qsortVoice []    =  [];
 qsortVoice [a:xs] =  qsortVoice [x \\ x<-xs | x !< a] ++
                      [a] ++ qsortVoice [x\\x<-xs | not (x !< a)];

/*------------------------- qsortCompass-------------------------*/

qsortCompass :: [(Ritmos,Notas)] -> [(Ritmos,Notas)];
qsortCompass []    =  [];
qsortCompass [a:xs] =  qsortCompass [x \\ x<-xs | x !!< a] ++
                       [a] ++ qsortCompass [x\\x<-xs | not (x !!< a)];
```

```
/*------------------ Linhas Horizontais e Verticais----------------*/

WindowRect= ((0,0),(wW, wH));

horizontalLines 0 = [];
horizontalLines is =
        [ DrawCLine ((0,is),        (wW,is))          BlackColour
        , DrawCLine ((0,is+grid),   (wW,is+grid))     BlackColour
        , DrawCLine ((0,is+(2*grid)), (wW,is+(2*grid)))  BlackColour
        , DrawCLine ((0,is+(3*grid)), (wW,is+(3*grid)))  BlackColour
        , DrawCLine ((0,is+(4*grid)), (wW,is+(4*grid)))  BlackColour
        ];

vertLineLim1 = wW/3;
vertLineLim2 = 2*wW/3;

verticalLines is = [ DrawCLine ((vertLineLim1,is), (vertLineLim1, is+(4*grid)))
BlackColour] ++
        [ DrawCLine ((vertLineLim2,is), (vertLineLim2, is+(4*grid))) BlackColour
];

twoStaffs gc fc = (horizontalLines gc) ++ (verticalLines gc) ++
        (horizontalLines fc) ++ (verticalLines fc);

Wnd=
    FixedWindow
    WindowId    // nro da janela.
    (winX, winY)  // pos da janela.
    "Score Editor" // titulo da janela.
    WindowRect // tamanho da janela.
    UpdFunction // funcao da janela.
    [GoAway Quit, Mouse Able Markit];
```

```
heightStaff   = 4*grid;
betweenStaff  = 4*grid;
oneStaff      = heightStaff + betweenStaff;
htwoStaffs n m = (n*grid)+m*oneStaff;
gPos n m = (htwoStaffs n m) + 2*grid;
fPos n m = (htwoStaffs n m) + grid;
tsgPosx     = 5*grid;
tsgPosy2 n m =  (htwoStaffs n m) + 3*grid;

Score = (twoStaffs (htwoStaffs 3 0) (htwoStaffs 3 1)) ++
      (twoStaffs (htwoStaffs 3 2) (htwoStaffs 3 3)) ++
      Clef "&" (gPos 3 0) ++ Clef "?" (fPos 3 1) ++
      timeSignature "4" "4" tsgPosx (htwoStaffs 3 0) (tsgPosy2 3 0) ++
      timeSignature "4" "4" tsgPosx (htwoStaffs 3 1) (tsgPosy2 3 1) ++
      Clef "&" (gPos 3 2) ++ Clef "?" (fPos 3 3) ++
      timeSignature "4" "4" tsgPosx (htwoStaffs 3 2) (tsgPosy2 3 2) ++
      timeSignature "4" "4" tsgPosx (htwoStaffs 3 3) (tsgPosy2 3 3);


UpdFunction area s=:{g} =
    (s, EraseRect ++ Score ++ (mkDrawing g));

Quit s io = (s, QuitIO io);

Clef title step = [ SetFont          font4
                  ,MovePenTo        (grid/2,step)
                  ,SetPenMode       OrMode
                  ,DrawString       title
                  ];

timeSignature title1 title2 x y1 y2 =   [ SetFont   font4
                                        ,MovePenTo  (x,y1)
                                        ,DrawString title1
```

```
                        ,MovePenTo  (x,y2)
                        ,SetPenMode OrMode
                        ,DrawString title2
                       ];
/*------------------------ Acoes dos botoes---------------------------*/

dlgPos= DialogPos (Pixel (winX+525)) (Pixel (125+winX));

dialSys= CommandDialog Dialog_id  "Notes & Rests" [dlgPos] 0 bbb;

bbb=
  [ DialogButton quitID  Left          "Quit"    Able  Closit
  , DialogButton delID   (RightTo quitID) "Del "    Able  deleteNote
  , DialogButton saveID  (Below quitID)  "Save"    Able  Closit //saveFile
  , DialogButton midiID  (RightTo saveID) "Midi"    Able  Closit //midiFile
  , CalcButton   bID     (Below saveID)  "W"       font3 1 drawBreve
  , CalcButton   sbID    (RightTo bID)   "w"       font3 1 drawSBreve
  , CalcButton   miID    (Below  bID)    "h"       font  2 drawMinima
  , CalcButton   smID    (RightTo miID)  "q"       font  2 drawSMinima
  , CalcButton   coID    (Below  miID)   "e"       font  2 drawColcheia
  , CalcButton   scID    (RightTo coID)  "x"       font  2 drawSColcheia
  , CalcButton   fuID    (Below coID)    "q"       font  3 drawFusa
  , CalcButton   sfID    (RightTo fuID)  "q"       font  4 drawSFusa
  , CalcButton   pbID    (Below fuID)    "ã"       font  1 drawPBreve
  , CalcButton   psbID   (RightTo pbID)  "î"       font  1 drawPSBreve
  , CalcButton   psmID   (Below pbID)    "Î"       font3 1 drawPSMinima
  , CalcButton   pcoID   (RightTo psmID) "ä"       font2 1 drawPColcheia
  , CalcButton   pscID   (Below psmID)   "Å"       font3 1 drawPSColcheia
  , CalcButton   pfuID   (RightTo pscID) "¨"       font  1 drawPFusa
  , CalcButton   psfID   (Below pscID)   "ô"       font  1 drawPSFusa
  , CalcButton   bemolID (RightTo sbID)  "b"       font3 1 bemois
  , CalcButton   susID   (Below bemolID) "#"       font3 1 sustenidos
  , CalcButton   beqID   (Below susID)   "n"       font3 1 bequadros
  ]
```

```
    where {
        [quitID,delID,saveID,midiID,bID,sbID,miID,smID,coID,scID,fuID,sfID,
            pbID,psbID,pmiID,psmID,pcoID,pscID,pfuID,psfID,bemolID,
            susID,beqID:ids]= [Button_id..];

        };

    fontTpl (k,f)= f;
    font  = fontTpl( SelectFont "Petrucci" [] fontLen);
    font2 = fontTpl( SelectFont "Petrucci" [] (2*fontLen));
    font3 = fontTpl( SelectFont "Petrucci" [] 16);
    font4 = fontTpl( SelectFont "Petrucci" [] (3*fontLen));

    Closit info s io = (s, QuitIO io);

    deleteNote  info s=:{g}  io=
            ( {s&stt= (DEL,NOTHING)}
            , DrawInActiveWindow (ReadFont g ) io
            );

    drawBreve   info s=:{g}  io=
            ( {s&stt= (Breve,NOTHING)}
            , DrawInActiveWindow (ReadFont g) io
            );

    drawSBreve  info s=:{g}  io=
            ( {s&stt= (SBreve,NOTHING)}
            , DrawInActiveWindow (ReadFont g ) io
            );

    drawMinima  info s=:{g}  io=
            ( {s&stt= (Minima,NOTHING)}
            , DrawInActiveWindow (ReadFont g ) io
            );
```

```
drawSMinima info s=:{g}  io=
        ( {s&stt= (SMinima,NOTHING)}
        , DrawInActiveWindow (ReadFont g ) io
        );

drawColcheia info s=:{g}  io=
        ( {s&stt= (Colcheia,NOTHING)}
        , DrawInActiveWindow (ReadFont g ) io
        );

drawSColcheia info s=:{g} io=
        ( {s&stt= (SColcheia,NOTHING)}
        , DrawInActiveWindow (ReadFont g ) io
        );

drawFusa  info  s=:{g}  io=
        ( {s&stt= (Fusa,NOTHING)}
        , DrawInActiveWindow (ReadFont g ) io
        );

drawSFusa  info  s=:{g}  io=
        ( {s&stt= (SFusa,NOTHING)}
        , DrawInActiveWindow (ReadFont g ) io
        );

bemois info s=:{g,stt} io
#  (x,y)=:s.stt =
        ( {s&stt=(x,Bemol)}
        , DrawInActiveWindow (ReadFont g) io
        );

sustenidos info s=:{g,stt} io
#  (x,y)=:s.stt =
```

```
        ({s&stt=(x,Sustenido)}
        , DrawInActiveWindow (ReadFont g) io
        );

bequadros info s=:{g,stt} io
#       (x,y)=:s.stt =
        ( {s&stt= (x,NOTHING)}
        , DrawInActiveWindow (ReadFont g) io
        );

/*-------------------------PAUSAS---------------------*/

drawPBreve  info s=:{g}  io=
        ( {s&stt= (PBreve,NOTHING)}
        , DrawInActiveWindow (ReadFont g) io
        );

drawPSBreve  info s=:{g}  io=
        ( {s&stt= (PSBreve,NOTHING)}
        , DrawInActiveWindow (ReadFont g ) io
        );

drawPSMinima info s=:{g}  io=
        ( {s&stt= (PSMinima,NOTHING)}
        , DrawInActiveWindow (ReadFont g ) io
        );

drawPColcheia info s=:{g}  io=
        ( {s&stt= (PColcheia,NOTHING)}
        , DrawInActiveWindow (ReadFont g ) io
        );

drawPSColcheia info s=:{g} io=
        ( {s&stt= (PSColcheia,NOTHING)}
```

```
        , DrawInActiveWindow (ReadFont g ) io
        );

  drawPFusa  info  s=:{g}  io=
        ( {s&stt= (PFusa,NOTHING)}
        , DrawInActiveWindow (ReadFont g ) io
        );

  drawPSFusa  info  s=:{g}  io=
        ( {s&stt= (PSFusa,NOTHING)}
        , DrawInActiveWindow (ReadFont g ) io
        );

/*------------------------- ReadFont ----------------------------*/

ReadFont bs =
        [ EraseRectangle WindowRect,
         SetPenColour BlackColour] ++ Score ++ (mkDrawing bs);

