UNIVERSIDADE FEDERAL DE PERNAMBUCO

GRADUAÇÂO EM CIÊNCIA DA COMPUTAÇÂO

CENTRO DE INFORMÁTICA

2004.2

ANÁLISE HARMÔNICA FUNCIONAL AUTOMÁTICA

TRABALHO DE GRADUAÇÃO
EM INTELIGÊNCIA ARTIFICIAL

Aluno: Ricardo Enrique Pereira Scholz

Orientador: Geber Lisboa Ramalho

Recife, 02 de março de 2005.

Agradecimentos

A minha mãe, pai e irmão, pelo apoio e incentivo em todas as minhas caminhadas, sejam elas quais forem.

A Dani, que nos últimos quatro anos esteve ao meu lado, nos melhores momentos da minha vida e nos mais difíceis também.

Aos meus amigos, pelo apoio, pelos conselhos, pelas alegrias divididas e experiências vividas, e com quem eu aprendi e aprendo até hoje.

A Bel Zanforlin e Sylvio Pessoa, pela atenção que tiveram em ceder seus trabalhos, colocando-se à disposição para qualquer dúvida minha na utilização dos mesmos.

Aos professores do Centro de Informática, por todo conhecimento passado e aos funcionários pela infra-estrutura proporcionada.

Aos professores do Concervatório Pernambucano de Música, particularmente a Leonardo Saldanha, meu professor de piano, pela grande importância e influência na minha formação musical.

A Geber, em especial, por todo conhecimento e experiência compartilhados, além da importância na formação intelectual que busco.

Resumo

A realização de diversas tarefas na música ocidental, como improvisação, re-harmonização e arranjo, entre outras, necessita de um passo preliminar denominado análise harmônica funcional. Esta análise consiste na determinação do papel (função) de cada acorde de uma grade de acordes (harmonia), levando-se em consideração o contexto no qual cada acorde está inserido, ou seja, os acordes que o precedem e os que o sucedem.

A objetivo deste trabalho de graduação é realizar um estudo das técnicas aplicáveis ao problema e apresentar um *framework* para realização da análise harmônica funcional automática. Para tanto, estudamos as abordagens existentes, retomando uma delas para estendê-la às inúmeras variantes de seqüências de acordes e à correção *a posteriori* da harmonia analisada.

O processo se divide em duas fases, a análise e o pós-processamento, ambas fazendo uso de um motor de inferência, o JEOPS [FIG01], e quatro bases de regras, três para a fase de análise e uma para a fase de pós-processamento. A fase de análise se divide em três etapas: identificação de seqüências de funções pré-definidas, eliminação de seqüências sobrepostas e, por fim, atribuição de função aos acordes que não foram inseridos anteriormente em nenhuma seqüência. Já a fase de pós-processamento é responsável pela correção do nome das notas, a fim de manter a análise realizada conceitualmente consistente.

Índice

1. Introdução

Na música ocidental, a realização de diversas tarefas – como, por exemplo, re-harmonização, arranjo ou improvisação – exige como passo intermediário a análise harmônica funcional. Sendo assim, a automação de tal análise facilitaria bastante a automação destas outras tarefas, bem como seria útil a músicos que realizam estas tarefas manualmente.

A análise harmônica funcional consiste na atribuição de funções a cada acorde de uma grade de acordes. Assim, é possível inferir qual o sentido de um determinado acorde, ou seja, qual a intenção do compositor ao utilizar aquele acorde no contexto onde foi utilizado.

Este trabalho tem como objetivo propor um *framework* para automação da análise harmônica funcional, permitindo, a partir de uma grade de acordes, acompanhada ou não de uma letra de música, gerar uma estrutura mais completa que contém também a informação dos graus (ou funções) de cada acorde e a tonalidade de cada trecho.

A técnica utilizada tem por base a proposta de Pachet [PAC01], porém com algumas alterações. O processo sugerido utiliza um motor de inferência baseado em regras, o JEOPS [FIG01], nas duas fases em que é dividido, e foi implementado utilizando um *framework* musical em Java, o Ritornello [SER01]. A primeira fase, chamada fase de análise, divide-se em três etapas. A segunda fase, o pós-processamento, tem por objetivo tornar a grade de acordes conceitualmente consistente em etapa única.

Na fase de análise, a etapa inicial consiste na identificação de seqüências de funções utilizadas com certa freqüência e fáceis de identificar sem causar ambigüidades. Uma vez identificadas estas seqüências, que chamaremos de *chunks*, o segundo passo é responsável por eliminar os *chunks* redundantes ou menos relevantes, de maneira a não permitir a sobreposição de dois *chunks*. Por fim, a terceira etapa atribui funções aos acordes que não estão associados a nenhum *chunk*, os chamados *gaps*, inserindo-os nos *chunks* vizinhos.

O pós-processamento é responsável pela correção dos nomes das notas, a fim de manter a consistência da música após a análise. Esta

correção não modifica o acorde usado, apenas o substitui por acordes enarmônicos, ou seja, as mudanças são apenas conceituais.

O *framework* criado inclui ainda uma interface gráfica que recebe como entrada um texto, como por exemplo uma cifra da *web*, e realiza a análise harmônica, mostrando o resultado encontrado e permitindo que seja editado pelo usuário.

No próximo capítulo alguns termos utilizados e conceitos necessários ao bom entendimento deste trabalho serão explicados. Além disso, o problema da automação da análise harmônica funcional será descrito em mais detalhes. Em seguida, as técnicas até então propostas para resolver o problema serão apresentadas. O quarto capítulo explica as fases de análise e pós-processamento, mostrando como ocorrem e que tipo de regras são utilizadas em cada etapa. A maneira como o *framework* foi modelado e implementado é exposta no quinto capítulo. Em seguida, os resultados obtidos e a maneira como foram calculados são explicados. E por fim, no sétimo capítulo, tem-se uma breve conclusão, analisando as dificuldades encontradas e os possíveis trabalhos futuros.

2. Contexto

Nesta sessão, serão apresentados alguns conceitos relacionados com o problema discutido neste trabalho. Em seguida, a análise harmônica funcional será descrita enquanto tarefa, juntamente com uma análise das principais dificuldades computacionais existentes em sua automação.

2.1 Conceitos

Os principais conceitos necessários ao bom entendimento deste trabalho são: enarmonia, campo harmônico, função dos acordes e empréstimo modal.

O conceito de **enarmonia** é simples: duas notas são enarmônicas quando têm o mesmo som (mesma freqüência de vibração), porém nomes diferentes. Exemplos de notas enarmônicas são: dó bemol e si, ou fá sustenido e sol bemol.

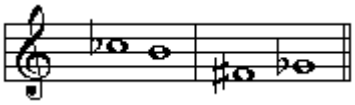


Figura 2.1 – Exemplos de notas enarmônicas. Em cada compasso, ambas as notas têm o mesmo som, embora sejam, conceitualmente, notas diferentes.

Grosso modo, o **campo harmônico** de uma determinada nota é o conjunto de acordes gerados a partir de uma escala cuja tônica é a nota em questão. Por exemplo, se a nota considerada for dó e a escala escolhida for a escala maior, teremos o que se segue:

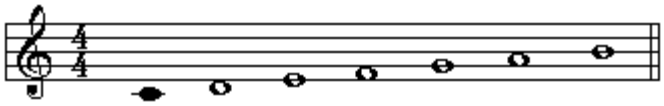
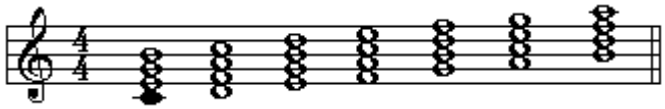


Figura 2.2 – Escala de Dó Maior.

Com base nisso, formando os acordes com as notas da escala, a partir de terças sobrepostas, obtemos os seguintes acordes, que formam o campo harmônico de dó maior.



C^{7M} Dm^{7} Em^{7} F^{7M} G^{7} Am^{7} $Bm^{7(b5)}$

Figura 2.3 – Campo harmônico de Dó Maior

Em um dado campo harmônico, cada acorde tem uma **função**, ou grau. Estes graus são relativos à tônica do campo harmônico, e vão do primeiro ao sétimo. A um conjunto de funções de acordes consecutivos, num mesmo campo harmônico, chamaremos de ***chunk***.

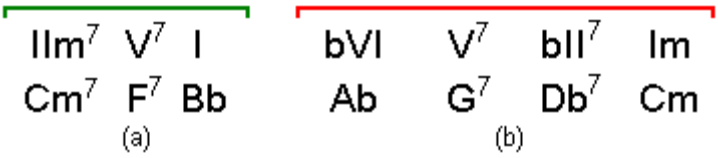


Figura 2.4 – Exemplos de *chunks*. (a) *Chunk* II-V-I da tonalidade Bb. (b) *Chunk* bVI-V-subV-Im da tonalidade Cm.

O objetivo da análise harmônica funcional é exatamente descobrir os campos harmônicos de cada trecho da música e os graus de cada acorde destes campos.

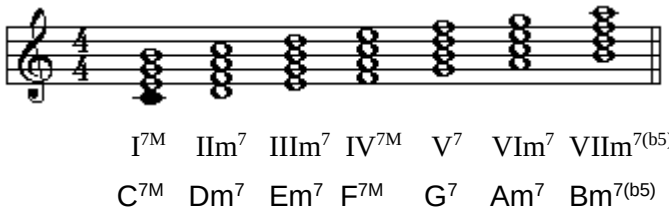


Figura 2.5 – Campo harmônico de Dó Maior com respectivas funções dos acordes.

Quando um determinado trecho, em um dado campo harmônico, utiliza um acorde pertencente ao campo harmônico de outra tonalidade cuja tônica é a mesma do campo harmônico em questão, i. e., apenas o modo é diferente, diz-se que ocorreu um **empréstimo modal**. Considere o exemplo abaixo.

bVI^{7+} V^{7} I^{7M}

Ab^{7+} G^{7} C^{7M}

Figura 2.5 – Empréstimo modal. O Ab^{7+} é um empréstimo modal de C menor harmônico.

