FORMAL SEMANTICS FOR MUSIC NOTATION CONTROL FLOW*

Zeyu Jin
Carnegie Mellon University
School of Music, Pittsburgh, PA
zeyuj@andrew.cmu.edu

Roger Dannenberg
Carnegie Mellon University
Computer Science Department, Pittsburgh, PA
rbd@cs.cmu.edu

ABSTRACT

Music notation includes a specification of *control flow*, which governs the order in which the score is read using constructs such as repeats and endings. Music theory provides only an informal description of control flow notation and its interpretation, but interactive music systems need unambiguous models of the relationships between the static score and its performance. A framework is introduced to describe music control flow semantics using theories of formal languages and compilers. A formalization of control flow answers several critical questions: Are the control flow indications in a score valid? What do the control flow indications mean? What is the mapping from performance location to static score location? Conventional notation is extended to handle practical problems, and an implementation, Live Score Display, is offered as a component for interactive music display.

1. INTRODUCTION

Music notation has been evolving for centuries, creating a symbolic system to convey music information. Early music notation contained only lines and notes, which are sufficient for communicating pitches and durations. It was later that bar lines and time signatures emerged, grouping music into measures and introducing the idea of beats.[1] The notation for music control flow, like repeats and codas, came even later. Control flow helps to identify repeating structures of music and eliminates duplication in the printed score. In the Classical period, control flow notation is closely tied to music forms such as binary, ternary and sonata and is more of a musical architecture than a means of saving space.[2] Conventional practice for control flow notation is well established. The literature [6, 15] has formalized the notation in all kinds of ways and there is little conflict among definitions. However, traditional music theory has not explored the possibilities of expanded or enriched representations for control flow, and there is a gap between often simplified theoretical ideals and actual practice, especially in modern works. In practice, we find nested repeats, exceptions and special cases indicated by textual annotations, multiple endings, and symbols for rearrangement.

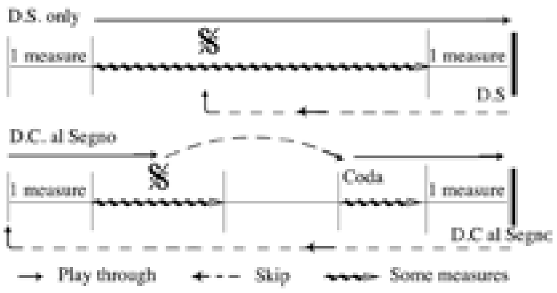


Figure 1: Control flow definition in Read's book

We encountered this gap between theory and practice in the implementation of music notation display software. We needed a formal (computable) way to relate notation to its performance, and we found conventional notions too limiting to express what we found in actual printed scores. To address this problem, we developed new theoretical foundations based upon models of formal language and compilation, and we applied these developments to the implementation of a flexible music display system.

Music control flow is the reading order of measures affected by control symbols including the time signature, measures, repeats, endings, etc. It can also be viewed formally as a function f that maps the performed beat k to a location of a score, $<m,b>$, a measure and beat pair. $f(k)$ describes the reading order of the score. In principle, we can rewrite the score in the order $f(1)$, $f(2)$, $\ldots$ to create an equivalent score with no control flow (other than reading sequentially). We call this the "flattened score" or "performance score." Audio recordings and MIDI sequences are both in the order of the flattened representation of the corresponding score.

Existing music theory devotes little attention to control flow, and in fact, there does not seem to be even a standard term for the concept of control flow. To define the meaning of control flow symbols, the conventional practice is to use words and visuals to illustrate the reading order. For example, Read uses arrows to mark the true reading order (see Figure 1) [15]. This approach defines both the syntax and meaning.

[1]Far beyond formalizing the notion of beats, music notation led to the "discovery" of time as an independent dimension that did not depend upon physical actions. In particular the musical rest is the first direct representation of "nothingness" existing over time, or of time itself. Composers developed this concept centuries before the scientific revolution, Kepler, Newton, graphs with a time axes, etc. [3]

[2]For example, "In practically all the sonatas of the earlier period the exposition is repeated, as is indicated by the repeat-sign at its end, which is also helpful for the reader in finding the end of the exposition ..." [2]

*Zeyu Jin and Roger B. Dannenberg, "Formal Semantics for Music Notation Control Flow," in *Proceedings of the 2013 International Computer Music Conference,* Perth, Australia, August 2013, pp. 85-92.