/*------------------------- Acoes do Mouse --------------------*/

Markit (pMouse, ButtonDown, _) s =: {stt} io =
              case stt of {
                    (NOTHING,NOTHING)    -> (s,io);
      (DEL,NOTHING)      -> delNote      (pMouse) s io;

                (Breve,NOTHING)    -> addBreve     (NotePosition "Bequadro"
                pMouse) s io;
                (SBreve,NOTHING)   -> addSBreve    (NotePosition "Bequadro"
                pMouse) s io;
                (Minima,NOTHING)   -> addMinima    (NotePosition "Bequadro"
                pMouse) s io;
                (SMinima,NOTHING)  -> addSMinima   (NotePosition "Bequadro"
                pMouse) s io;
```

```
(Colcheia,NOTHING) -> addColcheia (NotePosition "Bequadro"
pMouse) s io;
(SColcheia,NOTHING) -> addSColcheia (NotePosition "Bequadro"
pMouse) s io;
(Fusa,NOTHING)     -> addFusa     (NotePosition "Bequadro"
pMouse) s io;
(SFusa,NOTHING)    -> addSFusa    (NotePosition "Bequadro"
pMouse) s io;

(PBreve,NOTHING)   -> addPBreve   (NotePosition "Bequadro"
pMouse) s io;
(PSBreve,NOTHING)  -> addPSBreve  (NotePosition "Bequadro"
pMouse) s io;
(PMinima,NOTHING)  -> addPMinima  (NotePosition "Bequadro"
pMouse) s io;
(PSMinima,NOTHING) -> addPSMinima (NotePosition
"Bequadro" pMouse) s io;
(PColcheia,NOTHING) -> addPColcheia (NotePosition "Bequadro"
pMouse) s io;
(PSColcheia,NOTHING)-> addPSColcheia (NotePosition "Bequadro"
pMouse) s io;
(PFusa,NOTHING)    -> addPFusa    (NotePosition "Bequadro"
pMouse) s io;
(PSFusa,NOTHING)   -> addPSFusa   (NotePosition "Bequadro"
pMouse) s io;

(Breve,Bemol)      -> addbBreve   (NotePosition "Bemol" pMouse) s
io;
(SBreve,Bemol)     -> addbSBreve  (NotePosition "Bemol" pMouse)
s io;
(Minima,Bemol)     -> addbMinima  (NotePosition "Bemol"
pMouse) s io;
(SMinima,Bemol)    -> addbSMinima (NotePosition "Bemol"
pMouse) s io;
```

```
(Colcheia,Bemol)  -> addbColcheia   (NotePosition "Bemol" pMouse)
s io;
(SColcheia,Bemol) -> addbSColcheia  (NotePosition "Bemol"
pMouse) s io;
(Fusa,Bemol)      -> addbFusa       (NotePosition "Bemol" pMouse) s
io;
(SFusa,Bemol)     -> addbSFusa      (NotePosition "Bemol" pMouse) s
io;


(Breve,Sustenido)     -> addSusBreve      (NotePosition "Sustenido"
pMouse) s io;
(SBreve,Sustenido)    -> addSusSBreve     (NotePosition "Sustenido"
pMouse) s io;
(Minima,Sustenido)    -> addSusMinima     (NotePosition "Sustenido"
pMouse) s io;
(SMinima,Sustenido)   -> addSusSMinima    (NotePosition
"Sustenido" pMouse) s io;
(Colcheia,Sustenido)  -> addSusColcheia   (NotePosition "Sustenido"
pMouse) s io;
(SColcheia,Sustenido) -> addSusSColcheia  (NotePosition
"Sustenido" pMouse) s io;
(Fusa,Sustenido)      -> addSusFusa       (NotePosition "Sustenido"
pMouse) s io;
(SFusa,Sustenido)     -> addSusSFusa      (NotePosition "Sustenido"
pMouse) s io;

otherwise     -> (s,io)
};

Markit _  s io  = (s,io);

delNote   point s io=
      let! {nug= eraseNote point s.g}
```

```
        in  ( {s&g= nug, stt=(NOTHING,NOTHING)}
            , DrawInActiveWindow (ReadFont nug) io
            );

eraseNote p [] = [];
eraseNote p [n:r] | p ~=~ getPos(n) = r;
eraseNote p [n:r] = [n: eraseNote p r];

/*------------------------- OPERADORES --------------------------
 ---------------------------- (~!) ----------------------------*/

(~!) infix 5 :: !Int !Int -> Bool ;
(~!) y1 y2 = y1-y2 < h && y2-y1 < h where {h=grid/5};

/*------------------------- Semelhante----------------------------*/

(~=~) infix 5 :: !Point !Point -> Bool ;
(~=~) (x1,y1) (x2,y2) = x1-x2 < h && x2-x1 < h && y1-y2 < h && y2-y1 < h where
{h=grid};


/*------------------------- (!<) ----------------------------*/

(!<) infix 5 :: !(Ritmos, Notas) !(Ritmos, Notas) -> Bool ;
(!<) y1 y2
#   voic1 = whatVoice y1;

    voic2 = whatVoice y2;
=   if (voic1 < voic2) True False;


/*------------------------- (!!<) ----------------------------*/

(!!<) infix 5 :: !(Ritmos, Notas) !(Ritmos, Notas) -> Bool ;
```

```
    (!!<) y1 y2
    #   voc1 = takeCompass y1;
            voc2 = takeCompass y2;
    = if (voc1 < voc2) True False;

    whatCompass v c x y
            |x <= wW/3 = (v,c+1,y);
            |x > wW/3 && x <= 2*wW/3 = (v,c+2,y);
            |x > 2*wW/3 = (v,c+3,y);

    /*---------------------------- NotePosition e Prx ----------------------------*/

    NotePosition acd p=:(x,y) = Prx acd p rPos wSf grid;
    where {
      YPos
            | y <= (htwoStaffs 3 1)-(betweenStaff/2) =
                (whatCompass 1 0 x (htwoStaffs 3 0) );
            | y <= (htwoStaffs 3 2)-(betweenStaff/2) && y > (htwoStaffs 3 1)-
   (betweenStaff/2) =
            (whatCompass 2 0 x (htwoStaffs 3 1) );
            | y <= (htwoStaffs 3 3)-(betweenStaff/2) && y > (htwoStaffs 3 2)-
   (betweenStaff/2) =
            (whatCompass 1 3 x (htwoStaffs 3 2) );
            | y <= (htwoStaffs 3 4)-(betweenStaff/2) && y > (htwoStaffs 3 3)-
   (betweenStaff/2) =
            (whatCompass 2 3 x (htwoStaffs 3 3) );
      | otherwise = abort ("Erro na rotina NotePosition ");

    rPos = tupla_3_3(YPos);
    wSf  = tupla_3_12(YPos);
    };

/* Prx devolve a tupla (posx, posy, Nota) que alem da nota descreve sua posicao.
```

```
exemplo:  (15,45,FSus (5,1,12)), que dizer, (15,45) eh posicao da nota FSus(5,1,12)
sendo que 5 eh a oitava da nota, 1 eh a voz (clave de G) e 12 eh o compasso em que
a nota esta.
*/

Prx :: String Point Int (Int,Int) Int -> (!Int,!Int,!Notas);
Prx acd (x,y) step whatStaff g
        | y ~! (step)     && (acd == "Bequadro") = (x,(step),     (nG6B4 whatStaff));
        | y ~! (step+  g) && (acd == "Bequadro") = (x,(step+  g), (nM6G4 whatStaff));
        | y ~! (step+2*g) && (acd == "Bequadro") = (x,(step+2*g), (nC6M4
 whatStaff));
        | y ~! (step+3*g) && (acd == "Bequadro") = (x,(step+3*g), (nA5C4 whatStaff));
        | y ~! (step+4*g) && (acd == "Bequadro") = (x,(step+4*g), (nF5A3 whatStaff));
        | y ~! (step+5*g) && (acd == "Bequadro") = (x,(step+4*g), (nD5F3 whatStaff));


        | y ~! ( step     +g/2) && (acd == "Bequadro")  = (x,( step+     g/2), (nF6A4
 whatStaff));
        | y ~! ((step+  g)+g/2) && (acd == "Bequadro")  = (x,((step+  g)+g/2), (nD6F4
 whatStaff));
        | y ~! ((step+2*g)+g/2) && (acd == "Bequadro")  = (x,((step+2*g)+g/2),
(nB5D4 whatStaff));
        | y ~! ((step+3*g)+g/2) && (acd == "Bequadro")  = (x,((step+3*g)+g/2),
(nG5B3 whatStaff));
        | y ~! ((step+4*g)+g/2) && (acd == "Bequadro")  = (x,((step+4*g)+g/2),
(nM5G3 whatStaff));


        | y ~! (step)     && (acd == "Sustenido") = (x,(step),     (nGss6Bss5 whatStaff));
        | y ~! (step+  g) && (acd == "Sustenido") = (x,(step+  g), (nMss6Gss4
whatStaff));
        | y ~! (step+2*g) && (acd == "Sustenido") = (x,(step+2*g), (nCss6Mss4
whatStaff));
        | y ~! (step+3*g) && (acd == "Sustenido") = (x,(step+3*g), (nAss5Css4
whatStaff));
```

