

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/288182574>

# Compilador musical: Uso da tecnologia de compiladores para validação de música simbólica.

Conference Paper · November 2015

CITATIONS

0

READS

271

2 authors:



[Leandro Pessoa](#)

Federal University of Viçosa

2 PUBLICATIONS 0 CITATIONS

[SEE PROFILE](#)



[Flávio Luiz Schiavoni](#)

Federal University of São João del-Rei

133 PUBLICATIONS 261 CITATIONS

[SEE PROFILE](#)

# Compilador musical: Uso da tecnologia de compiladores para validação de música simbólica

Leandro E. F. Pessoa<sup>1</sup>, Flávio L. Schiavoni<sup>2</sup>

<sup>1</sup>Instituto de Ciências Exatas e Tecnológica  
Universidade Federal de Viçosa - Campus Florestal  
Rodovia LMG 818, km 06 – Cep: 35690-000 – Florestal – MG – Brasil

<sup>2</sup>Departamento de Ciência da Computação  
Universidade Federal de São João Del Rei (UFSJ)  
São João Del Rei – MG – Brasil

leandro.pessoa@ufv.br, fls@ufs.j.edu.br

**Abstract.** *The symbolic musical notation is characterized by the use of textual elements denoting as a musical figures and the relationship between these symbols. Such characteristics allude to programming languages that have an analytical methods condensed in a process known as compilation. These similarities inspire the application of the three constituent stages of compilation: lexical analysis, syntactic and semantic in the musical context as an aid to various applications. This paper presents an approach to the analysis of musical symbolic notation based on the current technology to developing compilers.*

**Resumo.** *A notação simbólica musical se caracteriza pelo uso de elementos textuais que denotam figuras musicais e a relação entre estes símbolos. Tais características fazem alusão a linguagens de programação, que possuem métodos de análise condensados em um processo conhecido como compilação. Estas similaridades inspiram a aplicação das três etapas constituintes da compilação: análise léxica, sintática e semântica no contexto musical como auxílio à diversas aplicações. Este trabalho apresenta uma abordagem para a análise da notação simbólica musical com base na atual tecnologia para desenvolvimento de compiladores.*

## 1. Introdução

A notação simbólica musical fornece para os sistemas computacionais uma abordagem formal que permite o estudo para resolução de desafios presentes na representação musical [Dannenberg, 1993]. Por meio de notações simbólicas, se torna possível e adequada atividades de análise que permitem validação de notação musical padrão.

Muitas das notações simbólicas possuem formato textual estruturado, se assemelhando com a morfologia de linguagens de programação de alto nível, onde a análise léxica, sintática e semântica se mostram necessárias para o processo de compilação presentes no *front end* [Aho et al., 2008], que como resultado final, apresenta um programa executável em uma linguagem de máquina capaz de ser interpretada em nível de hardware, produto final do *back end* do compilador.

O uso da tecnologia de compiladores para análise de uma peça musical representada em uma notação simbólica, se mostra útil devido a possibilidade de definição dos parâmetros necessários para uma especificação completa ou relevante de peças musicais

no escopo de sua notação. Se torna possível então a caracterização de sistema de escrita utilizado para representar uma peça musical em uma notação simbólica, onde existem símbolos a serem reconhecidos como lexemas pertencentes a gramática durante análise léxica. A gramática é constituída de figuras musicais em formato textual. O correto uso das figuras musicais se caracterizam pela sintaxe, que é determinada pelas relações formais que interligam os constituintes da música, atribuindo assim uma estrutura. A especificação da estrutura sintática e semântica são definidas através de regras baseadas no contexto fornecido pela notação musical padrão, onde tempo e compasso regulam quantas unidades de tempo devem existir em cada compasso por exemplo.

O uso das três etapas do *front end* de um compilador fornecem a possibilidade de tratamento de erros de notação, através da busca de padrões sintáticos definidos, tornando-se assim uma ferramenta útil para validação da escrita de peças musicais que pode ser utilizada em procedimentos de impressão, reconhecimento ótico e sonoro.

## 2. Proposta de implementação

Para a elaboração de um compilador para uma linguagem de programação de alto nível, devemos implementar primeiramente as três etapas que compõem o processo de análise da compilação. É necessário a aplicação de conceitos básicos em computação, como máquinas de estados finitos. Diante do custo de tempo que demandaria aplicação deste e muitos outros conceitos, foram desenvolvidas ferramentas que são capazes de atuar na busca de elementos léxicos baseadas em padrões determinados através de expressões regulares que representam os padrões léxicos da linguagem.