However, the visual definition is incomplete; there are cases where we are not sure which production to use. One such case is nested repeats where there are two ways of grouping the right repeat with the left repeat. In common practice music, we often restrict the number of levels of repeats to be 1, which excludes the nested repeat problem. In more modern and flexible music practice, there are nested structures, text annotations [8] and repetitions conditioned on actual time [17]. There lacks an unambiguous way to formalize the syntax and the meaning of such notation. While nested repeats are relatively simple to formalize, we argue that the general problem of formalizing control flow is non-obvious, interesting, and useful.

In this paper, we present a new framework for formalizing control flow notation based on formal grammar and compilation. One core feature of this framework is that it can be easily designed by humans and applied automatically by computer. As an example, we show a formal definition of control flow notation that unifies conventional music notation and some of the most frequent notations used in modern music practice. We evaluate the ambiguity and completeness of this definition within our framework. Finally, we show the implementation of this method and its application.

2. RELATED WORK

The score representation and interpretation is essential to score writing, printing, auto-accompaniment and conducting systems. Most works in this field focus mainly on storing and visualizing the score so as to accommodate both traditional and modern music practice, and, as a minor concern, provide room for implementing correct and user-friendly score play-back systems. Early studies such as MuseData [9] and Common Music Notation [16] mainly concern the encoding of conventional measure-based scores in their original notational presentation. To further encode logical structure of the score and support a wide range of modern presentations, structured music description languages like MusicXML [12] offer standards for score sharing and archiving. More recently, commercial score printing software like Finale[3] and Sibelius[4], and score accompaniment software like SmartMusic[5] can generate a music performance from notation. However, the algorithms that map the score to performance are problematic when nested repeats are involved. Given that popular score editors attempt to perform scores by following control flow, it is surprising that they do not define a syntax for control flow, check scores for validity, or offer a consistent meaning for control flow notation.

More recent study with an emphasis on music control flow modeling led us to develop a score player for HCMP [7]. This work provides data structures and algorithms for finding the dynamic score (similar to the "flattened score" in this work) by unfolding the static (original) score presentation. This work also draws attention to *arrangement* in which performance order is specified as a list of section labels (e.g. "play the intro, verse 1, verse 2, chorus, chorus"), a common practice in popular music. However, the algorithms in this work do not handle nested repeats or provide a systematic way to validate the correctness of notation. Overall, previous work has made great contributions to the notational presentation of the score, while the basic questions of music control flow, such as "what notation is right" and "what is the correct performance order given this notation" are still unanswered. In this work, we provide a systematic scheme to define the "right" notation and produce the flattened score representation without ambiguity.

[3] Finale music notation software: http://www.finalemusic.com/
[4] Sibelius music notation software: http://www.sibelius.com/
[5] SmartMusic software: http://www.smartmusic.com/

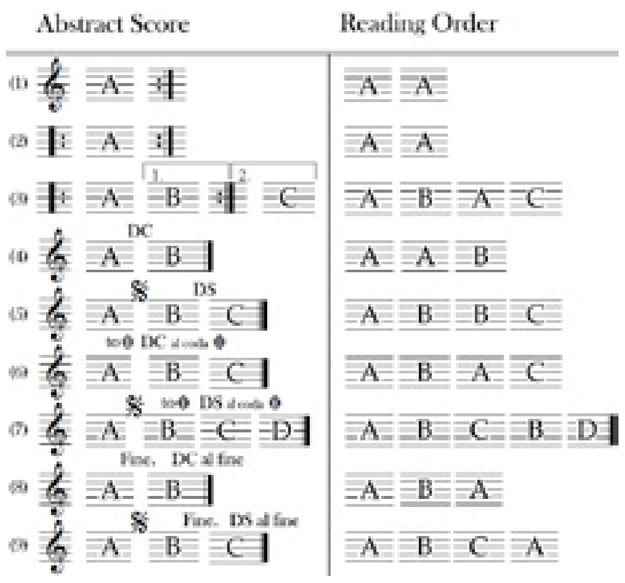

Abstract Score
Reading Order
(1) A | A A
(2) A | A A
(3) A B C 1 2 | A B A C
(4) DC A B | A A B
(5) DS A B C | A B B C
(6) to DC al coda A B C | A B A C
(7) to DS al coda A B C D | A B C B D
(8) Fine, DC al fine A B | A B A
(9) Fine, DS al fine A B C | A B C A

Figure 2: Control flow notation and the corresponding reading order in common music practice

3. COMMON MUSIC PRACTICE

Frequently used control flow symbols in common practice music are summarized in Figure 2. To avoid ambiguity, we often make the following restrictions:

- No control flow symbols are allow within the blank staff in Rule 1, 2 and 3.
- There can only be at most one type of DC/DS symbols in the entire music.
- Left and right repeats can be used within the blank staff in Rules 4-9.