```
        | y ~! (step+4*g) && (acd == "Sustenido") = (x,(step+4*g), (nFss5Ass3
    whatStaff));
        | y ~! (step+5*g) && (acd == "Sustenido") = (x,(step+5*g), (nDss5Fss3
    whatStaff));

        | y ~! (step+     g/2) && (acd == "Sustenido")=(x,(step+     g/2),(nFss6Ass4
    whatStaff));
        | y ~! ((step+  g)+g/2) && (acd == "Sustenido")=(x,((step+  g)+g/2),(nDss6Fss4
    whatStaff));
        | y ~! ((step+2*g)+g/2) && (acd ==
    "Sustenido")=(x,((step+2*g)+g/2),(nBss5Dss4 whatStaff));
        | y ~! ((step+3*g)+g/2) && (acd ==
    "Sustenido")=(x,((step+3*g)+g/2),(nGss5Bss4 whatStaff));
        | y ~! ((step+4*g)+g/2) && (acd ==
    "Sustenido")=(x,((step+4*g)+g/2),(nMss5Gss3 whatStaff));

        | y ~! (step)     && (acd == "Bemol") = (x,(step),     (nGb6Bb4 whatStaff));
        | y ~! (step+  g) && (acd == "Bemol") = (x,(step+  g), (nMb6Gb4 whatStaff));
        | y ~! (step+2*g) && (acd == "Bemol") = (x,(step+2*g), (nCb5Mb4 whatStaff));
        | y ~! (step+3*g) && (acd == "Bemol") = (x,(step+3*g), (nAb5Cb3 whatStaff));
        | y ~! (step+4*g) && (acd == "Bemol") = (x,(step+4*g), (nFb5Ab3 whatStaff));
        | y ~! (step+5*g) && (acd == "Bemol") = (x,(step+5*g), (nDb5Fb3 whatStaff));

        | y ~! (step+     g/2) && (acd == "Bemol")  = (x,(step+ g/2),     (nFb6Ab4
    whatStaff));
        | y ~! ((step+  g)+g/2)&& (acd == "Bemol")  = (x,((step+  g)+g/2),(nDb6Fb4
    whatStaff));
        | y ~! ((step+2*g)+g/2)&& (acd == "Bemol")  = (x,((step+2*g)+g/2),(nBb5Db4
    whatStaff));
        | y ~! ((step+3*g)+g/2)&& (acd == "Bemol")  = (x,((step+3*g)+g/2),(nGb5Bb3
    whatStaff));
        | y ~! ((step+4*g)+g/2)&& (acd == "Bemol")  = (x,((step+4*g)+g/2),(nMb5Gb3
    whatStaff));
```

```
    | otherwise = Prx acd (x,y+1) step whatStaff g;

 nG6B4 wStaff = if (tupla_2_1(wStaff)==1) (G (6,1,tupla_2_2(wStaff))) (B
 (4,2,tupla_2_2(wStaff)));
 nF6A4 wStaff = if (tupla_2_1(wStaff)==1) (F (6,1,tupla_2_2(wStaff))) (A
 (4,2,tupla_2_2(wStaff)));
 nM6G4 wStaff = if (tupla_2_1(wStaff)==1) (M (6,1,tupla_2_2(wStaff))) (G
 (4,2,tupla_2_2(wStaff)));
 nD6F4 wStaff = if (tupla_2_1(wStaff)==1) (D (6,1,tupla_2_2(wStaff))) (F
 (4,2,tupla_2_2(wStaff)));
 nC6M4 wStaff = if (tupla_2_1(wStaff)==1) (Cc (6,1,tupla_2_2(wStaff))) (M
 (4,2,tupla_2_2(wStaff)));
 nB5D4 wStaff = if (tupla_2_1(wStaff)==1) (B (5,1,tupla_2_2(wStaff))) (D
 (4,2,tupla_2_2(wStaff)));
 nA5C4 wStaff = if (tupla_2_1(wStaff)==1) (A (5,1,tupla_2_2(wStaff))) (Cc
 (4,2,tupla_2_2(wStaff)));
 nG5B3 wStaff = if (tupla_2_1(wStaff)==1) (G (5,1,tupla_2_2(wStaff))) (B
 (4,2,tupla_2_2(wStaff)));
 nF5A3 wStaff = if (tupla_2_1(wStaff)==1) (F (5,1,tupla_2_2(wStaff))) (A
 (3,2,tupla_2_2(wStaff)));
 nM5G3 wStaff = if (tupla_2_1(wStaff)==1) (M (5,1,tupla_2_2(wStaff))) (G
 (3,2,tupla_2_2(wStaff)));
 nD5F3 wStaff = if (tupla_2_1(wStaff)==1) (D (5,1,tupla_2_2(wStaff))) (F
 (3,2,tupla_2_2(wStaff)));

 nGss6Bss5 wStaff = if (tupla_2_1(wStaff)==1) (Gsus (6,1,tupla_2_2(wStaff))) (Bsus
 (5,2,tupla_2_2(wStaff)));
 nFss6Ass4 wStaff = if (tupla_2_1(wStaff)==1) (Fsus (6,1,tupla_2_2(wStaff))) (Asus
 (4,2,tupla_2_2(wStaff)));
 nMss6Gss4 wStaff = if (tupla_2_1(wStaff)==1) (Msus (6,1,tupla_2_2(wStaff))) (Gsus
 (4,2,tupla_2_2(wStaff)));
 nDss6Fss4 wStaff = if (tupla_2_1(wStaff)==1) (Dsus (6,1,tupla_2_2(wStaff))) (Fsus
 (4,2,tupla_2_2(wStaff)));
```

```
nCss6Mss4 wStaff = if (tupla_2_1(wStaff)==1) (Csus (6,1,tupla_2_2(wStaff))) (Msus
(4,2,tupla_2_2(wStaff)));
nBss5Dss4 wStaff = if (tupla_2_1(wStaff)==1) (Bsus (6,1,tupla_2_2(wStaff))) (Dsus
(4,2,tupla_2_2(wStaff)));
nAss5Css4 wStaff = if (tupla_2_1(wStaff)==1) (Asus (5,1,tupla_2_2(wStaff))) (Csus
(4,2,tupla_2_2(wStaff)));
nGss5Bss4 wStaff = if (tupla_2_1(wStaff)==1) (Gsus (5,1,tupla_2_2(wStaff))) (Bsus
(4,2,tupla_2_2(wStaff)));
nFss5Ass3 wStaff = if (tupla_2_1(wStaff)==1) (Fsus (5,1,tupla_2_2(wStaff))) (Asus
(3,2,tupla_2_2(wStaff)));
nMss5Gss3 wStaff = if (tupla_2_1(wStaff)==1) (Msus (5,1,tupla_2_2(wStaff))) (Gsus
(3,2,tupla_2_2(wStaff)));
nDss5Fss3 wStaff = if (tupla_2_1(wStaff)==1) (Dsus (5,1,tupla_2_2(wStaff))) (Fsus
(3,2,tupla_2_2(wStaff)));

nGb6Bb4 wStaff = if (tupla_2_1(wStaff)==1) (Gb (6,1,tupla_2_2(wStaff))) (Bb
(4,2,tupla_2_2(wStaff)));
nFb6Ab4 wStaff = if (tupla_2_1(wStaff)==1) (Fb (6,1,tupla_2_2(wStaff))) (Ab
(4,2,tupla_2_2(wStaff)));
nMb6Gb4 wStaff = if (tupla_2_1(wStaff)==1) (Mb (6,1,tupla_2_2(wStaff))) (Gb
(4,2,tupla_2_2(wStaff)));
nDb6Fb4 wStaff = if (tupla_2_1(wStaff)==1) (Db (6,1,tupla_2_2(wStaff))) (Fb
(4,2,tupla_2_2(wStaff)));
nCb5Mb4 wStaff = if (tupla_2_1(wStaff)==1) (Cb (5,1,tupla_2_2(wStaff))) (Mb
(4,2,tupla_2_2(wStaff)));
nBb5Db4 wStaff = if (tupla_2_1(wStaff)==1) (Bb (5,1,tupla_2_2(wStaff))) (Db
(4,2,tupla_2_2(wStaff)));
nAb5Cb3 wStaff = if (tupla_2_1(wStaff)==1) (Ab (5,1,tupla_2_2(wStaff))) (Cb
(3,2,tupla_2_2(wStaff)));
nGb5Bb3 wStaff = if (tupla_2_1(wStaff)==1) (Gb (5,1,tupla_2_2(wStaff))) (Bb
(3,2,tupla_2_2(wStaff)));
nFb5Ab3 wStaff = if (tupla_2_1(wStaff)==1) (Fb (5,1,tupla_2_2(wStaff))) (Ab
(3,2,tupla_2_2(wStaff)));
```

```
nMb5Gb3 wStaff = if (tupla_2_1(wStaff)==1) (Mb (5,1,tupla_2_2(wStaff))) (Gb
(3,2,tupla_2_2(wStaff)));
nDb5Fb3 wStaff = if (tupla_2_1(wStaff)==1) (Db (5,1,tupla_2_2(wStaff))) (Fb
(3,2,tupla_2_2(wStaff)));

/*---------- TUPLAS ----------*/

tupla_3_12 (x,y,n) = (x,y);
tupla_3_3  (x,y,n) = n;

tupla_2_1::(!Int,!Int) -> Int;
tupla_2_1  (x,y)  = x;
tupla_2_2  (x,y)  = y;

/*---------- Notas Bequadro ----------*/
addBreve point s io =
        let!  {nug= [( Br (tupla_3_12(point)),tupla_3_3 (point)  ):s.g]}
        in    ( {s&g= nug, stt= (NOTHING,NOTHING)}
              , DrawInActiveWindow (mkDrawing nug) io
              );

addSBreve point s io =
        let!  {nug= [ (SB (tupla_3_12(point)),tupla_3_3 (point)  ):s.g]}
        in    ( {s&g= nug, stt= (NOTHING,NOTHING)}
              , DrawInActiveWindow (mkDrawing nug) io
              );

addMinima point s io =
        let!  {nug= [ (Mi (tupla_3_12(point)),tupla_3_3 (point)  ):s.g]}
        in    ( {s&g= nug, stt= (NOTHING,NOTHING)}
              , DrawInActiveWindow (mkDrawing nug) io
              );

addSMinima point s io =
```

```
        let! {nug= [ (SM (tupla_3_12(point)),tupla_3_3 (point) ):s.g]}
        in   ( {s&g= nug, stt= (NOTHING,NOTHING)}
             , DrawInActiveWindow (mkDrawing nug) io
             );

addColcheia point s io =
        let! {nug= [ (Co (tupla_3_12(point)),tupla_3_3 (point) ):s.g]}
        in   ( {s&g= nug, stt= (NOTHING,NOTHING)}
             , DrawInActiveWindow (mkDrawing nug) io
             );

addSColcheia point s io =
        let! {nug= [ (SC (tupla_3_12(point)),tupla_3_3 (point) ):s.g]}
        in   ( {s&g= nug, stt= (NOTHING,NOTHING)}
             , DrawInActiveWindow (mkDrawing nug) io
             );

addFusa point s io =
        let! {nug= [ (Fu (tupla_3_12(point)),tupla_3_3 (point) ):s.g]}
        in   ( {s&g= nug, stt= (NOTHING,NOTHING)}
             , DrawInActiveWindow (mkDrawing nug) io
             );

addSFusa point s io =
        let! {nug= [ (SF (tupla_3_12(point)),tupla_3_3 (point) ):s.g]}
        in   ( {s&g= nug, stt= (NOTHING,NOTHING)}
             , DrawInActiveWindow (mkDrawing nug) io
             );