$[A-Za-z][A-Za-z0-9]^*$

**Figura 1: Expressão regular capaz de reconhecer identificadores.**

Durante a análise léxica, uma tabela de símbolos é gerada com os respectivos *tokens* que denotam cada lexema encontrado no código de entrada, e seus respectivos atributos.

token	valor
id	x
op	+
id	y

**Figura 2: Tabela de símbolos para a expressão x+y.**

Através da análise dos *tokens* que foram instalados na tabela de símbolos, a análise sintática se torna possível em etapa posterior. Durante a análise semântica, duas etapas principais são executadas: a análise de contexto e é possível a geração de código intermediário. Os erros mais típicos que são encontrados durante esta análise são:

Uma variável não declarada.

Uma operação entre tipos de dados diferentes.

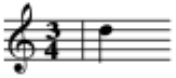

Atribuição de um literal para outro tipo.

Todas estas etapas podem ser realizadas com o uso de duas ferramentas conhecidas como Lex e Yacc. O programa Lex é um gerador de analisador léxico que auxilia na geração de um programa que controla o fluxo de lexemas a partir de expressões regulares [Lesk and Schimdt, 2001]. As expressões regulares fornecidas ao Lex, são traduzidas

para um programa que lê o código de entrada e o separa em blocos de acordo com os padrões encontrados que casam com os padrões especificados pelas expressões regulares. Cada bloco então identificado deverá fazer parte de uma tabela de símbolos para uso nas etapas seguintes. O programa Yacc através da descrição de uma gramática livre de contexto, é capaz de realizar a análise sintática e análise semântica através de sub rotinas que geralmente se resumem a verificação de tipos [Johnson, 2001].

Diante das similaridades entre linguagens de programação e linguagens de notação simbólica musical, o uso de ferramentas para construção de compiladores se torna intuitivo e requer poucas adaptações. A figura 3 apresenta uma comparação entre elementos léxicos de uma linguagem de programação e uma linguagem de notação simbólica.

Em ambas existem elementos léxicos e regras sintáticas bem definidas. Nas linguagens de programação as regras semânticas podem ser ou não bem definidas, por outro lado, a semântica da notação musical é passível de contexto histórico entre outros elementos que não são baseados em parâmetros técnicos, o que dificulta a especificação de regras semânticas em seu sentido completo.

	Lexema	Token	Valor	Exemplo de sintaxe
Linguagem de programação	a	id	a	int a;
Linguagem simbólica	d	note	semínima	dce
Notação musical	-	-		

**Figura 3: Similaridades entre linguagem de programação e linguagem simbólica.**

O ponto de partida deste trabalho são as linguagens simbólicas existentes como Lilypond [Nienhuys and Nieuwenhuizen, 2003], music21 [Ariza and Cuthbert, 2011, Cuthbert and Ariza, 2010], ABC [Oppenheim et al., 2010], MuseData [Hewlett, 1997], MIDI [Association et al., 1996], MusicXML [Good, 2001] entre outras. Cada formato um destes formatos de música simbólica trará um conjunto de lexemas e uma estrutura musical hierárquica para a aplicação das técnicas de compilador. A linguagem ABC foi utilizada para os testes apresentados neste artigo.

### 3. Compilando música simbólica

Assim como no processo de compilação de linguagens de programação, a estrutura básica do compilador é respeitada, onde existem duas seções evidentes: análise e síntese. A seção de análise divide o programa de entrada em partes e impõe uma estrutura gramatical sobre as partes. Se a análise detectar que o programa de entrada está sintaticamente mal formado ou semanticamente incorreto, devem ser emitidas mensagens de erro de maneira que o usuário consiga corrigir estes erros. A parte de síntese constrói o programa objeto desejado a partir da representação intermediária e das informações na tabela de símbolos gerada pela análise. Respectivamente chamamos estas duas partes do compilador de **front-end** e **back-end**.

#### 3.1. Front-end

As análises que o front-end executa para a compilação de música simbólica são semelhantes às empregadas nas linguagens de programação. Para reconhecimento de padrões da

notação são especificadas expressões regulares. Como as análises do front-end são dependentes da linguagem de programação, usaremos como exemplo nesta seção a linguagem de notação simbólica Abc.

Temos que na linguagem Abc, o lexema mais recorrente se refere ao lexema que denota notas musicais e seus atributos. Expressão regular capaz de detectar este padrão seleciona o lexema e identifica a nota e seus atributos, instalando assim um token na tabela de símbolos.