O campo harmônico do trecho “G^{7} C^{7M}” é C maior. Entretanto, o Ab^{7+} pertence ao campo harmônico de C menor harmônico. Ou seja, a tônica de

ampos os campos harmônicos é a mesma, neste caso C, embora os modos sejam diferentes. Sendo assim, diz-se que o Ab^{7+} é um empréstimo modal de C menor harmônico.

2.2 O Problema

Como já foi dito, a análise harmônica funcional consiste na atribuição de funções, ou graus, aos acordes de uma música. Tal tarefa deve levar em consideração o contexto em que cada acorde está inserido, ou seja, os acordes anteriores e posteriores ao acorde em questão.

Cada trecho da música tem uma tonalidade, que gera um campo harmônico. Sendo assim, os acordes dentro do trecho em questão têm alguma relação com a tônica da tonalidade deste trecho. Esta relação é justamente a função do acorde. Logo, toda função só faz sentido dentro de um campo harmônico. A análise harmônica funcional consiste em descobrir como uma música se divide em campos harmônicos e a função de cada acorde em seus respectivos campos harmônicos.

A análise harmônica funcional não é trivial nem para os seres humanos por lidar com uma linguagem cuja gramática é sensível ao contexto, embora músicos experientes a façam com naturalidade. Por exemplo, no caso da figura 2.5, desconsiderando-se o contexto, o Ab^{7+} poderia ser $\mathbf{I^{7+}}$ de Ab, $\mathbf{VI^{7+}}$ de Cm, $\mathbf{IV^{7+}}$ de Eb, $\mathbf{III^{7+}}$ de Fm, etc. O contexto considerado pode ir além do acorde vizinho, sendo necessário, em alguns casos raros, analisar a música inteira para determinar a função de um dado acorde. Cada músico tem sua própria maneira, na maioria das vezes muito subjetiva, de inferir os campos harmônicos e o papel dos acordes numa música. Entretanto, apesar da subjetividade dos métodos, os resultados são, quase sempre, bastante parecidos, quando não iguais.

Sendo assim, a automação desta tarefa é um problema não trivial, dado que trata-se de uma atividade que exige bastante conhecimento especialista. A forma de modelar o conhecimento, a existência de situações ambíguas ou mais de uma possibilidade de análise para um mesmo trecho e a inexistência de regras formais para realizar a tarefa tornam o problema ainda mais complexo.

Portanto, um sistema ou *framework* que se proponha a realizar análise harmônica funcional deve ter como principais características a abrangência, ou seja, o sistema deve conseguir analisar muitos tipos de seqüências; a corretude, sendo preferível que acordes da seqüência fiquem sem função atribuída a terem funções incorretas, embora as funções não ideais – funções não utilizadas com freqüência pelos especialistas – possam ser usadas; a extensibilidade, isto é, a melhoria incremental do sistema deve ser fácil; uma interface gráfica adequada, preferencialmente permitindo que o resultado da análise seja modificado pelo usuário e recebendo como entrada um formato simples, como texto, facilitando que cifras da *web*, por exemplo, sejam utilizadas como entrada.

3. Estado da Arte

A automação da análise harmônica funcional já foi objeto de estudo em diversas pesquisas, embora mais como meio que como fim. Veremos que a grande maioria dos estudos que envolvem automação da análise harmônica funcional o fazem como tarefa intermediária, e por este motivo a tratam de maneira superficial. Além disso, os dados de entrada utilizados costumam ser bastante específicos para cada problema.

As técnicas utilizadas são as mais diversas. Em Ulrich [ULR01], por exemplo, uma música é modelada como uma série de acordes, onde cada acorde é composto por um conjunto de notas, podendo um determinado conjunto de notas representar mais de um acorde. Os acordes são reconhecidos através de uma gramática livre de contexto. Em seguida, os campos harmônicos são identificados iterativamente: inicialmente, cada acorde é um campo harmônico, e a cada passo, dois campos adjacentes são combinados, se os acordes de ambos puderem ser analisados sob o mesmo campo harmônico. Já em Giomi e Ligabue [GIO01], trata-se essencialmente da composição de seqüências harmônicas através de uma estrutura em árvore, de caráter derivacional, que sintetiza, compasso-a-compasso, as várias soluções, juntamente com as probabilidades de incidência de cada acorde. A ávore definida gera seqüências derivadas do Blues.

Estas abordagens não foram utilizadas neste trabalho por considerar entradas em formatos muito específicos, que não seriam encontradas com facilidade, e tratar o processo de análise harmônica funcional como um meio para uma outra tarefa, na maioria das vezes, composição automática, ocasionando em análises não abrangentes, suficientes apenas para a realização das tarefas desejadas. Além disso, as abordagens não são facilmente extensíveis, dificultando a melhoria incremental da análise realizada.

Em particular, duas das referências encontradas tratam do problema de maneira mais específica e numa abordagem mais aprofundada. A primeira baseia-se na definição de uma gramática capaz de gerar seqüências

harmônicas jazzísticas e é proposta por Steedman [STE01]. A idéia principal consiste no uso de regras para definir uma gramática capaz de reconhecer seqüências de acordes numa hamonia jazzística. A definição da gramática é incremental, i. e., a priori, a gramática é bastante simples. Após a aplicação de todas as regras de derivação, chega-se então uma gramática mais completa, capaz de reconhecer seqüências de acordes mais complicadas. Tal abordagem não foi escolhida para o problema em questão neste trabalho, principalmente pela dificuldade em definir tal gramática. Além disso, casos muito particulares não seriam analisados, já que provavelmente ficariam de fora da gramática definida, visto que seria necessário balancear a complexidade da gramática e a qualidade do resultado.

A segunda referência, Pachet [PAC01], propõe uma análise em três etapas: busca de padrões recorrentes, eliminação de padrões conflitantes e classificação de *gaps*, nesta ordem. Todas as etapas usam uma base de regras e um motor de inferência. O primeiro passo consiste em definir regras para reconhecer padrões muito usados na harmonia jazzística. Uma vez identificados estes padrões, regras para eliminar padrões sobrepostos e/ou redundantes são definidas, a fim de que cada acorde tenha somente uma função atribuída. Por fim, são definidas regras para classificar os acordes que restam após o segundo passo da análise.

Após analisar as abordagens utilizadas nestes trabalhos, escolheu-se esta última por melhor se adequar às necessidades deste trabalho de graduação. Além da facilidade de modificação das regras, permitindo o aperfeiçoamento da análise, a extensão da idéia a fim de incluir novas tarefas além da análise harmônica funcional, através da inclusão de novas etapas no processo, após a etapa de classificação dos *gaps*, torna-se simples, bastando definir regras adequadas para tal.

4. Automação da Análise Harmônica Funcional

A seguir, o processo utilizado para automatizar a análise harmônica funcional será explicado em maiores detalhes. Cada uma das etapas da análise será discutida e exemplificada, bem como a etapa única do pós-processamento.

A música que se deseja analisar passa por um pré-processamento, onde são identificados os acordes e a letra, se esta existir. É gerado então um objeto composto por uma letra, contendo todos os meta dados necessários, como a posição dos acordes em relação a ela, uma lista de acordes, contendo todos os acordes da música em questão, ordenadamente, e uma lista de *chunks*, a princípio vazia.

4.1 Encontrando Padrões Simples

A primeira etapa do processo proposto é a busca por padrões simples, recorrentes e que não causem ambigüidades. Para tal, usa-se uma base contendo dezessete regras, uma para cada padrão pré-definido. O motor de inferência tenta aplicar as regras a todas as combinações possíveis de acordes.

A maioria das regras desta base foi adaptada de Ramalho [RAM01]. As demais foram criadas a partir do resultado da análise manual de diversas músicas com harmonia jazzística, de compositores como Tom Jobim, Baden Powel, Cartola, Vinícius de Moraes, George Benson, Louis Armstrong, Herbie Hancok, Miles Davis, entre outros, observando os padrões que ocorriam com freqüência, mas tomando-se o cuidado para não criar regras ambíguas. Além disso, cadências e trechos muito utilizados em Chediak [CHE01] foram considerados.

Após a criação de uma nova regra, esta era testada com várias músicas antes de ser definitivamente incorporada à base de regras. A inserção de novas regras na primeira base requer bastante cuidado, já que as redundâncias e sobreposições geradas pela nova regra terão de ser tratadas pela segunda base de regras, implicando na mudança, exclusão ou inserção de regras nesta última.

As regras se aplicam a seqüências de acordes consecutivos. Uma vez que uma regra é disparada para um conjunto de acordes, é criado um novo *chunk* que é preenchido com as funções dos acordes em questão. Este *chunk* é então inserido na lista de *chunks*, como no exemplo a seguir.

```
regra II-V Maior {
    declarações
        c1 e c2 são acordes da base de conhecimento;
    condições
        c1 precede c2;
        c1 é menor e tem sétima menor, caso a tenha;
        c2 é dominante;
        O intervalo entre as tônicas de c1 e c2 tem
            dois tons e meio;
    ações
        Crie um novo chunk c, cuja tonalidade é
            maior e uma quarta justa acima da tônica
            de c2;
        Insira a função "IIm7" no chunk c;
        Insira a função "V7" no chunk c;
        Insira c na base de conhecimento;
}
```

O arquivo de regras utilizado está no anexo A1. Os *chunks* identificados pelas regras desta base são os da Tabela 4.1 a seguir.