/*---------------------- Notas Bemois ----------------------*/

addbBreve point s io =
        let! {nug= [( BeBr (tupla_3_12(point)),tupla_3_3 (point) ):s.g]}
        in   ( {s&g= nug, stt= (NOTHING,NOTHING)}
```

```
        , DrawInActiveWindow (mkDrawing nug) io
        );

addbSBreve point s io =
        let! {nug= [ (BeSB (tupla_3_12(point)),tupla_3_3 (point) ):s.g]}
        in  ( {s&g= nug, stt= (NOTHING,NOTHING)}
            , DrawInActiveWindow (mkDrawing nug) io
            );

addbMinima point s io =
        let! {nug= [ (BeMi (tupla_3_12(point)),tupla_3_3 (point) ):s.g]}
        in  ( {s&g= nug, stt= (NOTHING,NOTHING)}
            , DrawInActiveWindow (mkDrawing nug) io
            );

addbSMinima point s io =
        let! {nug= [ (BeSM (tupla_3_12(point)),tupla_3_3 (point) ):s.g]}
        in  ( {s&g= nug, stt= (NOTHING,NOTHING)}
            , DrawInActiveWindow (mkDrawing nug) io
            );

addbColcheia point s io =
        let! {nug= [ (BeCo (tupla_3_12(point)),tupla_3_3 (point) ):s.g]}
        in  ( {s&g= nug, stt= (NOTHING,NOTHING)}
            , DrawInActiveWindow (mkDrawing nug) io
            );

addbSColcheia point s io =
        let! {nug= [ (BeSC (tupla_3_12(point)),tupla_3_3 (point) ):s.g]}
        in  ( {s&g= nug, stt= (NOTHING,NOTHING)}
            , DrawInActiveWindow (mkDrawing nug) io
            );

addbFusa point s io =
```

```
        let!  {nug= [ (BeFu (tupla_3_12(point)),tupla_3_3 (point)  ):s.g]}
        in    ( {s&g= nug, stt= (NOTHING,NOTHING)}
              , DrawInActiveWindow (mkDrawing nug) io
              );

addbSFusa point s io =
        let!  {nug= [ (BeSF (tupla_3_12(point)),tupla_3_3 (point)  ):s.g]}
        in    ( {s&g= nug, stt= (NOTHING,NOTHING)}
              , DrawInActiveWindow (mkDrawing nug) io
              );

/*---------------------- Notas Sustenidas ----------------------*/

addSusBreve point s io =
        let!  {nug= [( SusBr (tupla_3_12(point)),tupla_3_3 (point)  ):s.g]}
        in    ( {s&g= nug, stt= (NOTHING,NOTHING)}
              , DrawInActiveWindow (mkDrawing nug) io
              );

addSusSBreve point s io =
        let!  {nug= [ (SusSB (tupla_3_12(point)),tupla_3_3 (point)  ):s.g]}
        in    ( {s&g= nug, stt= (NOTHING,NOTHING)}
              , DrawInActiveWindow (mkDrawing nug) io
              );

addSusMinima point s io =
        let!  {nug= [ (SusMi (tupla_3_12(point)),tupla_3_3 (point)  ):s.g]}
        in    ( {s&g= nug, stt= (NOTHING,NOTHING)}
              , DrawInActiveWindow (mkDrawing nug) io
              );

addSusSMinima point s io =
        let!  {nug= [ (SusSM (tupla_3_12(point)),tupla_3_3 (point)  ):s.g]}
        in    ( {s&g= nug, stt= (NOTHING,NOTHING)}
```

```
            , DrawInActiveWindow (mkDrawing nug) io
            );

    addSusColcheia point s io =
        let! {nug= [ (SusCo (tupla_3_12(point)),tupla_3_3 (point) ):s.g]}
        in  ( {s&g= nug, stt= (NOTHING,NOTHING)}

            , DrawInActiveWindow (mkDrawing nug) io
            );

    addSusSColcheia point s io =
        let! {nug= [ (SusSC (tupla_3_12(point)),tupla_3_3 (point) ):s.g]}
        in  ( {s&g= nug, stt= (NOTHING,NOTHING)}
            , DrawInActiveWindow (mkDrawing nug) io
            );

    addSusFusa point s io =
        let! {nug= [ (SusFu (tupla_3_12(point)),tupla_3_3 (point) ):s.g]}
        in  ( {s&g= nug, stt= (NOTHING,NOTHING)}
            , DrawInActiveWindow (mkDrawing nug) io
            );

    addSusSFusa point s io =
        let! {nug= [ (SusSF (tupla_3_12(point)),tupla_3_3 (point) ):s.g]}
        in  ( {s&g= nug, stt= (NOTHING,NOTHING)}
            , DrawInActiveWindow (mkDrawing nug) io
            );

/*-------------------- PAUSAS ----------------------*/

addPBreve point s io =
        let! {nug= [(PB (tupla_3_12(point)), NO) :s.g]}
        in  ( {s&g= nug, stt= (NOTHING,NOTHING)}
            , DrawInActiveWindow (mkDrawing nug) io
```

```
        );

addPSBreve point s io =
        let!  {nug= [(PSB (tupla_3_12(point)),NO) :s.g]}
        in    ( {s&g= nug, stt= (NOTHING,NOTHING)}
              , DrawInActiveWindow (mkDrawing nug) io
              );

addPMinima point s io =
        let!  {nug= [ (PMi (tupla_3_12(point)), NO):s.g]}
        in    ( {s&g= nug, stt= (NOTHING,NOTHING)}
              , DrawInActiveWindow (mkDrawing nug) io
              );

addPSMinima point s io =
        let!  {nug= [ (PSM (tupla_3_12(point)), NO):s.g]}
        in    ( {s&g= nug, stt= (NOTHING,NOTHING)}
              , DrawInActiveWindow (mkDrawing nug) io
              );

addPColcheia point s io =
        let!  {nug= [ (PCo (tupla_3_12(point)),NO):s.g]}
        in    ( {s&g= nug, stt= (NOTHING,NOTHING)}
              , DrawInActiveWindow (mkDrawing nug) io
              );

addPSColcheia point s io =
        let!  {nug= [ (PSC (tupla_3_12(point)), NO):s.g]}
        in    ( {s&g= nug, stt= (NOTHING,NOTHING)}
              , DrawInActiveWindow (mkDrawing nug) io
              );

addPFusa point s io =
        let!  {nug= [ (PFu (tupla_3_12(point)), NO):s.g]}
```

```
        in   ( {s&g= nug, stt= (NOTHING,NOTHING)}
             , DrawInActiveWindow (mkDrawing nug) io
             );

addPSFusa point s io =
        let! {nug= [ (PSF (tupla_3_12(point)), NO):s.g]}
        in   ( {s&g= nug, stt= (NOTHING,NOTHING)}
             , DrawInActiveWindow (mkDrawing nug) io
             );

EraseRect=
        [  SetPenColour BlackColour// ou (RGB 0.0 0.0 0.8)
        ,  EraseRectangle  WindowRect  // apaga toda partitura: linhas, notas, tudo!
        ];

mkDrawing []=[];
mkDrawing [note:r]=
                [       SetFont                 font2
                ,       MovePenTo       (getPos note) // devolve tipo "Point"
                ,       SetPenMode      OrMode
                ,       DrawString      (NoteSym note) // devolve a string ritmo da
nota.
                : mkDrawing r
                ];


/*---------------------------- CalcButton ----------------------------*/

CalcButton :: DialogItemId ItemPos ItemTitle Font Int (ButtonFunction *State IO) ->
                                DialogItem    *State IO;
CalcButton id pos title font type bfunc
        = DialogIconButton id pos buttonDomain (look type) Able bfunc;
where
        {
```

```
buttonWidth   = 30;
buttonHeight   = 30;
buttonDomain   = ((0,0),(buttonWidth,buttonHeight));
bhf          = (buttonHeight-fontLen);
x            = (buttonWidth-FontStringWidth title font)/2-1;
look :: Int SelectState -> [DrawFunction];
look 1 _
      = [    SetFont                    font
        ,    FillRectangle  shadowrect
        ,    SetPenColour  WhiteColour //ou (RGB 0.5 0.5 0.5)
        ,    FillRectangle  buttonrect
        ,    SetPenColour BlackColour
        ,    DrawRectangle      buttonrect
        ,    MovePenTo          (x,bhf)
        ,    SetPenMode         OrMode
        ,    DrawString         title
        ];
look 2 _
      = [    SetFont                    font
        ,    FillRectangle  shadowrect
        ,    SetPenColour  WhiteColour //ou (RGB 0.5 0.5 0.5)
        ,    FillRectangle  buttonrect
        ,    SetPenColour BlackColour
        ,    DrawRectangle      buttonrect
        ,    MovePenTo          (x,bhf+5)
        ,    SetPenMode         OrMode
        ,    DrawString         title
        ];

look 3 _
      = [    SetFont                    font
        ,    FillRectangle  shadowrect
        ,    SetPenColour  WhiteColour //ou (RGB 0.5 0.5 0.5)
        ,    FillRectangle  buttonrect
```

```
                ,       SetPenColour BlackColour
                ,       DrawRectangle       buttonrect
                ,       MovePenTo           (x,bhf+5)
                ,       SetPenMode          OrMode
                ,       DrawString          title
                ,       MovePenTo           (x+(x/2)-1,x/2)
               , DrawString     "r"
    ,MovePenTo          (x+(x/2)-1,x+2)
               , DrawString     "K"
               ];

    look 4 _
           =[     SetFont                      font
            ,     FillRectangle  shadowrect
            ,     SetPenColour WhiteColour //ou (RGB 0.5 0.5 0.5)
            ,     FillRectangle  buttonrect
            ,     SetPenColour BlackColour
            ,     DrawRectangle       buttonrect
            ,     MovePenTo           (x,bhf+5)
            ,     SetPenMode          OrMode
            ,     DrawString          title
            ,     MovePenTo           (x+(x/2)-1,x/2)
           , DrawString     "r"
,MovePenTo          (x+(x/2)-1,x+2)
           , DrawString     "r"
           ];

shadowrect     = ((2,2),(buttonWidth,buttonHeight));
buttonrect     = ((0,0),(buttonWidth-2,buttonHeight-2));