4. EXTENDED MUSIC PRACTICE

In modern music practice, the control flow notation is much more flexible; there are many examples undefined in the conventional notation framework. Figure 3 shows some of the many unconventional examples found in *The Real*

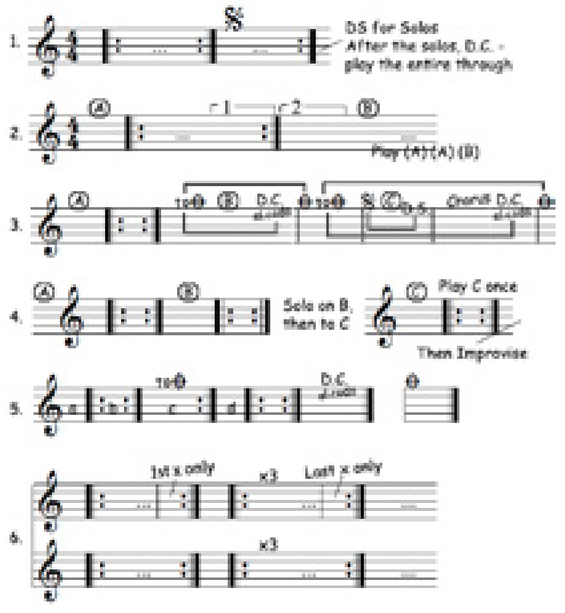


Figure 3: Control flow notation in modern music practice

Book [8]. The six examples[6] contain four typical notations in modern music practice:

Nested repeats (example 1 and 5).

Multiple and nested DS/DC-coda notation: In Example 3, the notation is complicated. The relation among D.C. al coda, to coda and coda signs are connected by lines.

Re-arranged sections (Example 2): re-order the section marks to create a different reading order of the score.

Word annotation (Example 1, 4 and 6): omitting or playing some measures conditioned on different rounds of repetition.

One could dismiss these particular scores as anomolies, but we find that in music, the exception proves the rule. Existing commercial software interprets Example 5 inconsistently. In Finale 2012 and Sibelius 7, the default reading order is a-b-b-c-b-b-c-d[7], but in MuseScore[8], the order is a-b-b-c-a-b-c-d. The correct order is a-b-b-c-a-b-b-c-d, which differs from both software interpretations. Clearly, we need a more sophisticated approach that offers a syntax and meaning that can be shared by both humans and computers. In the next section, we show such an approach based on formal language and attribute grammars. In Section 6, we show an even more flexible method "the intermediate notation" that supports extensions based on textual annotations.

[6]From 1 to 6: "Follow Your Heart" by J. Mclaughlin pp.154–155; "Forest Flower" by Charles Lloyd p.158; "Good Evening Mr. & Mrs. America" by John Guerin p.176; "Group your own" by Keith Jarrett p.182; Little Niles by Raudy Weston p.267; "Mallet Man" by Gordon Boek p.282

[7]One can notate nested repeat by manually specifying which measure the right repeat goes to.

[8]MuseScore music notation software: http://musescore.org/

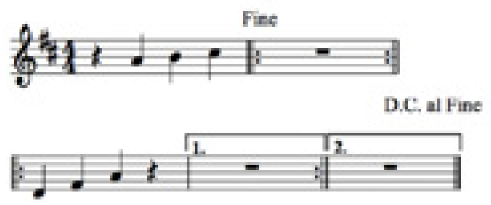


Figure 4: Symbolizing the score: this score is converted to the following list of strings to represent control flow: (block, 0, 4) Fine ||: (block, 4, 4) :|| ||: (block, 8, 4) [(block, 12, 4)] :|| [(block, 16, 4)] DC.Fine

5. MODEL FOR EXTENDED MUSIC PRACTICE

The framework for defining music notation control flow takes three steps: First, abstract the score into symbols; next, define the syntax for a well-formed score based on formal grammars; finally, define the meaning of each grammar using meta expressions. From this, we obtain the mapping function $f(k)$.