***Chunks* da Tonalidade Menor Harmônica**	***Chunks* da Tonalidade Maior**
bII^{7} Im	**V^{7} I^{7M}**
IVm^{7} Im	**IIm^{7} V^{7}**
IIm$^{7(b5)}$ bII^{7} Im	**bII^{7} I^{7M}**
V^{7} Im^{7M}	**IV^{7M} I^{7M}**
VII° Im^{7M}	**IIm^{7} V^{7} I^{7M}**
IIm$^{7(b5)}$ V^{7}	**IIm^{7} bII^{7} I^{7M}**
IIm$^{7(b5)}$ V^{7} Im^{7M}	**VIm^{7} IIm^{7} V^{7} I^{7M}**
bVI^{7M} IIm$^{7(b5)}$ V^{7} Im^{7M}	**IV^{7M} IIIm^{7} VIm^{7} IIm^{7} V^{7} I**
Ivm^{7} bIII$^{7M(\#5)}$ bVI^{7M} IIm$^{7(b5)}$ V^{7} Im^{7M}	

Tabela 4.1 – *Chunks* identificados pela primeira base de regras ("Match *Chunks* Base").

Não há ordem de prioridade entre as regras, i. e., para um conjunto de acordes, se duas regras distintas se aplicam, ambas serão disparadas. Por isso, há o cuidado em não definir regras que causem ambigüidades, ou

seja, que permitam o disparo concomitante com outras regras para o mesmo conjunto de acordes. A maneira escolhida para resolver este problema foi não definir as regras ambíguas e tratar o caso no terceiro passo da análise. Embora este problema possa ser também resolvido com a inserção de regras na segunda base que escolhem determinado *chunk*, de acordo com o contexto em que está inserido.

O problema acima ocorreria, por exemplo, com as regras para identificar uma cadência deceptiva (**"V^7 VIm^7"** na tonalidade maior) e uma cadência do tipo **"$bVII^7$ Im"**, na tonalidade menor. A priori, é impossível definir se uma seqüência de acordes como "G^7 Am" tem as funções **"V^7 VIm"** em dó maior ou **"$bVII^7$ Im"** em lá menor.

Entretanto, considerando o contexto em que a seqüência está inserida, é possível tal classificação. Se, por exemplo, o contexto é "F^{7M} Em^7 Am^7 Dm^7 G^7 Am", sabe-se que o campo harmônico em questão é de dó maior (com as respectivas funções **"IV^{7M} $IIIm^7$ VIm^7 IIm^7 V^7 VIm"**, logo, trata-se de uma cadência deceptiva. Já num contexto como "Dm7 Bm7(b5) G7 Am", é possível inferir que trata-se das seguintes funções **"IVm^7 $IIm^{7(b5)}$ $bVII^7$ Im"**, no campo harmônico de lá menor.

Já a estratégia adotada resolve o problema sem identificar as funções destes acordes no primeiro passo de análise, deixando-os como *gaps* e resolvendo suas funções na terceira etapa da análise, depois que o contexto já foi analisado. Ou seja, o Am é classificado com base no campo harmônico do *chunk* anterior.

A figura abaixo mostra como a terceira etapa de análise consegue resolver o problema. À esquerda, a harmonia sem a identificação do *chunk* "V7 VIm7" e à direita, o resultado da análise após a etapa de classificação dos *gaps*.

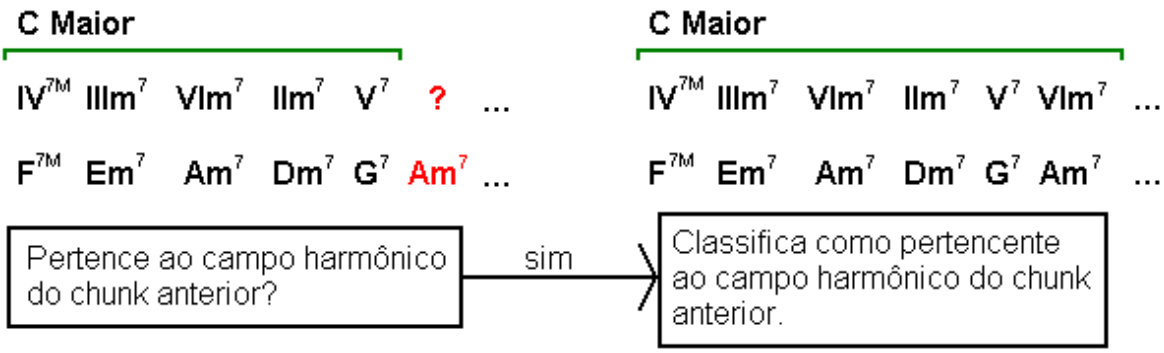


Figura 4.1.2 – Resolvendo ambigüidade na terceira etapa de análise.

Nota-se que, desta maneira, a lista de *chunks* identificados nesta fase irá conter diversos *chunks* redundantes, como no exemplo da Figura 4.1.3. No entanto, esta redundância se faz necessária para que a segunda etapa da análise consiga eliminar *chunks* de menor prioridade, sem perder informações sobre as outras funções do *chunk* eliminado. No exemplo abaixo, se o *chunk* verde fosse eliminado na segunda fase da análise, para que o Am^7 fosse inserido em outro *chunk*, a informação sobre as funções dos outros acordes continuaria existindo nos *chunks* vermelho, azul e mostarda.

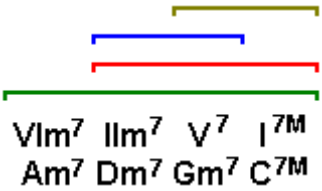


Figura 4.1.3 – *Chunks* encontrados pela primeira base de regras. Os *chunks* vermelho, mustarda e azul são redundantes.

4.2. Removendo Padrões Sobrepostos

A lista de *chunks* obtida após a primeira etapa da análise pode conter *chunks* sobrepostos, dada a maneira como o motor de inferência tenta aplicar as regras da primeira base. A segunda etapa da análise tem por objetivo gerar uma lista de *chunks* sem sobreposições, i. e., eliminar *chunks* de maneira que, ao final da etapa, a lista de *chunks* não contenha mais que uma função para cada acorde da música. Para isso, as regras definidas retiram da base *chunks* considerados de menor prioridade ou redundantes.

As quatorze regras desta base foram definidas a partir das possíveis sobreposições de *chunks* geradas pela primeira base. O arquivo de regras

utilizado está no anexo A2. Numa análise mais detalhada, percebe-se que as sobreposições possíveis podem ser de dois tipos: totais – ou seja, um *chunk* está totalmente imerso em outro – ou parciais com apenas uma função em comum – isto é, a última função do primeiro *chunk* coincide com a primeira função do segundo *chunk*.

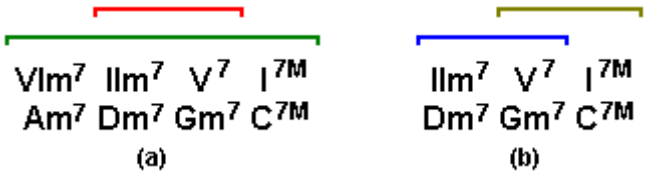


Figura 4.2.1 – Sobreposição total (a) e sobreposição parcial (b).

Como já foi dito, as sobreposições são resolvidas através de dois tipos de regras: regras de eliminação de *chunks* de menor prioridade e regras de eliminação de *chunks* redundantes. A base de regras desta fase é prioritária, ou seja, uma vez que, para um determinado conjunto de objetos, uma regra disparou, nenhuma outra regra irá disparar para os mesmos objetos. A regra disparada será sempre a que houver sido definida primeiro no arquivo de regras. Na tabela abaixo estão as regras definidas nesta base e os *chunks* eliminados por cada uma. As regras do tipo "Sobreposição A – B" significam uma sobreposição parcial em que o primeiro *chunk* termina com a função A e o segundo *chunk* começa com a função B.

Regra	Exemplo	*Chunk* Eliminado
1. Sobreposição Im – VIm^7	**A. (...) Im** **B. VIm^7 (…)**	**A**
2. Sobreposição Im – IIm^7	**A. (...) Im** **B. IIm^7 (...)**	**A**
3. Sobreposição Im – IVm^7	**A. (...) Im** **B. Ivm^7 (...)**	**A**
4. Sobreposição Im – Vm^7	**A. (...) Im** **B. Vm^7 (...)**	**A**
5. Sobreposição Vm^7 – VIm^7	**A. (...) Vm^7** **B. VIm^7 (...)**	**B**
6. Sobreposição Vm^7 – IIm^7	**A. (...) Vm^7** **B. IIm^7 (...)**	**B**
7. Sobreposição Vm^7 – IVm^7	**A. (...) Vm^7** **B. IVm^7 (...)**	**B**

8. Sobreposição V^7 – bII^7	**A. (...) V^7** **B. bII^7 (...)**	**A**
9. Sobreposição V^7 – V^7	**A. (...) V^7** **B. V^7 (...)**	**A**
10. Sobreposição I – V^7	**A. (...) I** **B. V^7 (...)**	**A**
11. Sobreposição I – bII^7	**A. (...) I** **B. bII^7 (...)**	**A**
12. Sobreposição I – IV^{7M}	**A. (...) I** **B. IV^{7M} (...)**	**B**
13. Sobreposição I – bVI^{7M}	**A. (...) I** **B. bVI^{7M} (...)**	**B**
14. Imersão	**A. Y Z** **B. X Y Z W**	**A**

Tabela 4.2 – Regras da segunda base (“Resolve Conflicts Base”).

4.2.1. Eliminando Chunks de Menor Prioridade

As treze primeiras regras do arquivo de regras utilizado têm como finalidade eliminar *chunks* de menor prioridade. Estas regras resolvem os casos de sobreposição parcial, simplesmente eliminando um dos dois *chunks* em questão, como no exemplo que se segue. Considere os dois *chunks* da figura abaixo:

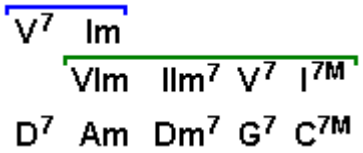


Figura 4.2.2 – Sobreposição parcial de *chunks*.