};
```

APÊNDICE I

DIAGRAMAS

Relação dos módulos

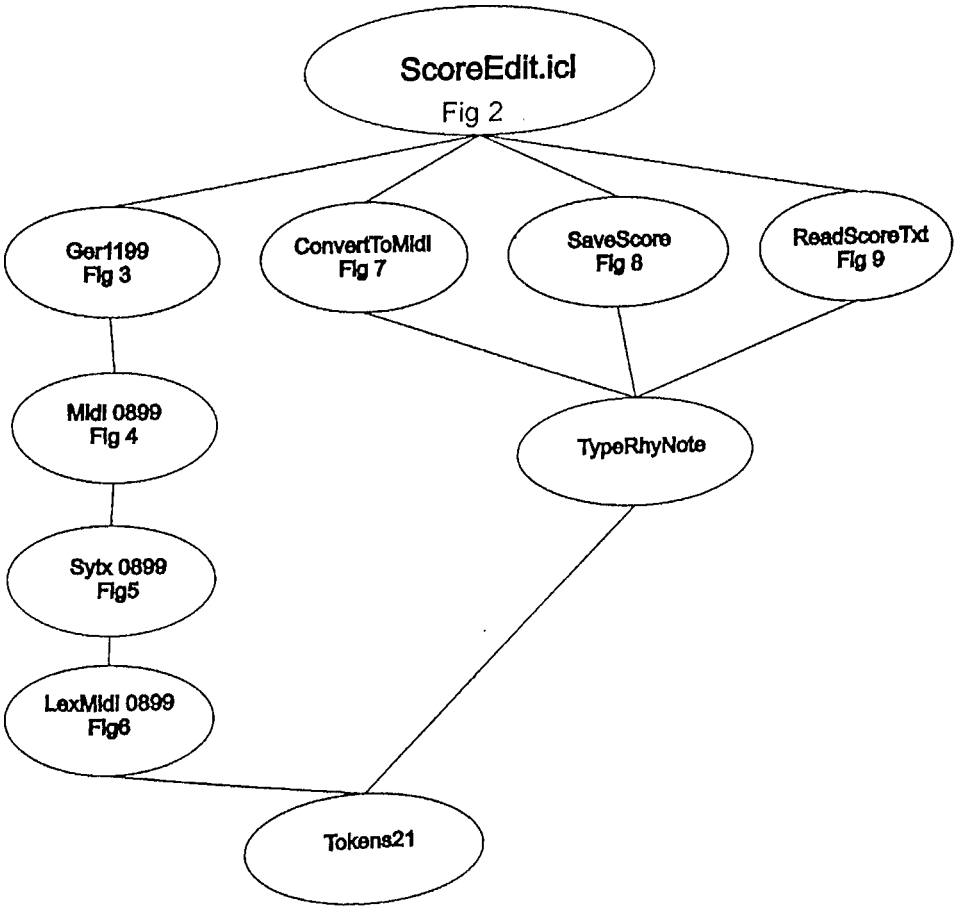


Figura 01

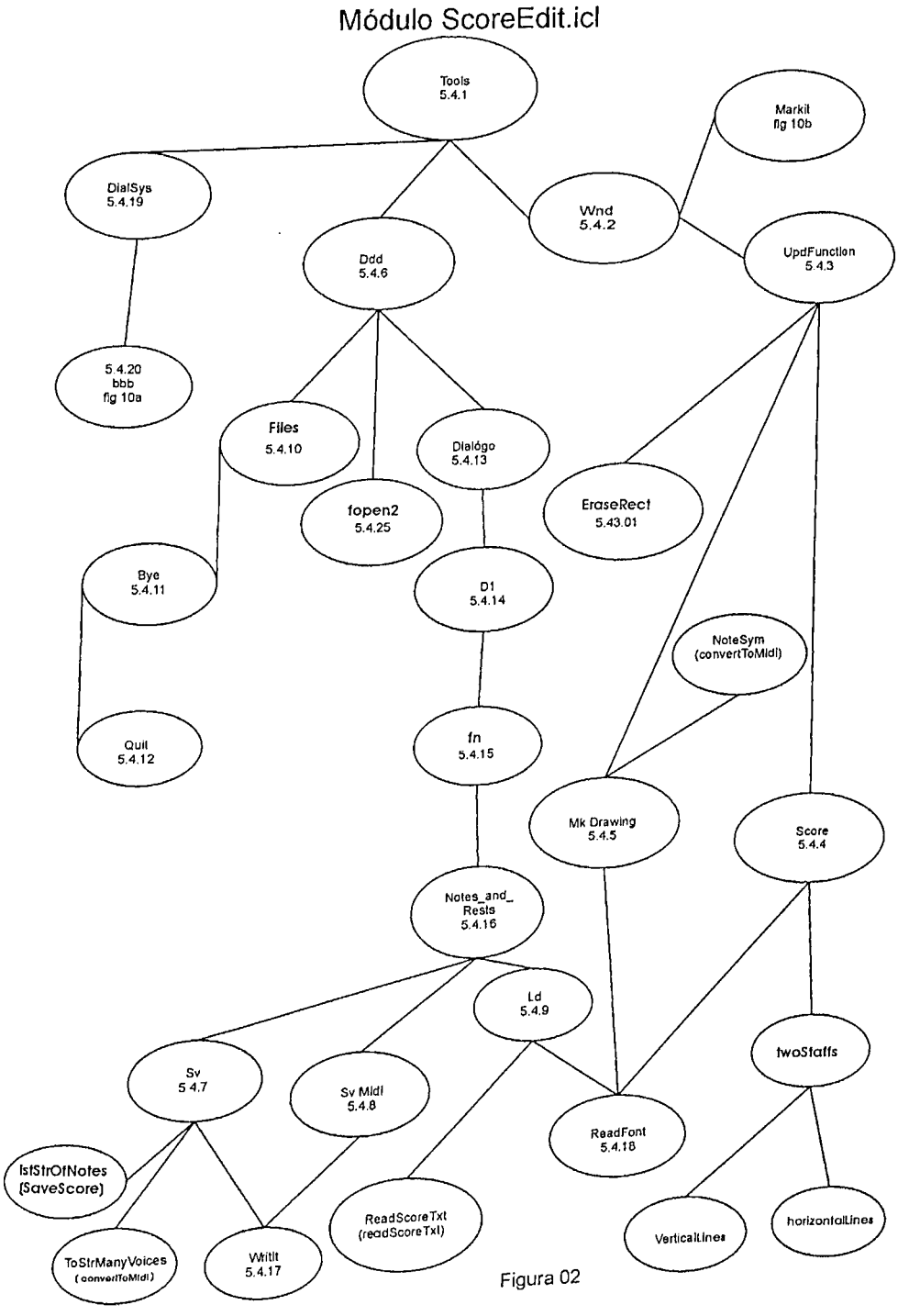

Módulo ScoreEdit.icl
Tools
5.4.1
Markit
fig 10b
DialSys
5.4.19
Wnd
5.4.2
Ddd
5.4.6
UpdFunction
5.4.3
5.4.20
bbb
fig 10a
Files
5.4.10
Dialógo
5.4.13
fopen2
5.4.25
EraseRect
5.43.01
Bye
5.4.11
D1
5.4.14
NoteSym
(convertToMidi)
Quit
5.4.12
fn
5.4.15
Mk Drawing
5.4.5
Score
5.4.4
Notes_and_
Rests
5.4.16
Ld
5.4.9
twoStaffs
Sv
5.4.7
Sv Midi