5.1. Symbolize the Score

For both convenience and clarity, we first convert the visual score to a list of symbols, each corresponds to a notation or a block of notations in the score. Table 1 shows a list of these symbols. To separate the problem of control flow semantics from the (much) larger and more general problem of music notation semantics, we simply label each block of notation between control symbols and assume the meaning and duration of a block is known.

The procedure is to read the score from left to right, write down each control flow symbol and name the intervening blocks as shown in Figure 4.

Table 1: Symbols used to label Control flow notation

Symbol	Meaning
(b, k, l)	A block of score, starting from beat k with length l
||: and :||	Left Repeat and Right Repeat
[	Beginning of an ending
]	Endpoint of an ending
DC	*dal Capo*
DC.Coda	*D.C. al Coda*
DC.Fine	*D.C al Fine*
DS	*Del Segno*
DS.Coda	*D.S. al Coda*
DS.Fine	*D.S. al Fine*
Segno	the *Segno* sign
Coda	the *Coda* sign
ToCoda	To Coda (or *Coda* symbol)
Fine	*Fine*

5.2. Context-free Grammar

A context free grammar (CFG) [1] is used to formalize the valid sequences of music symbols. A grammar is a set of rules that describe how a string (or a sequence of symbols) is formed. A context-free grammar (CFG) consists of terminal, nonterminal, productions, and a start symbol. In a CFG, one nonterminal is distinguished as the start symbol. The start symbol is replaced according to productions. After replacement, any remaining nonterminal can be replaced according to productions, and so on, until only terminals remain.

For example, we can define S as a whole score and E as a score element. Then we can define the following: A score S is a sequence of 1 or more musical elements (E), optionally followed by a right repeat. A grammar for this language (set of scores) is:

```
Starting: S
  (G1) S -> E
  (G2) S -> E :|
  (G3) E -> E E
  (G4) E -> b
```

Here, we consider "b" to be a terminal denoting any block, although we could add productions that expand "b" (now a nonterminal) to terminals of the form "(block *start duration*)." Based on this CFG, we are able to tell if a sequence of symbols is formed from this grammar using derivation. For example, `(b,0,4)(b,4,4):`‖ is well-defined from the grammar because we can derive it from S using the productions:

```
S => E :|
  => E E :|                   [use E -> E E]
  => (b, 0, 4) (b, 4, 4) :|   [use E -> b (twice)]
```

We say that a sequence of symbols is a *well defined* score if it can be derived using a grammar. The result of a derivation is a tree structure where each score symbol is a leaf and where each parent node and its children corresponds to a production in the grammar. The leaves from left to right (or more precisely in preorder traversal) will generate the input sequence. The tree is called a parse tree in compilation theory. As an example, the tree for `(b,0,4)(b,0,4):`‖ is shown in the Figure 5.

A program that converts a sequence of symbols into a parse tree is called a parser. A parser essentially runs the grammar "backwards," reducing a string of nonterminals to the start symbol by applying productions in reverse. Many parsing algorithms have been developed for grammars of different complexity. LR(1) [10] and LALR(1) [14] are the most frequently used parsers for programming

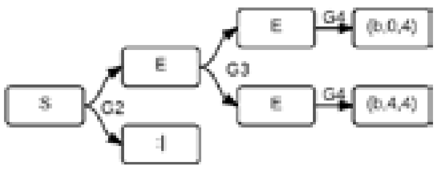


Figure 5: Parse tree for (b,0,4) (b,4,4) :‖

languages. However, not all context-free grammars can be parsed by LR(1). Since LR(1) parsers are balanced in generality and computational efficiency, we often design grammars acceptable to LR(1) parsers.

5.3. Ambiguity and Error Handling

The problem of ambiguity arises when there are multiple parse trees for the same the sequence of symbols and the same grammar. For example, if we change production (G2) to E → E : |, we get an ambiguous grammar. The score b : | b : | can be generated two ways:

```
S => E => E E => E :| E :| => b :| b :|
S => E => E :| => E E :| =>
       E b :| => E :| b :| => b :| b :|
```

The first derivation (line 1) implies a sequence of two repeated blocks. The second derivation (line 2) implies nested repeats. Notice that while semantics are not inherent in formal grammars, we often associate semantics closely with productions and parse trees.

5.4. Syntax Directed Translation and Attribute Grammar

A syntax-directed definition is a context free grammar together with attributes and rules. The attributes are associated with grammar symbols and the rules are used to define the relation among symbols [13]. For any terminal or nonterminal X, denote $X.a$ as some attribute of X. For a production like $X \rightarrow YZ$, the rule can be $X.a = op(Y.b, Z.c)$ which contains some attribute symbol, an equal sign and some expression op of $Y.b$ and $Z.c$.