Percebe-se que o acorde Am tem duas funções possíveis. Em muitos casos, um determinado acorde pode ter mais de uma função, entretanto, na prática, elege-se apenas uma das funções possíveis, a fim de simplificar a notação. Sendo assim, a regra disparada seria a seguinte.

```
regra Im Vs VIm {
    declarações
        c1 e c2 são chunks da base de conhecimento;
    condições
        O primeiro acorde de c2 é o último de c1;
        O ultimo acorde de c1 tem função Im;
        O primeiro acorde de c2 tem função VIm;
    ações
        retire c1 da base de conhecimento;
}
```

Neste caso, o *chunk* de menor prioridade é o primeiro – **"V^7 Im"** – que é então eliminado da base. Observe que, na Figura 4.2.2, os *chunks* `c1` e `c2` declarados na regra acima são azul e verde, respectivamente.

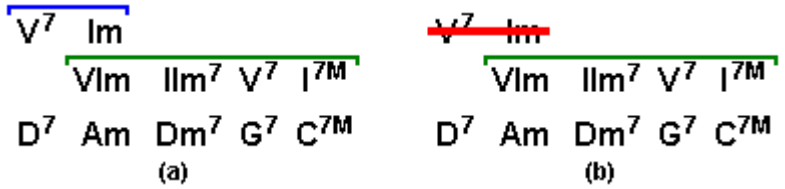


Figura 4.2.3 – (a) Estado inicial da base de conhecimento antes da segunda etapa da análise, e (b) estado da base após a aplicação da regra "Im Vs VIm", que eliminou o *chunk* "V^7 Im".

4.2.2. Eliminando Chunks Redundantes

A última regra do arquivo de regras, e por conseguinte a de menor prioridade, disparada somente quando a aplicação de nenhuma das outras regras é possível, trata os casos de sobreposição total, ou seja, redundância.

O funcionamento desta regra é simples. Dados dois *chunks*, se um está totalmente imerso em outro, este é eliminado. O exemplo abaixo mostra o funcionamento da regra.

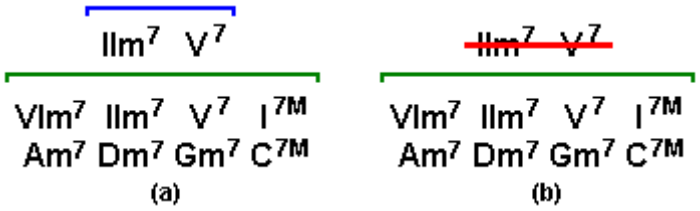


Figura 4.2.4 – (a) Estado inicial da base de conhecimento antes da segunda etapa de análise e (b) estado da base de conhecimento após a aplicação da regra de eliminação de *chunks* redundantes.

Nota-se que o *chunk* **"IIm^7 V^7"** (azul) é redundante, visto que identifica funções já identificadas pelo *chunk* **"VIm^7 IIm^7 V^7 I^{7M}"** (vermelho), e está totalmente imerso neste último. Sendo assim, é eliminado da base.

4.3. Classificando Acordes Ainda Não Classificados

Por fim, a terceira etapa da análise tem por objetivo classificar os acordes até então não classificados, i. e., identificar a função dos acordes que não estão inseridos em qualquer *chunk*. Para isso, a base de regras desta etapa contém vinte e oito regras, divididas em 8 grupos, conforme a tabela abaixo. As regras são aplicáveis a qualquer acorde que não faça parte de nenhum *chunk*.

Grupo	***Regra***
1. Igual.	Igual ao acorde posterior.
	Igual ao acorde anterior.
2. Campo harmônico dos *chunks* vizinhos.	Campo harmônico maior do próximo *chunk*.
	Campo harmônico menor harmônico do próximo *chunk*.
	Campo harmônico menor natural do próximo *chunk*.
	Campo harmônico maior do *chunk* anterior.
	Campo harmônico menor harmônico do *chunk* anterior.
	Campo harmônico menor natural do *chunk* anterior.
3. Campo harmônico dos acordes vizinhos.	Campo harmônico maior do próximo acorde.
	Campo harmônico menor harmônico do próximo acorde.
	Campo harmônico menor natural do próximo acorde.
	Campo harmônico maior do acorde anterior.
	Campo harmônico menor harmônico do acorde anterior.
	Campo harmônico menor natural do acorde anterior.
4. Empréstimo modal dos *chunks* vizinhos.	Empréstimo modal maior do próximo *chunk*.
	Empréstimo modal menor harmônico do próximo *chunk*.
	Empréstimo modal menor natural do próximo *chunk*.
	Empréstimo modal maior do *chunk* anterior.
	Empréstimo modal menor harmônico do *chunk* anterior.

	Empréstimo modal menor natural do *chunk* anterior.
5. Sub-dominante dos *chunks* vizinhos.	bII^7 do próximo *chunk*.
	bII^7 do *chunk* anterior.
6. Sub-dominante dos acordes vizinhos.	bII^7 do próximo acorde.
	bII^7 do acorde anterior.
7. V^7 (Dominante)	Dominante
8. I (Tônica)	I (tonalidade maior)
	Im^{7M} (tonalidade menor harmônica)
	Im^7 (tonalidade menor natural)

Tabela 4.3.1 – Regras da terceira base ("Resolve *Gaps* Base")

Analogamente à segunda base de regras, esta base também é prioritária. Isto significa que uma vez disparada uma regra para um acorde qualquer, nenhuma outra regra o será para o mesmo acorde. Além disso, o motor de inferência tenta aplicar as regras na ordem em que foram definidas na base. O arquivo de regras utilizado está no anexo A3.

Inicialmente, tenta-se classificar os *gaps* em que um dos vizinhos, posterior ou anterior, é o mesmo acorde. Desta maneira, o *gap* terá a mesma função que o acorde vizinho em questão, ou seja, será inserido no *chunk* do acorde vizinho com a mesma função deste último, como no exemplo da Figura 4.3.1 abaixo.

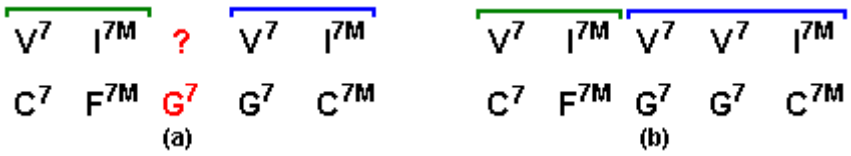


Figura 4.3.1 – Classificação de *gap* por igualdade com acorde vizinho. Estado da base de conhecimento antes (a) e depois (b) da aplicação da regra.

Se não for possível a aplicação da regra anterior, o campo harmônico dos *chunks* vizinhos é analisado e, caso o *gap* pertença a este campo harmônico, é inserido no *chunk*, com sua função no campo harmônico em questão. A Figura 4.3.2 mostra um exemplo da aplicação desta regra. Observa-se que o Em^7 pertence ao campo harmônico de C maior com a função $IIIm^7$.

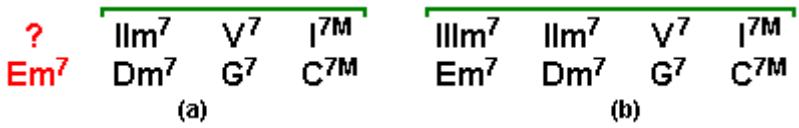


Figura 4.3.2 – Classificação de *gap* por pertencer ao campo harmônico do *chunk* vizinho. Estado da base de conhecimento antes (a) e depois (b) da aplicação da regra.

Não sendo possível classificar o acorde com base nos campos harmônicos de nenhum dos *chunks* vizinhos, os campos harmônicos dos acordes vizinhos são analisados. Se o *gap* tiver alguma função relativa ao acorde posterior ou anterior, é inserido no *chunk* em que este está inserido, com a função relativa ao mesmo, como mostra a Figura 4.3.3. Observa-se que o Am^7 pertence ao campo harmônico de D menor harmônico com a função V^7.

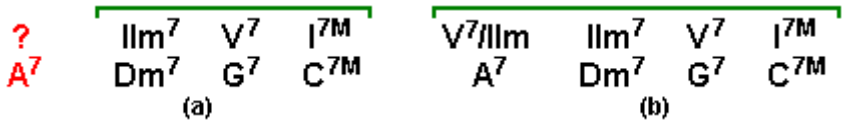


Figura 4.3.3 – Classificação de *gap* por pertencer ao campo harmônico do acorde vizinho. Estado da base de conhecimento antes (a) e depois (b) da aplicação da regra.

Seguindo a ordem de prioridade das regras, a próxima tentativa é classificar o *gap* como empréstimo modal de um dos *chunks* vizinhos. Caso a regra se aplique ao *gap*, este é inserido no *chunk* com a sua função na tonalidade da qual ocorreu o empréstimo. A Figura 4.3.4 exemplifica a aplicação desta regra. Observa-se que o $Dm^{7(b5)}$ pertence ao campo harmônico de C menor harmônico com a função $IIm^{7(b5)}$, embora a tonalidade do *chunk* vizinho seja C maior.

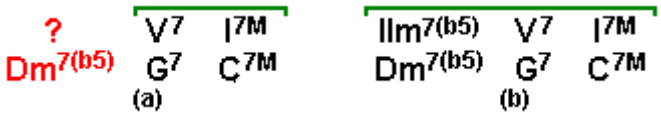


Figura 4.3.4 – Classificação de *gap* como empréstimo modal do *chunk* vizinho. Estado da base de conhecimento antes (a) e depois (b) da aplicação da regra.

O próximo passo é a tentativa de classificar o *gap* como dominante substituto (função $\mathbf{bII^7}$) de algum dos *chunks* vizinhos. Esta regra se faz necessária porque tal função não é identificada em nenhuma das outras regras até então definidas. A Figura 4.3.5 exemplifica a aplicação desta regra.

Figura 4.3.5 – Classificação de *gap* como dominante substituto do *chunk* vizinho. Estado da base de conhecimento antes (a) e depois (b) da aplicação da regra.

Analogamente, não conseguindo classificar o *gap* como dominante substituto do *chunk* vizinho, tenta-se classifica-lo como dominante substituto do acorde vizinho. O processo é semelhante ao da regra anterior, como mostra a Figura 4.3.6.