5.4.8
ReadFont
5.4.18
IstStrOfNotes
(SaveScore)
ReadScoreTxt
(readScoreTxt)
VerticalLines
horizontalLines
ToStrManyVoices
(convertToMidi)
WritIt
5.4.17

Figura 02

Módulo ger1199

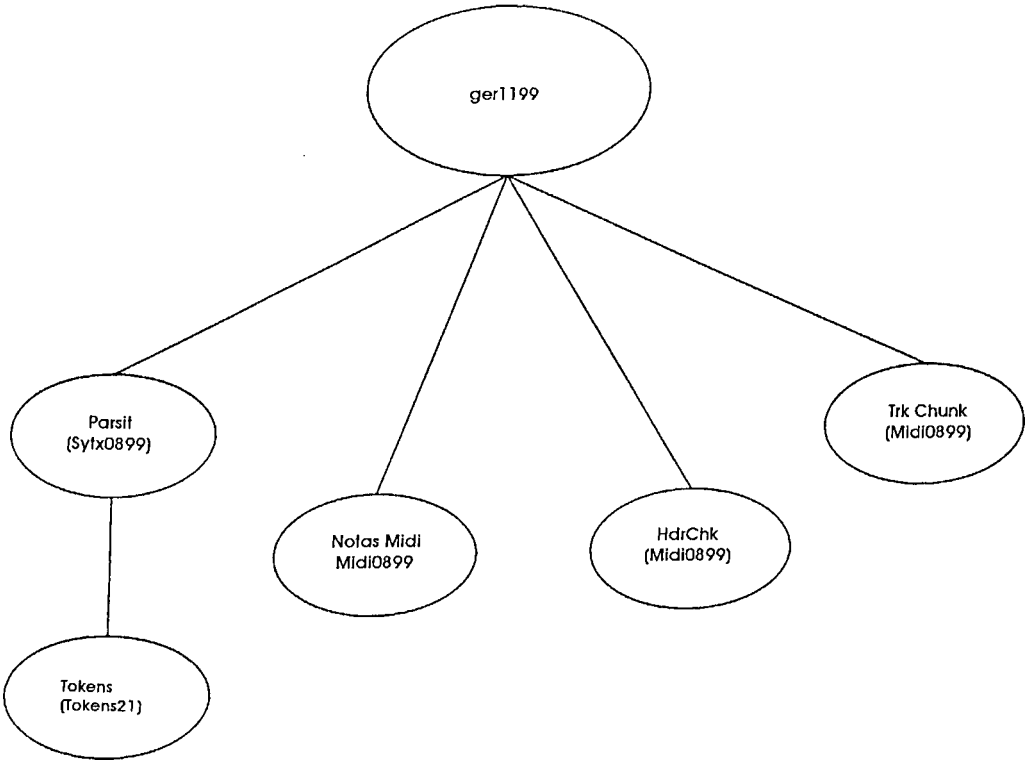


Figura 03

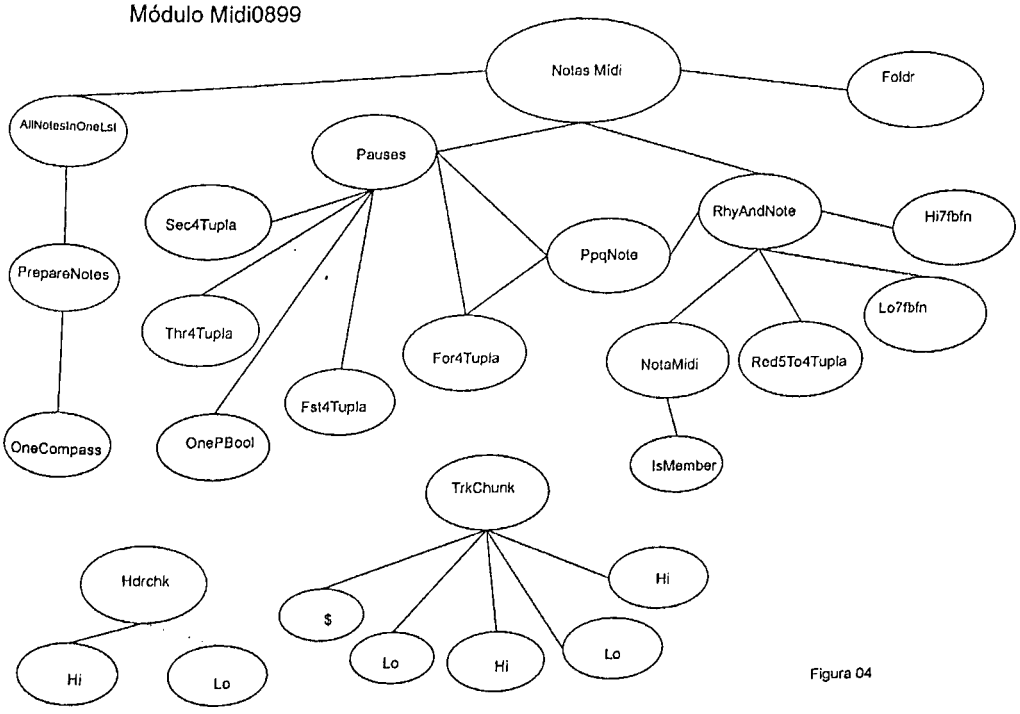

Módulo Midi0899
Notas Midi
Foldr
AllNotesInOneLst
Pauses
Sec4Tupla
PrepareNotes
PpqNote
RhyAndNote
Hi7fbfn
Lo7fbfn
Thr4Tupla
For4Tupla
NotaMidi
Red5To4Tupla
Fst4Tupla
OneCompass
OnePBool
IsMember
TrkChunk