## Análise Léxica

Temos então como lexemas a serem identificados na primeira etapa, informações da música como seu título e compositor. Temos também informações sobre a métrica, tempo e tonalidade que serão utilizadas para definir um contexto em posterior análise sintática e semântica. Temos ainda como lexemas, notas musicais e símbolos associados de acordo com a construção da notação Abc que representam sua duração, possíveis acidentes, dinâmica, ornamentos, adornos e articulação. Há também lexemas que representam linhas de compasso simples ou dupla, compasso tracejado e barra final, funcionam como lexemas delimitadores. E ainda lexemas que denotam repetições, marcas de interrupção e marcas de pedal.

```
X:0
K:C
T:Test Song
C:Hal 9000
L:1/4
M:4/4
Q:"Without afraid" 1/4=92
cegb|cega|
```

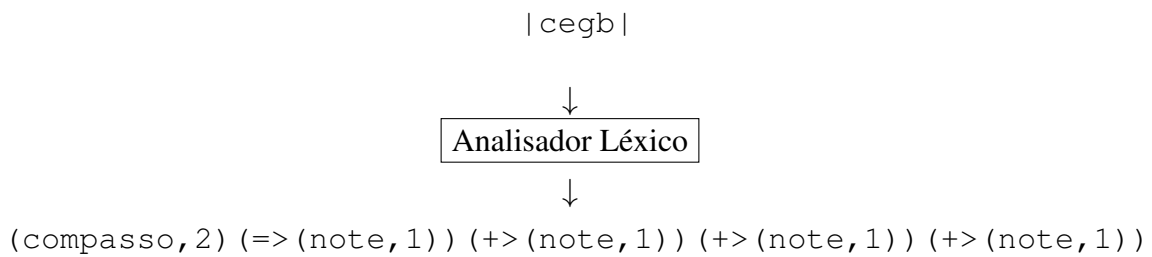
**Figura 4: Código de exemplo na linguagem Abc.**

O código apresentado na Figura 4 ao passar pelas etapas do front-end, em sua etapa inicial, lançaria e instalaria tokens correspondentes aos campos que definem o contexto para análise sintática e semântica, além de tokens que denotam título, compositor e andamento. O campo L:1/4 e o campo M:4/4, definem respectivamente a unidade de uma semínima como menor duração de uma nota no compasso e a quantidade máxima e mínima de 4 semínimas para completo preenchimento do compasso. O campo K: define a armadura de clave. Os demais campos são irrelevantes para análises no front-end.

Os demarcadores de compasso representados pelo carácter “|”, definem se a regra de compasso será ou não aplicada naquele compasso. Convenciona-se então que apenas o primeiro compasso da peça musical não precisa ter seu início delimitado pelo símbolo de compasso, permitindo assim anacruse e compassos acéfalos.

Como erros léxicos, temos a identificação de lexemas que não pertencem a gramática da notação simbólica. Se um lexema encontrado não se encaixa nos padrões definidos pelas expressões regulares, este é rejeitado e uma mensagem de erro é emitida, a figura 5 exibe tokens e atributos identificados como corretos e que serão instalados na tabela de símbolos.

Com o contexto definido e devidamente representado em tabela de símbolos, temos que a menor unidade que representa a duração de uma nota presente na partitura é a semínima. Qualquer alteração que seja necessária na duração da nota, deverá ser sinalizada através de operações matemáticas que são representadas na notação Abc de acordo



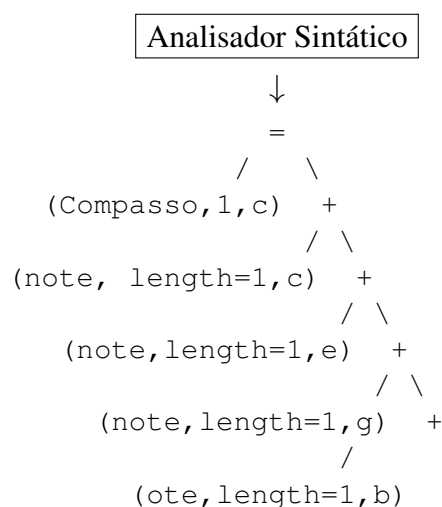
**Figura 5: Tokens de lexemas identificados.**

com a ordem da construção do lexema da nota musical e seus atributos. A construção “d/2” por exemplo, denotaria a divisão de semínima por 2, que resultaria em uma colcheia.