Figura 4.3.6 – Classificação de *gap* como dominante substituto do acorde vizinho. Estado da base de conhecimento antes (a) e depois (b) da aplicação da regra.

Se nenhuma das tentativas anteriores for aplicável ao *gap*, este é classificado com base em sua terça e sua sétima. Desta forma, considera-se quatro possibilidades. Se o acorde for dominante, i. e., tiver a terça maior e a sétima menor, é classificado como $\mathbf{V^7}$. Se for maior, é classificado como $\mathbf{I^{7M}}$. Sendo menor, as duas opções são: $\mathbf{Im^{7M}}$, ou seja, primeiro grau da tonalidade menor harmônica, ou $\mathbf{Im^7}$, isto é, primeiro grau da tonalidade menor natural.

4.4. Correção da nomenclatura dos acordes

A fase de pós-processamento tem por objetivo corrigir os nomes dos acordes, de maneira a tornar a análise realizada consistente. Esta fase se faz necessária para que a fase de análise possa ser mais robusta, encontrando as funções dos acordes pela relação entre os mesmos, no que diz respeito à quantidade de semitons, abstraindo os nomes das tônicas. A Figura 4.4.1 mostra um exemplo em que a correção da nomenclatura se faz necessária.

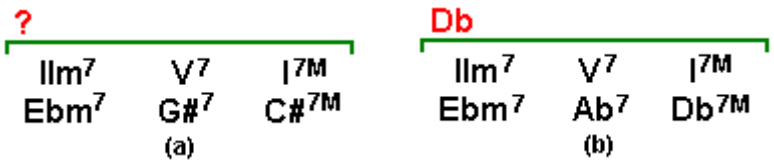


Figura 4.4.1 – Situação da cifra analizada antes da correção da nomenclatura (a). Observe que é impossível identificar a tonalidade do trecho. Á direita (b), uma das opções de correção de nomenclatura dos acordes. Agora a tonalidade é facilmente identificada.

Nota-se que, se o Ebm^7 for classificado como **IIm^7**, então a tonalidade em questão seria Db ao invés de C# - caso em que o $G\#^7$ e o $C\#^{7M}$ teriam as respectivas funções **V^7** e **I^{7M}**. Considerando a tonalidade Db, o **V^7** seria um Ab^7 ao invés do $G\#^7$ e o **I^{7M}** seria o próprio Db^{7M} ao invés do $C\#^{7M}$. No entanto, é bom que a fase de análise não diferencie notas enarmônicas, como o $G\#^7$ do Ab^7, assim as funções podem ser identificadas mesmo nos casos em que haja erros deste tipo na grade de acordes.

A base de regras utilizada também é prioritária e funciona analogamente às bases da segunda e terceira etapas da fase de análise. O arquivo de regras utilizado está no anexo A4. Esta base é composta de quatro regras, sendo a última necessária apenas por motivos técnicos, relacionados à maneira como o *framework* foi implementado. A Tabela 4.4.1 a seguir, mostra as regras definidas nesta base.

Regra
Corrige de acordo com o último acorde.
Usa tonalidade preferida.
Usa tonalidade recorrente.
Marca todos os *chunks*.

Tabela 4.4.1 – Regras da Correção de Nomes Dos Acordes (Correct Pitch Names Base)

Da maneira como as regras estão definidas na base de regras, o resultado obtido tem a ordem de preferência que se segue para redefinir a tonalidade de um *chunk*. Primeiro, tenta-se usar tonalidades que já apareceram na música. Em seguida, uma tabela de transição de tonalidades é usada, onde, dependendo da tonalidade do *chunk* anterior, escolhe-se a tonalidade do *chunk* atual, de maneira a facilitar a execução

da música. E por fim, para cada *chunk*, individualmente, escolhe-se a tonalidade de acordo com o último acorde do mesmo.

A tentativa de uso de tonalidades que já apareceram na música é simples. Percorre-se a música até chegar no *chunk* atual e caso algum *chunk* anterior tenha uma tonalidade igual ou enarmônica à do *chunk* em questão, este tem sua tonalidade modificada para a tonalidade anterior.

O uso da tabela de transição de tonalidades visa faciliar a leitura e execução da música, evitando grandes mudanças na armadura durante as trocas de tonalidade e o uso de tonalidades complexas. A tabela funciona da maneira a seguir.

Existem três grupos de notas, cada um associado a um conjunto de notas preferíveis. Se a tonalidade do *chunk* anterior estiver num dado grupo, usa-se para o *chunk* atual a tonalidade enarmônica do conjunto de notas preferíveis associado ao grupo em questão. Os três grupos de notas são disjuntos e sua união consiste em todas as notas existentes na música ocidental. Já os conjuntos de notas preferíveis contém exatamente doze notas cada, sem notas enarmônicas dentro de um mesmo conjunto. A Tabela 4.4.2 mostra os grupos de notas e seus respectivos conjuntos de notas preferíveis. Esta tabela trata apenas transições de tonalidades maiores para tonalidades maiores, sendo assim, quando há tonalidades menores envolvidas, a consulta à tabela ocorre utilizando as tonalidades relativas.

Grupos de Notas	***Tonalidades Preferíveis***
Bx, Ex, Ax, Dx, Gx, Cx, Fx, B#, E#, A#, D#, G#, C#, F#, B, E, A, D, G	C, C#, D, D#, E, F(?), F#, G, G#, A, A#, B
C	C, Db, D, Eb, E, F, F#, G, Ab, A, Bb, B
C, F, Bb, Eb, Ab, Db, Gb, Cb, Fb, Bbb, Ebb, Abb, Ebb, Gbb, Cbb, Fbb	C, Db, D, Eb, E, F, G, Gb, A, Ab, B

Tabela 4.4.2 – Tabela de preferência de tonalidades para transições.

Por fim, a correção da tonalidade do *chunk* e dos demais acordes com base no último acorde do *chunk* é apenas uma política utilizada, sem

maiores pesquisas a respeito, para manter a consistência local dos *chunks*, caso nenhuma das regras acima possa ser aplicada. O processo assume que o último acorde do *chunk* está correto e corrige a tonalidade do *chunk* e os demais acordes baseado neste acorde, conforme exemplificado na Figura 4.4.2.

Figura 4.4.2 – (a) Situação da cifra analizada antes da correção da nomenclatura. (b) Correção da nomenclatura baseada no último acorde.

No exemplo acima, o B^{7M} foi considerado correto e os outros dois acordes anteriores, o Dbm^{7} e o Gb^{7} foram corrigidos para se adaptar à tonalidade B maior, nas funções **IIm^{7}** e **V^{7}** respectivamente.

O motor de inferência aplica as regras da seguinte maneira: primeiro, todos os *chunks* têm sua tonalidade corrigida de acordo com o último acorde. Em seguida, cada *chunk*, exceto o primeiro, tem sua tonalidade modificada de acordo com a tonalidade do *chunk* anterior. Por fim, os *chunks* para os quais alguma tonalidade enarmônica já apareceu na música têm sua tonalidade substituída pela enarmônica em questão.

Percebe-se que é um processo “botton-up”, ou seja, corrigem-se os nomes das notas localmente – em cada *chunk* separadamente –, para em seguida considerar um trecho – transição entre dois *chunks* consecutivos – e só então considerar todo o contexto – no uso de tonalidades recorrentes.

5. Implementação

O problema foi modelado sob o paradigma de orientação a objetos e implementado utilizando a linguagem Java. Dentre os motivos que levaram a escolha de tal linguagem, o principal é o fato do motor de inferência JEOPS [FIG01] gerar código Java, além do *framework* musical utilizado, o Ritornello [SER01] também ser implementado em Java.

A arquitetura utilizada é composta por cinco módulos: um parser, um gerenciador de arquivos, um analisador, um validador e uma interface gráfica. Cada módulo será descrito em detalhes mais adiante. Os quatro primeiros módulos estão associados a uma interface, ou fachada, que torna transparente ao quinto módulo – interface gráfica – a implementação dos serviços oferecidos. Estes quatro módulos, juntamente com a fachada, formam o *framework* de análise harmônica funcional.

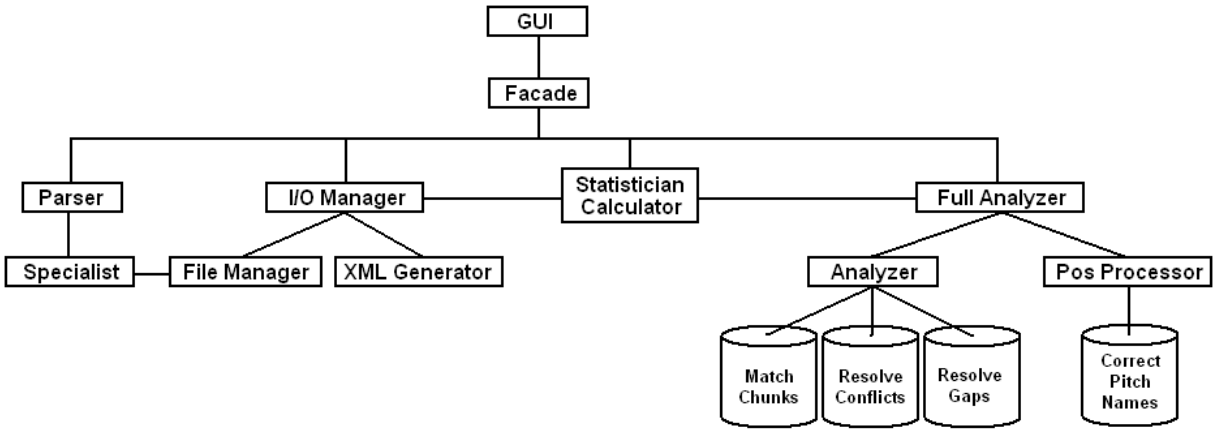


Figura 5.1 – Arquitetura da implementação.