Hdrchk
Hi
$
Lo
Hi
Lo
Hi
Lo

Figura 04

Módulo Sytx 0899

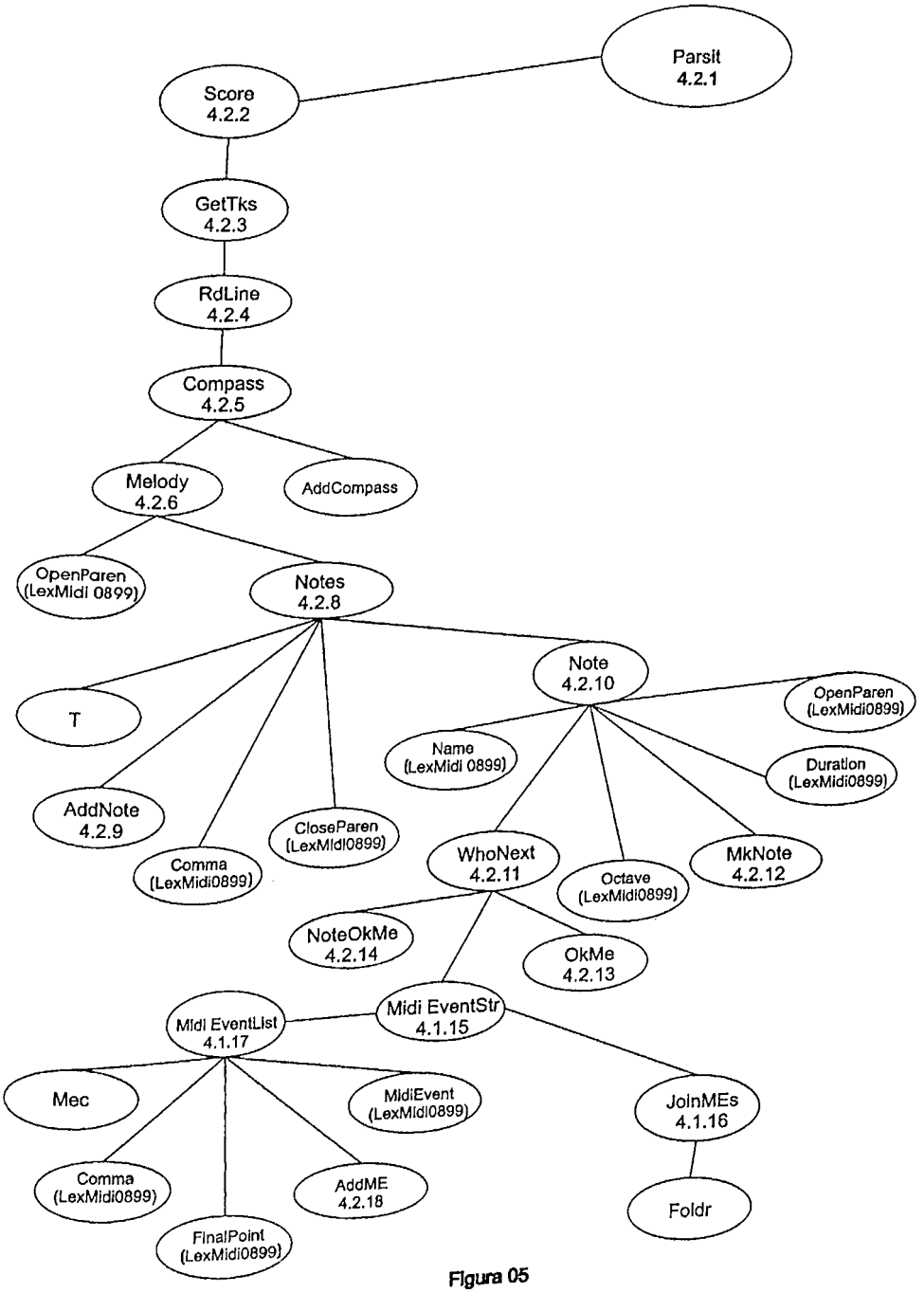


Figura 05

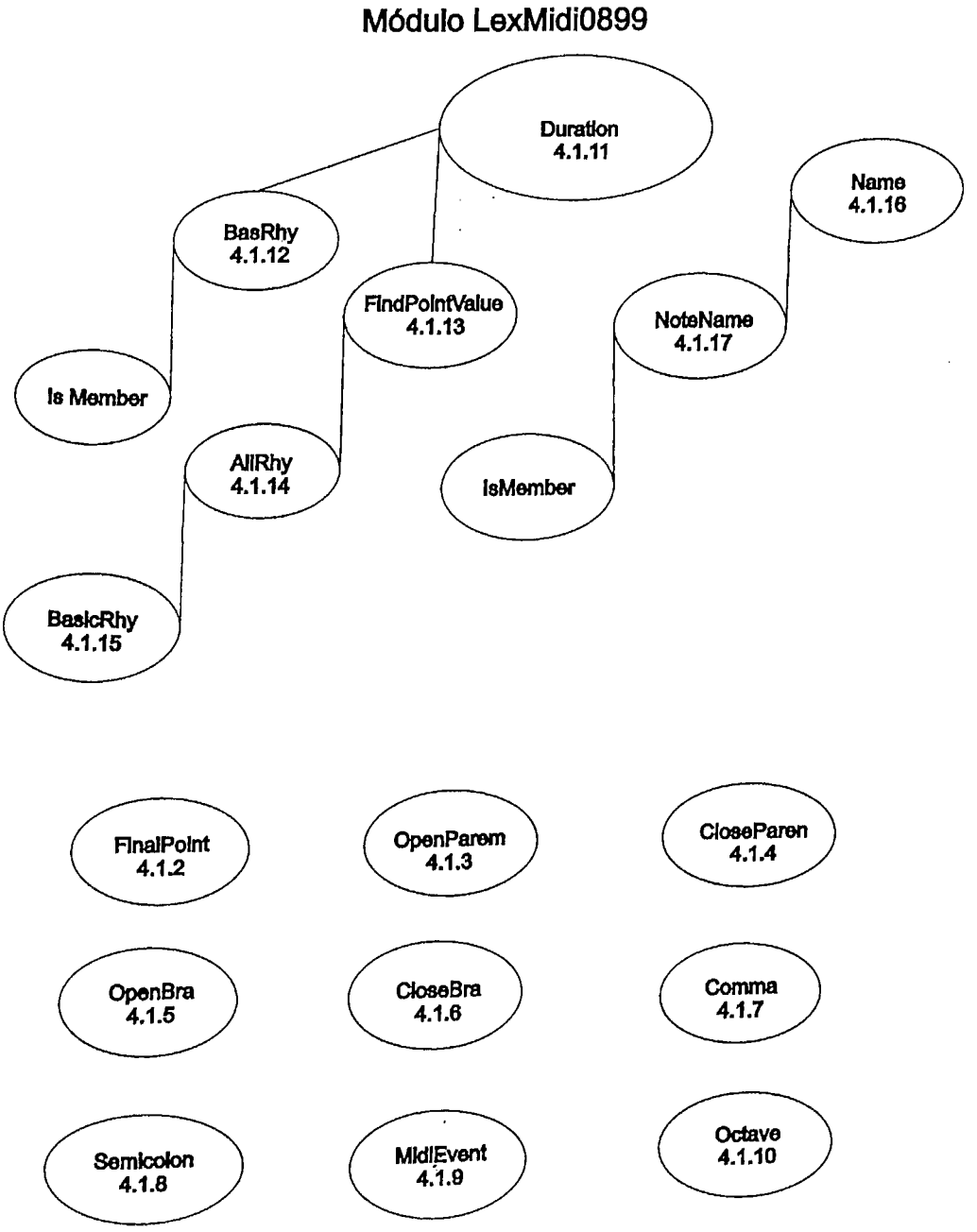

Módulo LexMidi0899
Duration
4.1.11
Name
4.1.16
BasRhy
4.1.12
FindPointValue
4.1.13
NoteName
4.1.17
Is Member
AllRhy
4.1.14
IsMember
BasicRhy
4.1.15
FinalPoint
4.1.2
OpenParem
4.1.3
CloseParen
4.1.4
OpenBra
4.1.5
CloseBra
4.1.6
Comma
4.1.7
Semicolon
4.1.8
MidiEvent
4.1.9
Octave
4.1.10

Figura 06

Módulo ConvertToMidi