## Análise Sintática

A validação sintática não diz respeito aos símbolos reconhecidos mas à combinação destes símbolos para a formação de sentenças mais complexas. No caso da notação musical, erros sintáticos podem ser entendidos como combinações inválidas de símbolos válidos para a criação de compassos (sentenças). Assim, seriam erros sintáticos: ter 4 semínimas em um compasso 4/4, dois símbolos de sustenido ou bemol em uma mesma linha, um símbolo de bequadro em uma linha sem sustenido ou bemol, duas claves em sequência em um compasso, duas fórmulas de compasso em sequência em um compasso entre outros. Erros sintáticos podem ser definidos ainda por campos que recebem informações de contexto incompletos ou a inexistência destes campos, pois impossibilitam a percepção de contexto para análise semântica. A falta de informação a cerca de métrica por exemplo não permite que análise semântica verifique se a quantidade de notas existentes em um compasso é compatível com a fórmula de compasso.

Com os tokens devidamente instalados em tabela de símbolos, a análise sintática transforma a entrada em uma estrutura de dados conveniente para processamento posterior. Normalmente esta estrutura é uma árvore de derivação sintática, como apresentada na Figura 6.



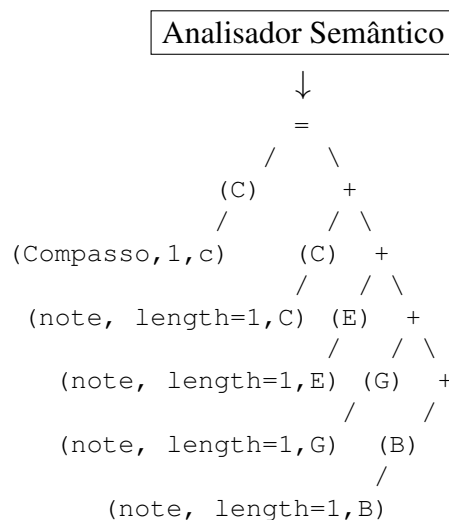
**Figura 6: Árvore sintática.**

## Análise semântica

A análise semântica deve encontrar sentenças corretas mas cuja ordem não possui um significado válido. Na linguagem de programação C, um *return* como primeira linha de um método *main* está sintaticamente correto mas é semanticamente incorreto.

No caso de notação musical, esta análise poderia basear-se em verificações de sentido musical que podem ser: correção de vozes em um contra ponto, verificação onde as notas presentes no compasso pertencem a escala representada na armadura de clave, verificação de estilos e ornamentos do estilo, verificação de extensão musical para partituras de determinado instrumento, verificação da existência de acordes para instrumentos solistas, entre outras.

É fácil notar que nenhum dos erros semânticos acima apresentados é realmente considerado um erro de escrita musical ou um erro de partitura. Por esta razão, assumiremos que é possível implementar este analisador mas que não há definições bem definidas de erros semânticos por não haver consenso sobre isto. De qualquer maneira, a Figura 7 apresenta a verificação de clave para as notas de um compasso.



**Figura 7: Verificação de notas musicais de acordo com armadura de clave.**

Ao fim do processo a análise estará completa e código intermediário pode ser gerado.


### 3.2. Back-end

No compilador, o Back-end é o responsável por otimizar o código e gerar o programa em uma linguagem de máquina. Por esta razão, o back-end é dependente da arquitetura da máquina onde o programa irá executar. Esta dependência ocorre tanto para a geração de código específico quanto para a otimização já que é possível que o compilador otimize a execução tomando vantagens de detalhes da arquitetura como, por exemplo, paralelismo em máquinas com mais de um core ou a escolha de melhores instruções em máquinas de arquitetura CISC.

No caso de compilação musical, pode ser possível trabalhar as tarefas do back-end não para uma arquitetura específica mas para formatos específicos de notação simbólica. Isto permitiria que o compilador funcionasse não apenas como um validador de música simbólica mas também como um conversor otimizado entre formatos distintos.

## Geração de código intermediário

Uma primeira tarefa do back-end pode ser converter as árvores sintáticas e semânticas em um código intermediário. A representação em código intermediário será capaz de exibir informações de cada nota musical presente em cada compasso e pode ser totalmente independente da linguagem de entrada do compilador, como apresentado na Figura 8.



```
C noteC = 1;
C noteE = 1;
C noteG = 1;
C noteB = 1;
C compassoId2 = noteC + noteE + noteG + noteB;
```

**Figura 8: Código intermediário.**

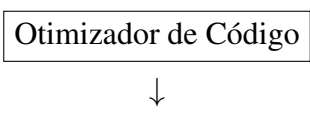
A representação de código intermediário se compara a estrutura de linguagem de programação onde a armadura de clave do compasso e nota pertencente a respectiva escala são do tipo "C", que denota que a armadura de clave está na escala de dó e as notas declaradas também fazem parte dessa escala.