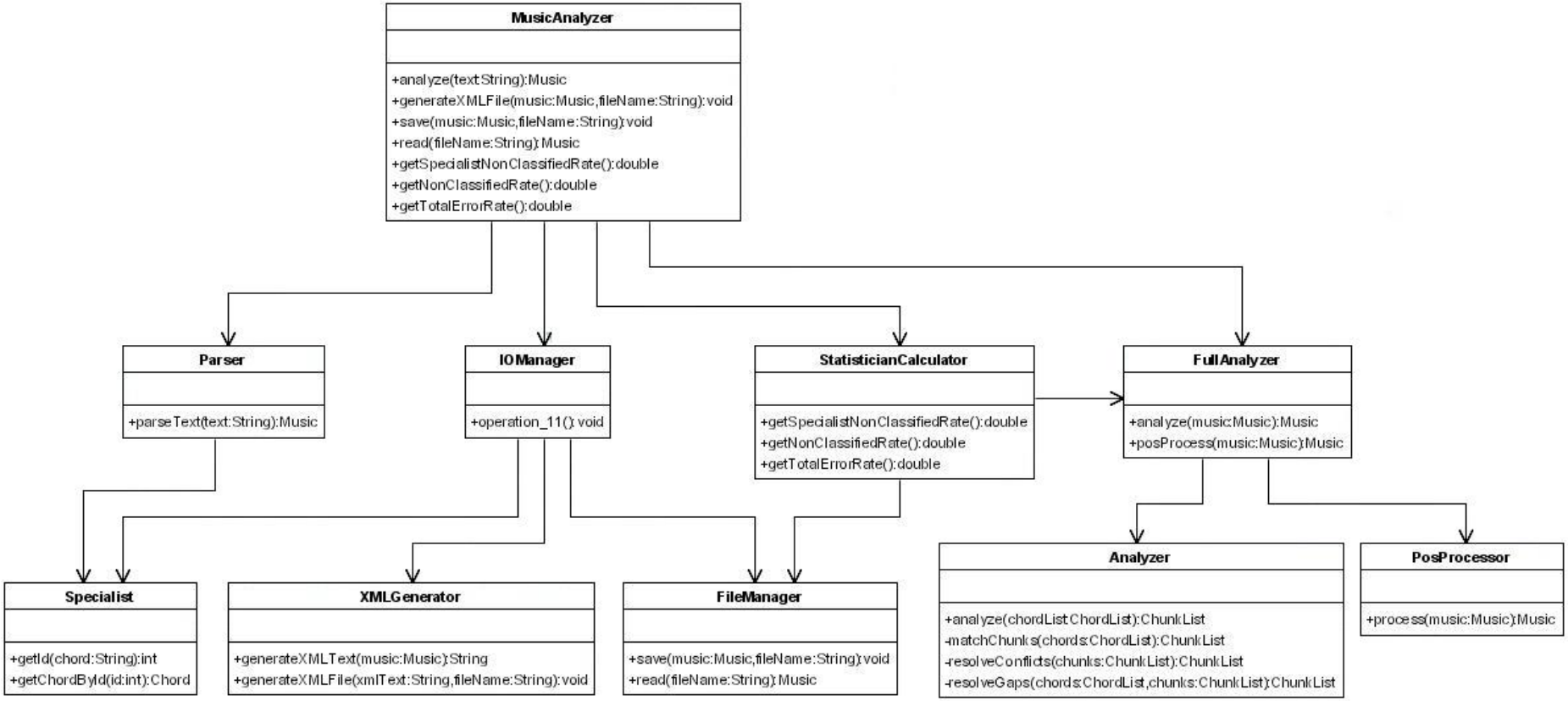


Figura 5.2 – Diagrama de Classes Resumido da Implementação

5.1 O Parser

O módulo de parsing é responsável por, dado um texto qualquer, identificar acordes, criando um objeto Music, que é formado pelos dados fornecidos como entrada, porém de maneira estruturada e com todas as meta informações necessárias para remontar o texto original.

O parser desenvolvido é capaz de reconhecer cifras no padrão brasileiro apenas e foi portado a partir do código fonte utilizado para parsing em Zanforlin [ZAN01], fornecido pela própria autora.

Este módulo contém uma classe `Parser`, que faz uso de uma classe `Specialist`. A classe Specialist é responsável por reconhecer um acorde ou levantar uma exceção caso o texto passado não seja um acorde.

O algoritmo utilizado, em linhas gerais, é simples: o parser quebra o texto em linhas e em seguida quebra cada linha em palavras. Para cada linha, tenta-se classificar as palavras como acordes; se em alguma das palavras o especialista levantar uma exceção, significa que não se trata de uma linha de acordes e sim de uma linha de texto. Passa-se então para a próxima linha e repete-se o processo. Se todas as palavras de uma linha forem classificadas como acordes, estes são inseridos na lista de acordes e passa-se para a próxima linha. Um fluxo secundário de processamento é realizado dentro deste algoritmo para salvar meta-informações sobre posição de cada acorde em relação ao texto associado na linha seguinte, entre outras, a fim de permitir a remontagem do texto no final da análise.

5.2 Gerenciador de Arquivos

O gerenciador de arquivos é o módulo responsável por gerar, salvar e recuperar arquivos nos dois formatos aceitos pelo *framework*: .mus e .xml. O primeiro formato, .mus, foi criado especificamente para guardar de maneira estruturada um objeto Music em um arquivo binario. O *framework* é capaz de ler e escrever arquivos neste formato. Assim, uma vez analisada, uma música pode ser salva em disco e recuperada posteriormente. Já no caso do formato .xml, o *framework* é capaz apenas de gerar o arquivo e salva-lo. Ainda assim, atualmente não são incluídas todas as informações do objeto Music no arquivo .xml, apenas a grade de acordes e suas funções, os campos harmônicos e as informações de

tensão e relaxamento na transição de acordes. A letra, quando existe, não é inserida no arquivo XML. A geração da saída no formato .xml é extremamente importante na utilização do *framework* por outras aplicações, facilitando enormemente a leitura das músicas analizadas por tratar-se de um formato adotado largamente hoje em dia.

Este módulo é formado por duas classes, a primeira – `FileManager` – é reponsável pela manipulação apenas de arquivos binários, encapsulando todo o código de escrita e leitura dos arquivos .mus. Já a segunda classe – `XMLGenerator` – cuida da geração de arquivos no formato .xml e das rotinas para salva-los em disco. O módulo oferece uma interface através da classe `IOManager`, que unifica os serviços das duas classes anteriores, deixando transparente para os outros módulos a manipulação de arquivos.

5.3 Analisador

O módulo de análise também é dividido em dois submódulos: análise e pós-processamento. O primeiro é formado por uma única classe – `Analyzer` –, juntamente com as classes geradas pelo motor de inferência JEOPS [FIG01] a partir dos arquivos de regras criados. A classe `Analyzer` oferece basicamente um único serviço, a análise, que encapsula as três etapas de análise, bem como as chamadas de métodos das classes geradas pelo motor de inferência, tornando transparente a maneira como a análise é realizada.

O submódulo de pós-processamento também é formado por uma classe única – `PosProcessor` –, além da classe gerada pelo motor de inferência JEOPS [FIG01], a partir do arquivo de regras definido. Esta classe oferece basicamente um único serviço, o de pós-processamento, o qual encapsula o código de chamada aos métodos da classe gerada pelo motor de inferência, mantendo a transparência sobre o pós-processamento.

Uma interface – `FullAnalyzer` – encapsula os serviços dos dois submódulos, além de permitir a adição de novos submódulos, a fim de oferecer mais serviços à fachada, com grande facilidade, bastando criar uma instância do referido submódulo e acrescentar seus serviços à interface do módulo de análise.

5.4 Validador

Este módulo é responsável pela validação dos resultados encontrados. A classe única do módulo – `StatisticianCalculator` – é capaz de fazer três tipos de cálculo: a percentagem de acordes não classificados (*gaps*) em uma série de músicas de teste, a percentagem de acordes classificados de maneira não ideal, considerando a opinião de um especialista, também em um conjunto de testes e, por fim, a percentagem de acordes não classificados pelo especialista no mesmo conjunto de testes. Para tal, este módulo utiliza arquivos de teste, editados por um especialista, com os resultados desejados na análise de várias músicas.

O cálculo da quantidade de acordes não classificados é trivial: os arquivos de teste são lidos, suas grades de acordes são analizadas pelo *framework* e o número de *gaps* no resultado é contado, gerando-se a porcentagem total de acordes não classificados.

Já o cálculo da percentagem de acordes classificados de maneira não ideal ocorre de maneira parecida: os arquivos de teste são lidos, suas grades de acordes são analizadas pelo *framework* e o resultado é comparado com a análise do arquivo lido. Cada função atribuída diferente da função do arquivo de teste é computada e a porcentagem total de acordes classificados de maneira não ideal é calculada ao final do processo.

Por fim, o cálculo da porcentagem de acordes não classificados pelo especialista é realizado apenas abrindo os arquivos de teste, contando quantos *gaps* existem ao todo e dividindo pela quantidade de acordes total.

5.5 Interface Gráfica

A interface gráfica é um módulo totalmente independente que faz uso do *framework* desenvolvido. Esta abordagem trás benefícios na implementação de uma nova interface gráfica, uma das possíveis melhorias propostas para este trabalho no capítulo 7. Por tratar-se de um módulo completamente independente, é importante que a nova interface faça uso dos serviços do *framework*, sem importar como estes estão implementados abaixo dela. A interface gráfica implementada possui as seguintes funcionalidades:

Abrir Arquivo: recupera-se um arquivo binário, .mus, anteriormente salvo em disco, permitindo sua vizualização e/ou edição;

Salvar Arquivo: uma vez analisada, uma música pode ser salva em disco para posterior consulta e/ou edição. O formato salvo é .mus;

Gerar Arquivo XML: gera e salva em disco um arquivo .xml, para uso por outras aplicações, por exemplo.