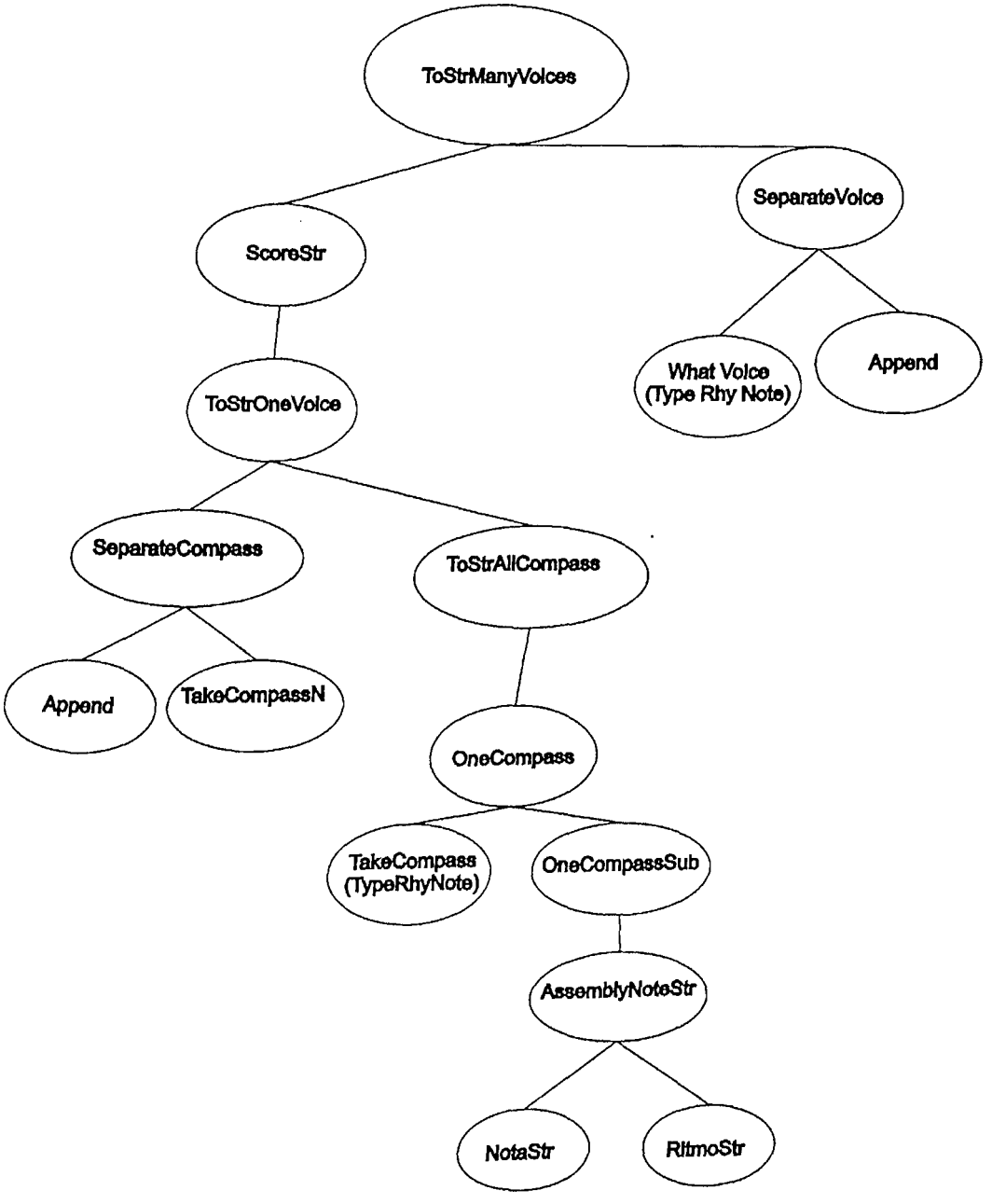


Figura 07

Módulo saveScore

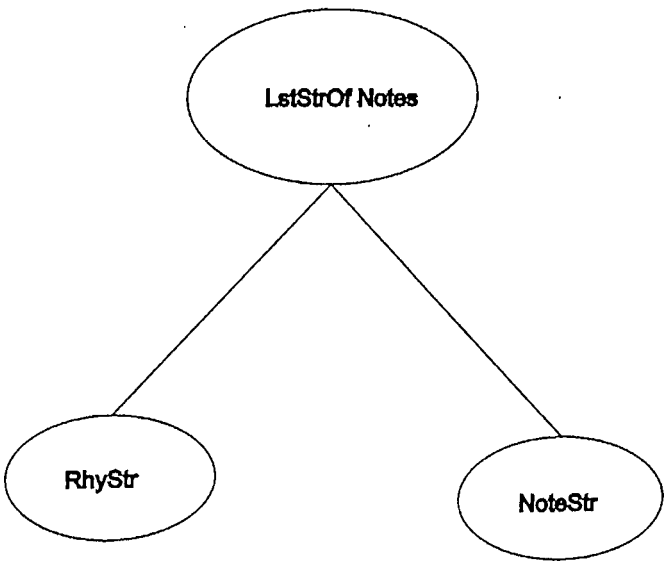


Figura 08

Módulo ReadScoretxt

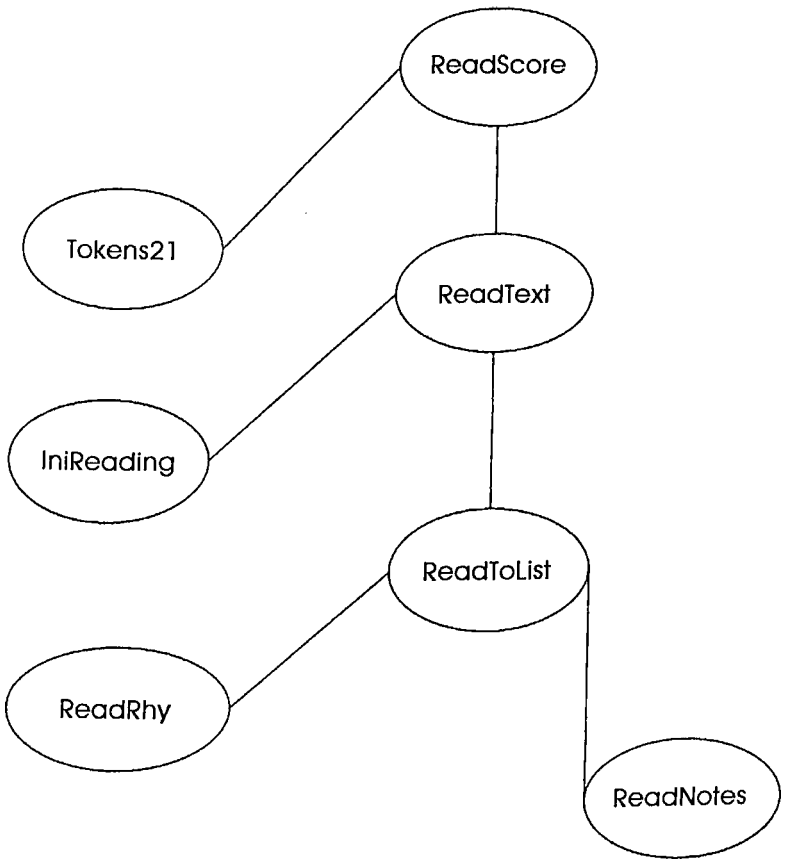


Figura 9

Função bbb (ScoreEdit)

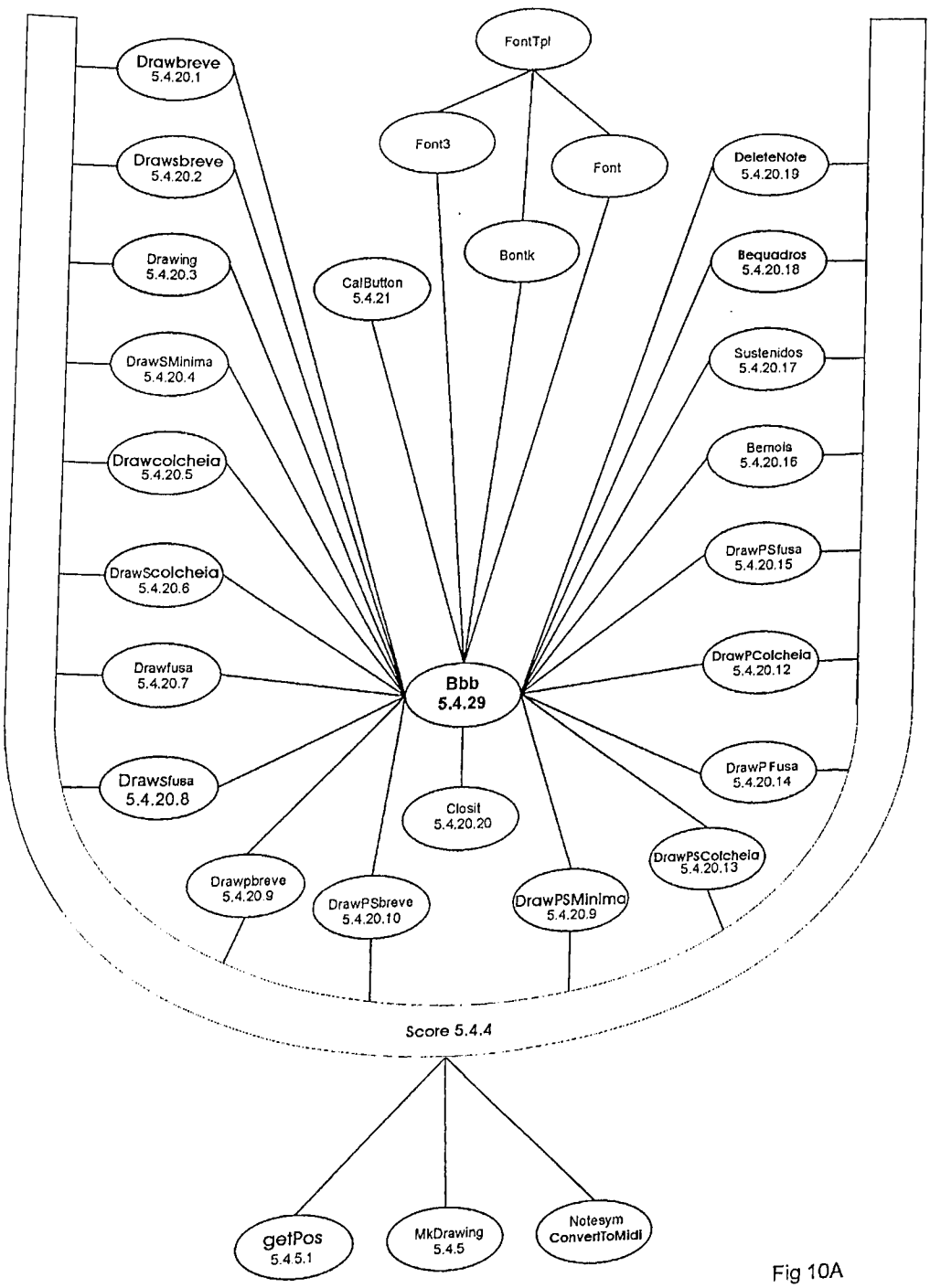


Fig 10A

Função Markit (ScoreEdit) - Ações do mouse

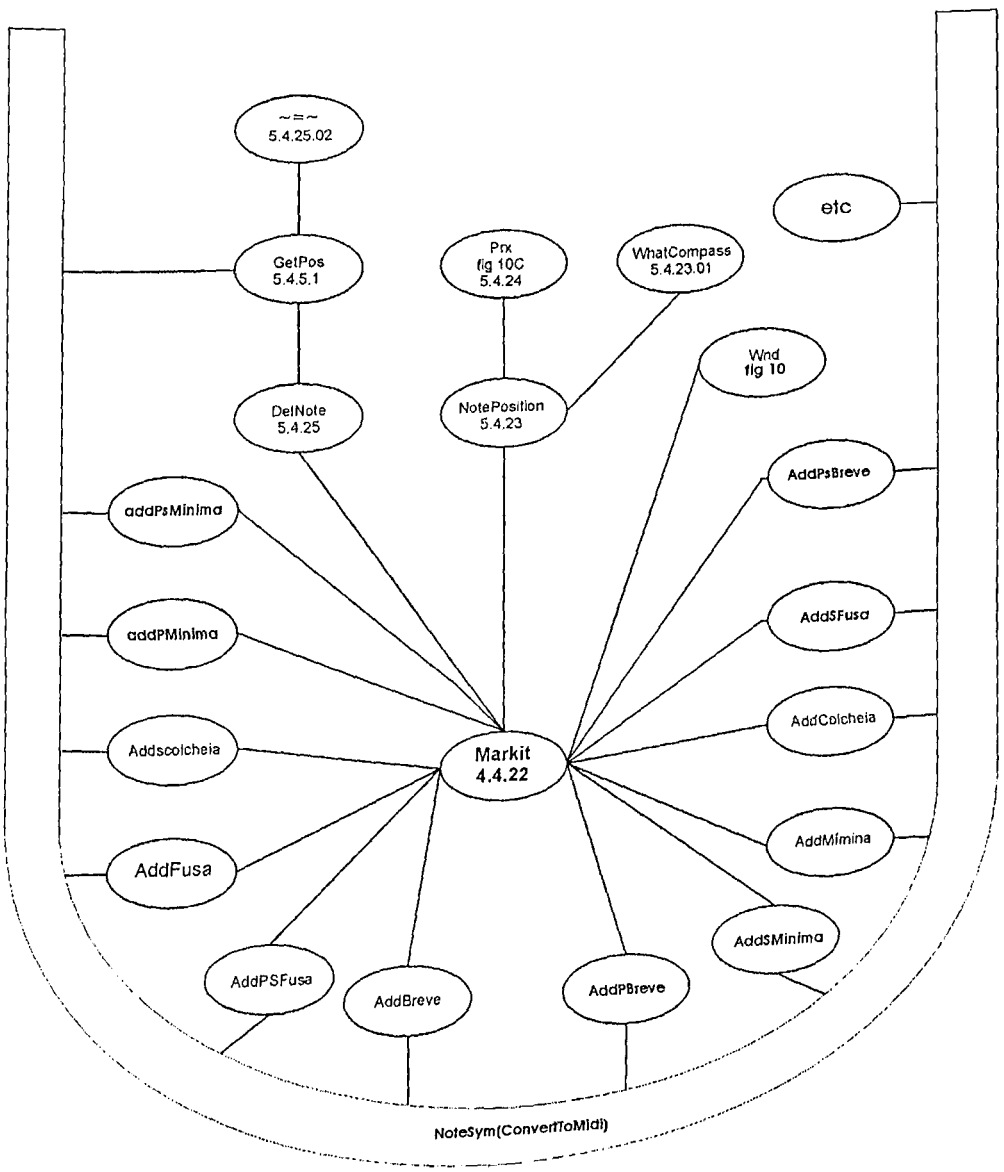
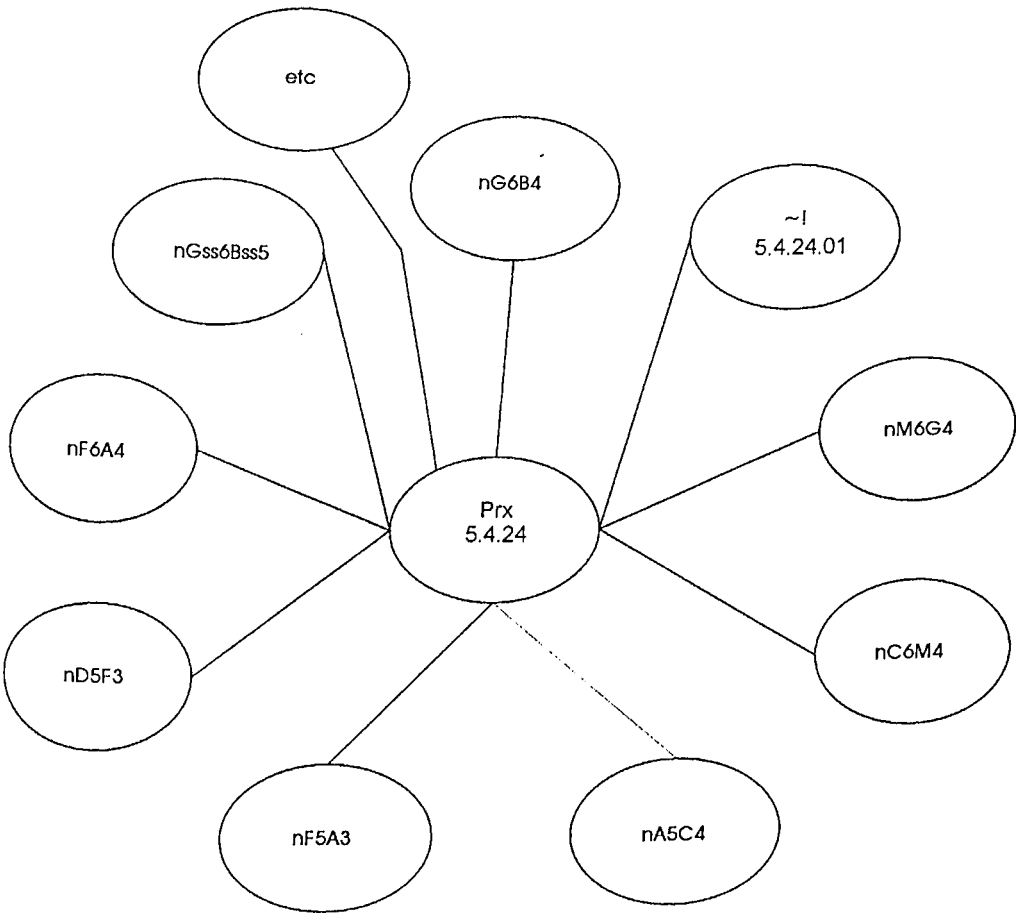
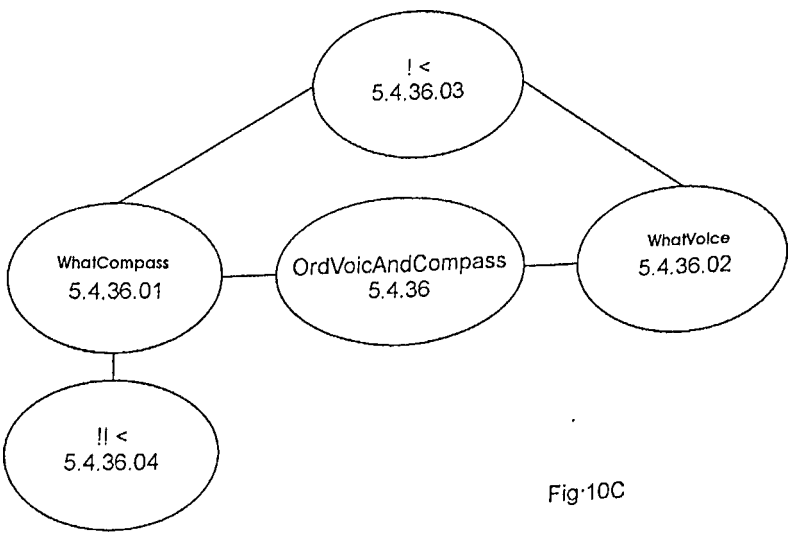


Fig 108

Função Prx(ScoreEdit)





Fig·10C