## Otimização de código

A otimização de código em um compilador é dividida em duas partes sendo uma dependente da arquitetura e a outra independente. No nosso caso, a parte dependente poderá variar de acordo com o formato de saída a ser utilizado no compilador.

A otimização independente pode, por exemplo, aglutinar várias notas ligadas em uma nota única com o mesmo tempo. O mesmo pode ocorrer com diversas pausas utilizadas para preencher um compasso de silêncio. Estas pausas podem ser substituídas por uma única pausa deixando a leitura desta partitura mais concisa.

Já a otimização dependente do formato pode utilizar representações específicas de um determinado formato para simplificar a notação de um trecho musical. Esta otimização permite que o código fique menos extenso e mais legível para etapa de geração de código, conforme apresentado na Figura 9.



```
C arpejo = 3;
C compassoId2 = arpejo + noteB;
```

**Figura 9: Código otimizado.**





de erros sintáticos apresentado neste trabalho. A compilação do código da Figura 12 tornou um erro de sintaxe devido a quantidade de notas exceder o tamanho do compasso.

## 5. Conclusão

O uso da tecnologia de compiladores para a validação da notação musical padrão em linguagens simbólicas, se mostra atraente devido a similaridade entre a estruturação destas linguagens e as linguagens de programação. Algumas linguagens simbólicas se mostram bastante permissivas, e os compiladores destas linguagens apenas fazem verificação da notação simbólica, deixando a cargo do compositor respeitar a notação musical padrão. A abordagem exposta permite uma automatização na validação da notação musical padrão em processos de transcrição musical auxiliados por computador.

O código intermediário que pode ser gerado, será utilizado para síntese de representação gráfica livre de erros de notação e troca de formatos. A possibilidade da conversão de uma notação simbólica em outra se mostra bastante atraente, além de permitir segurança durante o processo de tradução.

Este trabalho ainda se mostra útil para verificação em recuperação de partituras através de **OMR - (Optical Music Recognition)**, onde partituras em seu estado físico, muitas vezes debilitadas pela exposição ao tempo, são analisadas através de técnicas de visão computacional e convertidas para notação simbólica afim de serem armazenadas e se tornarem obras editáveis. A abordagem aqui exposta permite a verificação desta representação simbólica em busca de erros ocasionados pela leitura ótica, que pode reconhecer erroneamente defeitos físicos como elementos de notação.

### 5.1. Trabalhos futuros

O próximo passo desta pesquisa é adicionar ao compilador outra linguagem, tanto para a entrada quanto para a saída, permitindo assim a) utilizar o compilador proposto para conversão entre formatos de música simbólica, b) validar o código intermediário com outro formato de entrada e saída, c) verificar a existência de outras regras (sintáticas e semânticas) que podem ser definidas em um outro formato de notação.

## Referências

- Aho, A. V., Lam, M. S., Sethi, R., and Ullman, J. D. (2008). *Compilers: Principles, techniques, and tools*, volume 2 ed.
- Ariza, C. and Cuthbert, M. (2011). *The music21 stream: A new object model for representing, filtering, and transforming symbolic musical structures*. Ann Arbor, MI: MPublishing, University of Michigan Library.
- Association, M. M. et al. (1996). *The complete MIDI 1.0 detailed specification: incorporating all recommended practices*. MIDI Manufacturers Association.
- Cuthbert, M. S. and Ariza, C. (2010). music21: A toolkit for computer-aided musicology and symbolic music data.
- Dannenberg, R. B. (1993). Music representation issues, techniques, and systems. *Computer Music Journal*, 17(3):pp. 20–30.
- Good, M. (2001). Musicxml for notation and analysis. *The virtual score: representation, retrieval, restoration*, 12:113–124.
- Hewlett, W. B. (1997). Beyond midi. chapter MuseData: Multipurpose Representation, pages 402–447. MIT Press, Cambridge, MA, USA.

- Johnson, S. C. (2001). Yacc: Yet another compiler-compiler.
- Lesk, M. E. and Schmidt, E. (2001). Lex - a lexical analyzer generator.
- Nienhuys, H.-W. and Nieuwenhuizen, J. (2003). Lilypond, a system for automated music engraving. In *Proceedings of the XIV Colloquium on Musical Informatics (XIV CIM 2003)*, volume 1. Citeseer.
- Oppenheim, I., Walshaw, C., and Atchley, J. (2010). The abc standard 2.0.
- Walshaw, C. (2011). The abc music standard 2.1.