Analisar Música: realiza a análise de uma dada cifra ou grade de acordes fornecida, mostrando o resultado e permitindo que seja editado;

Analisar Musica Passo-a-passo: permite que a análise da música seja realizada passo-a-passo, mostrando o resultado parcial ao usuário a cada passo da análise.

Editar análise: permite a edição da análise realizada pelo *framework*, de maneira que acordes não analisados ou não analisados da maneira ideal possam ser corrigidos pelo próprio usuário. Esta funcionalidade é extremamente útil na geração do conjunto de testes.

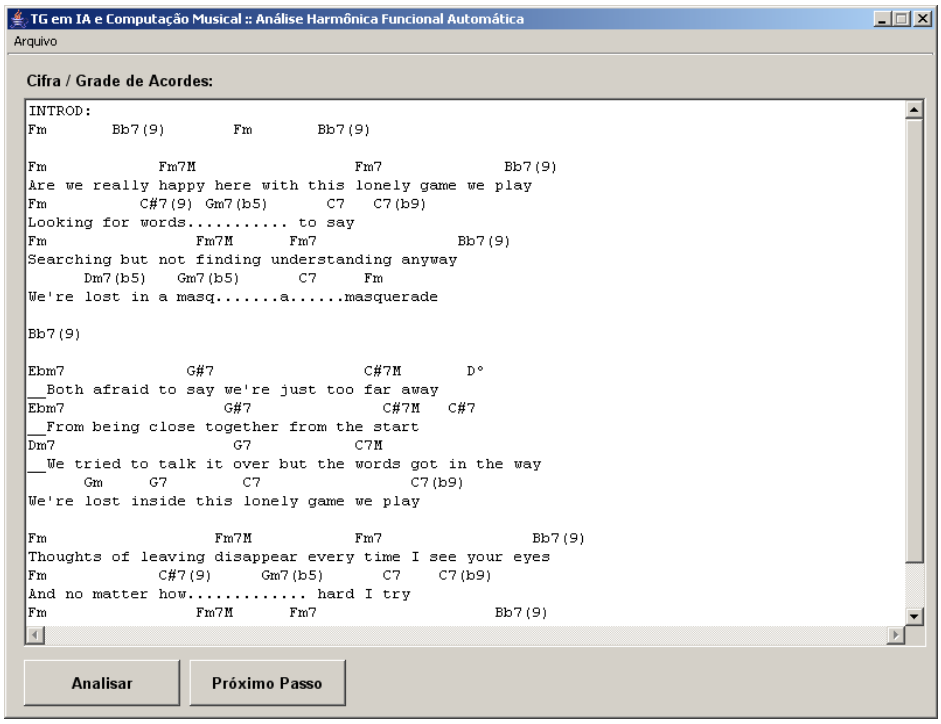


Figura 5.5.1 – Tela de entrada de dados. Os dados de entrada do sistema podem ser uma cifra, por exemplo.

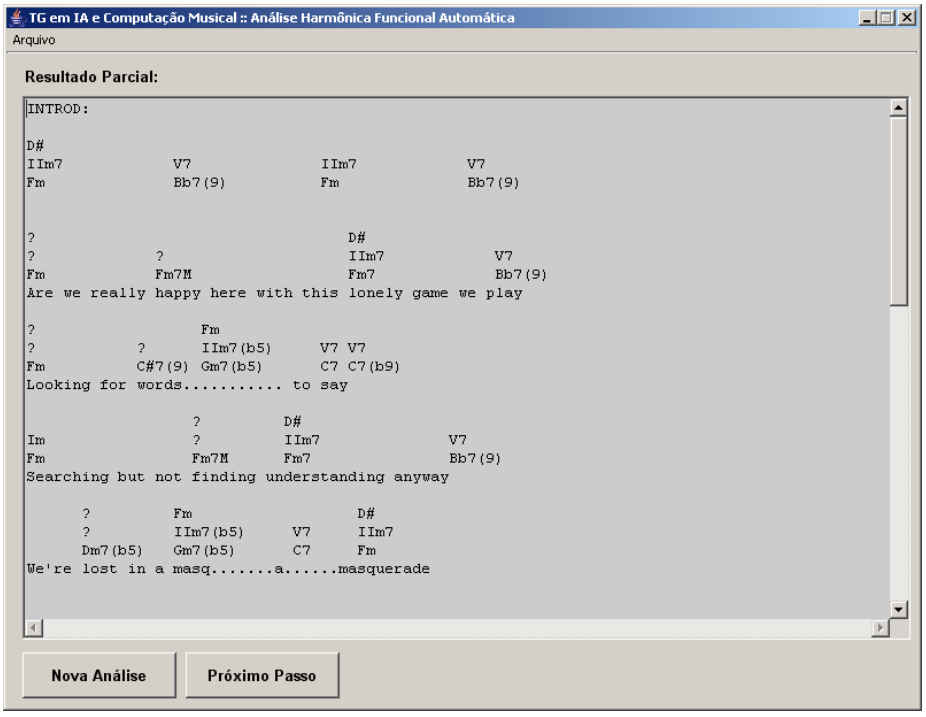


Figura 5.5.2 – Tela de análise parcial. Na análise passo-a-passo, esta tela mostra o resultado parcial da análise até então realizada. Percebe-se ainda uma grande quantidade de *gaps*.

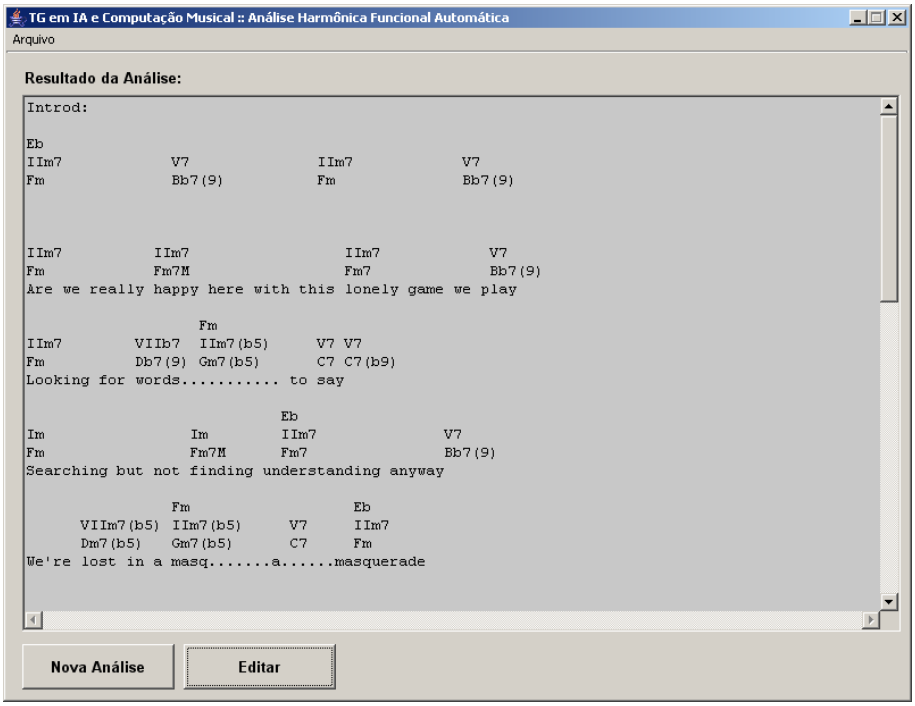
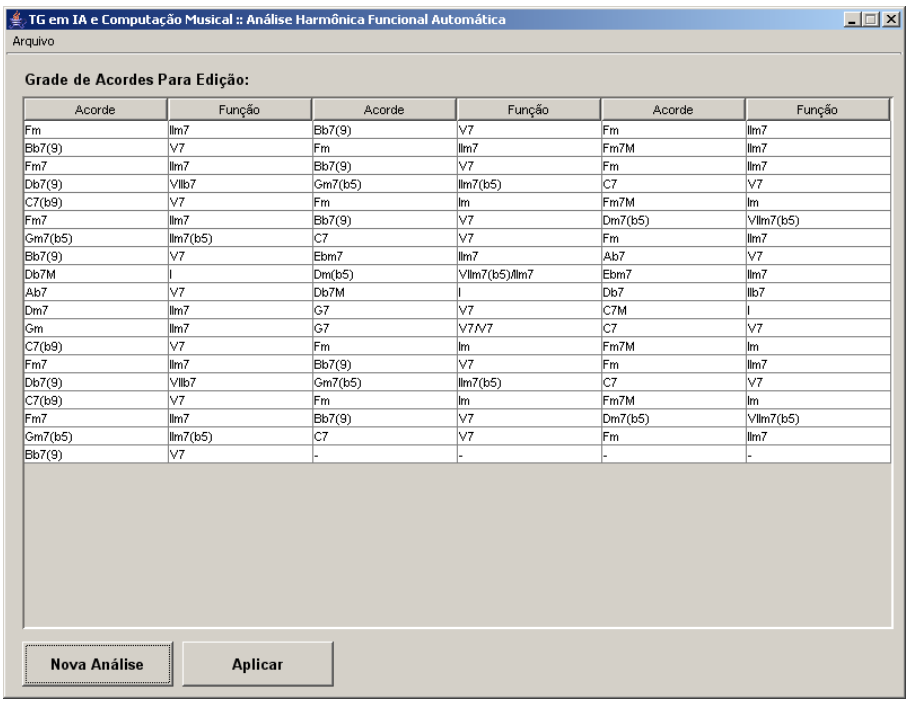


Figura 5.5.3 – Tela de resultado da análise. Após todas as etapas de análise e pós processamento, o resultado é mostrado. Nesta tela, as opções “Salvar” e “Gerar XML” são habilitadas no menu “Arquivo”.



Acorde	Função	Acorde	Função	Acorde	Função
Fm	IIm7	Bb7(9)	V7	Fm	IIm7
Bb7(9)	V7	Fm	IIm7	Fm7M	IIm7
Fm7	IIm7	Bb7(9)	V7	Fm	IIm7
Db7(9)	VIIb7	Gm7(b5)	IIm7(b5)	C7	V7
C7(b9)	V7	Fm	Im	Fm7M	Im
Fm7	IIm7	Bb7(9)	V7	Dm7(b5)	VIIm7(b5)
Gm7(b5)	IIm7(b5)	C7	V7	Fm	IIm7
Bb7(9)	V7	Ebm7	IIm7	Ab7	V7
Db7M	I	Dm(b5)	VIIm7(b5)/IIm7	Ebm7	IIm7
Ab7	V7	Db7M	I	Db7	IIb7
Dm7	IIm7	G7	V7	C7M	I
Gm	IIm7	G7	V7/V7	C7	V7
C7(b9)	V7	Fm	Im	Fm7M	Im
Fm7	IIm7	Bb7(9)	V7	Fm	IIm7
Db7(9)	VIIb7	Gm7(b5)	IIm7(b5)	C7	V7
C7(b9)	V7	Fm	Im	Fm7M	Im
Fm7	IIm7	Bb7(9)	V7	Dm7(b5)	VIIm7(b5)
Gm7(b5)	IIm7(b5)	C7	V7	Fm	IIm7
Bb7(9)	V7	-	-	-	-

Figura 5.5.4 – Tela de edição. Após o término da análise, o usuário tem a opção de editar a análise realizada pelo framwork, bastando corrigir as funções dos acordes desejados na tabela, escrevendo a nova função.

6. Resultados obtidos

O framework foi desenvolvido a fim de analisar músicas de harmonia funcional, excluindo-se o blues, por sua ambigüidade. Na música funcional, o jazz é o estilo de harmonia mais sofisticada, abrangendo tudo que é utilizado em outros estilos de música funcional, desde o período barroco até os dias atuais. Assim, os testes foram realizados com amostras de músicas cuja harmonia é jazzística, como o próprio jazz ou a bossa-nova.

A tabela abaixo mostra as porcentagens encontradas utilizando 12 músicas no conjunto de teste. Considera-se que o músico não é um especialista ou profissional, mas tem conhecimento de harmonia funcional suficiente para analisar satisfatoriamente boa parte das músicas de harmonia jazzistica.

Quantidade de Músicas / Total de acordes	*% de Acordes Não Analisados por um Músico*	*% de Acordes Não Analisados pelo Framework*	*% de Acordes Analisados de Maneira Não Ideal*
12 / 810	3,52	1,76	9,16

Tabela 6.1 – Resultados obtidos pelo módulo de testes.

Percebe-se que a percentagem de acordes não analisados pelo músico é maior que a percentagem de acordes não analisados pelo *framework*. Isto indica que a primeira é um subconjunto da segunda, ou seja, parte dos acordes não analisados pelo framework são os acordes não analisados pelo músico.

A análise não-ideal, ou seja, diferente do músico, gera erros em cadeia. Isto é, dado que um determinado acorde foi analisado de maneira não-ideal, os acordes relacionados a ele podem ser afetados, ampliando o efeito indesejado e prejudicando o resultado da análise.

7. Conclusões e Trabalhos Futuros

A proposta inicial consistia na construção de um *framework* para automação da análise harmônica funcional, supondo-se que a entrada estivesse corretamente cifrada, ou seja, os acordes estariam semanticamente e sintaticamente corretos. O *framework* de análise foi construído, obtendo-se resultados satisfatórios em relação aos requisitos. Constatou-se que muitas das entradas encontradas na *web* não vinham corretamente cifradas quanto à sintaxe, ou seja, após o término da análise, os nomes dos acordes tornavam-se inconsistentes conceitualmente. Sendo assim, decidiu-se implementar uma quarta base de regras, não prevista inicialmente, para identificar trechos inconsistentes e corrigir os nomes das tônicas dos acordes, substituindo-os por notas enarmônicas. Além disso, um editor foi criado, a fim de possibilitar a geração de músicas de exemplo com maior facilidade, para o cálculo dos percentuais de acerto.

Uma das maiores dificuldades encontradas foi a definição das regras das quatro bases de conhecimento, originada pela inexistência de definições formais de tais regras na literatura e pela própria maneira subjetiva com a qual a tarefa é realizada pelos músicos, não explicitando o uso de nenhuma regra. As regras definidas geraram resultados bastante satisfatórios, tendo em vista os percentuais de omissão (acordes não classificados) de um humano.

As regras definidas tiveram por base a harmonia jazzística, por ser esta a mais sofisticada estrutura harmônica na música ocidental. Ainda assim, houve extrema dificuldade em definir regras para identificar estruturas que, embora mais simples, geram grande ambigüidade, como o Blues. No Blues, o campo harmônico tem três funções muito utilizadas: **I^7**, **IV^7** e **V^7**. Assim, localmente, torna-se impossível identificar a diferença entre um **"I^7 IV^7"** e um **"V^7 I^7"**, por exemplo. Considere a seqüência "C^7 F^7". Tanto pode tratar-se de um **"V^7 I^7"** de um blues em F, quanto de um **"I^7 IV^7"** de um blues em C.

A definição de uma interface adequada foi um grande desafio. A versão apresentada neste trabalho não representa a interface desejada para o *framework*, sendo uma das possíveis melhorias em trabalhos futuros, principalmente no que diz respeito à etapa de edição de músicas já analisadas.

Um fato que ocorre com razoável freqüência em cifras encontradas na *web* é a troca de acordes por acordes formados pelas mesmas notas, porém com tônicas diferentes, impossibilitando uma análise automática adequada. Um músico com pouca experiência também não consegue identificar este tipo de problema. É o que ocorre com o trecho de *As Rosas Não Falam*, de Cartola, na figura abaixo, em cifra encontrada na *web*.

? *V* **Dm/B** E$^{7(b9)}$ ...O perfume que roubam de ti, Ai! *Im* Am Devias vir...	***IIm$^{7(b5)}$*** *V* **Bm$^{7(b5)}$** E$^{7(b9)}$ ...O perfume que roubam de ti, Ai! *Im* Am Devias vir...

Figura 7.1 – Trecho de "*As Rosas Não Falam*" (Cartola), disponível no site CifraClub (http://cifraclub.terra.com.br/cifras/cifras.php?idcifra=4133, acessado em 26/02/05).

É possível notar que os acordes Dm/B e Bm$^{7(b5)}$ são formados pelas mesmas notas (si, ré, fá e lá). Entretanto, no contexto da música, um Dm/B não teria sentido, enquanto o Bm$^{7(b5)}$ faria sentido como **IIm$^{7(b5)}$** da tonalidade A menor. No primeiro caso, o Dm/B poderia ser analisado como **IVm^{6}**, embora a análise como **IIm$^{7(b5)}$** seja bem mais coerente.

A correção deste tipo de problema, ou seja, a identificação dos acordes suspeitos, isto é, *gaps* ou acordes analisados que ficaram com funções estranhas no contexto e a troca por acordes equivalentes, seria interessante como trabalho futuro. Uma possível abordagem consistiria na definição de uma nova base de regras para, uma vez que a análise foi terminada, tentar analisar os acordes suspeitos da mesma maneira como os *gaps* são analisados, porém trocando-os por acordes formados pelas mesmas notas.

Este trabalho resultou num *framework* de análise harmônica abrangente – sendo capaz de analisar grande parte das seqüências encontradas nas harmonias jazzísticas –, correto – não permitindo a atribuição de funções erradas aos acordes, já que é baseado em regras, embora funções não ideais, algumas vezes sejam usadas – e facilmente extensível – bastando apenas mudar os arquivos de regras e recompilar o código para gerar uma nova versão. Além disso, a interface gráfica criada facilita a utilização do *framework*, permitindo que cifras da *web* sejam usadas com facilidade e as análises realizadas sejam editadas.

8. Referências

[FIG01] FIGUEIRA, Carlos S. F.. JEOPS – Integração entre Objetos e Regras de Produção em Java. Dissertação de Mestrado. 2000. Centro de Informática-UFPE.

[PAC01] PACHET, François. A meta-level architecture applied to the analysis of Jazz chord sequences. Institute Blaise Pascal – Laforia. Université Paris VI.

[SER01] SERAPIÃO, Sylvio P.. Ritornello: um *Framework* para Representação do Conhecimento Musical. Dissertação de Mestrado. 2004. Centro de Informática-UFPE.

[ULRO1] ULRICH, John W.. The Analysis And Synthesis Of Jazz By Computer. Computing and Information Science Department. University of New Mexico. Albuqerque, New Mexico.

[GIO01] GIOMI, Francesco e LIGABUE, Marco. Computational Generation and Study of Jazz Music. 1989.

[STE01] STEEDMAN, Mark J.. A Generative Grammar for Jazz Chord Sequences. 1984. University of Wariwik and University of Edinburgh.

[RAM01] RAMALHO, Geber L.. Construction D'un Agent Rationnel Jouant Du Jazz. Tese de Doutorado. 1997. Universite Paris VI.

[CHE01] CHEDIAK, Almir. O Livro do Músico. 4ª edição. Lumiar Editora. 1989. 182p.

[ZAN01] ZANFORLIN, Izabel. Um estudo sobre a personalização de encadeamento de acordes para violonistas. Trabalho de Graduação. 2004. Centro de Informática-UFPE.

[PAC02] PACHET, François. Computer Analysis of Jazz Chord Sequences: Is *Solar* a Blues?. 1997. SONY CSL-Paris.

[BAG01] BAGGI, Denis L.. NeurSwing: A Connectionist Workbench for the Investigation of Swing in Afro-American Jazz. 1989. International Computer Science Institute, Berkeley, CA. XI Computer Corporation, S. Clemente, CA. Istituto Dalle Molle per Studi sull´Intelligenza Artificiale, Lugano, Switzerland.

9. Datas e Assinaturas

Recife, março de 2005,

_____

Ricardo Enrique Pereira Scholz
Aluno

_____

Geber Lisboa Ramalho
Orientador