Define attribute *code* as the flattened presentation for all the terminals and nonterminals in our formal control flow framework. Define "|" as the concatenation operator that links two lists of symbols to be one. For example, if $E1.code$ = (b,0,4) (b,4,4); and $E2.code$ = ‖: (b,8,4) :‖, then $E1.code$ | $E2.code$ = (b,0,4) (b,4,4) ‖: (b,8,4) :‖. Furthermore, define attributes "beat" and "dur" to be the starting beat and duration for any terminal or nonterminal. For any block b, assume b.beat is the "starting beat" of b and len is the "length" of the block. The syntax directed definition for the simple grammar (G1)–(G4) can be defined as follows:

```
Starting: S
  (G1) S -> E    {
    (1.1) S.code = E.code
    (1.2) S.beat = E.beat
    (1.3) S.dur = E.dur            }
  (G2) S -> E :| {
    (2.1) S.code = E.code | E.code
    (2.2) S.beat = E.beat
    (2.3) S.dur = E.dur + E.dur    }
  (G3) E -> E E  {
    (3.1) E.code = E1.code | E2.code
    (3.2) E.beat = E1.beat
    (3.3) E.dur = E1.dur + E2.dur  }
  (G4) E -> b {
    (4.1) E.code = (b, b.beat, b.dur)
    (4.2) E.beat = b.beat
    (4.3) E.dur = b.dur            }
```

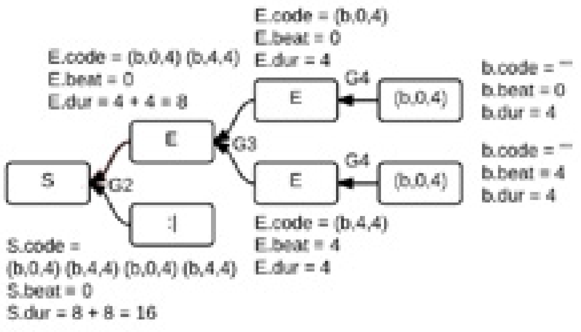


Figure 6: Data flow in syntax directed translation

Rule (G1) says the flattened representation of a score can be a flattened score element with the same starting beat and duration. Rule (G2) says if the score has a repeat, then the flattened score is two copies of the score element before the repeat sign with the starting beat equal to that of the score element and the duration equal to two times the duration of the score element. Rule (G3) says if a score element is formed by two sub-score elements, then the flattened element is made by concatenating the flattened representation of sub-elements, and the duration is the sum of the lengths of the sub-elements. Finally, if an element is a single block, then all the attributes of the block are copied to the element (G4).

Based on the parse tree, we can apply rules from leaves to root in order to synthesize the attributes of upper levels. For the example shown in Figure 5, the attribute flow is shown in Figure 6.

Finally the flattened score is in *s.code* and the duration of the entire score is 16 beats. Although this is a simple example, notice that we have precisely and unambiguously specified how to interpret score control flow notation. Moreover, we have introduced a *meta-notation* based on attribute grammars that allows us to define other interpretations of score control flow.

5.5. The Mapping Function

One goal of defining control flow is to obtain a mapping from beats in a performance of the score (i.e. the *flattened* score) to positions or blocks in the original score. We denote the mapping as $f(k)$, where k is the beat position in the flattened score (S.code) and $f(k)$ is the position in the score. The map can be constructed by summing beat durations in the flattened representation S.code to obtain k for each block in the original score.

5.6. Well Defined Music Control Flow

As a demonstration of how we can use this framework to formalize music control flow for extended music practice, the following grammar supports nested repeats and unlimited endings. An even more sophisticated implementation is used in a newly implemented score display system which can model word annotation and section-based arrangement. Because space is limited, we only show DS.al.coda in the DS/DC family and limit our attention to the ".code" attribute only.

Define nonterminal S as the score, L as the left-most part of the score, E as score element, LEND as the ending within L and ND as the ending. Add two additional symbols to the original score: # at the beginning and $ at the end.

The score S: to be consistent with case 1, the score needs to be decomposed into an L, the leftmost part that could have multiple right repeats, and an E followed by the ending sign $. This gives us

```
S -> L $ {
  S.code = L.code;                              }
S -> L E $ {
  S.code = L.code | E.code;                     }
S -> L Segno E ToCoda E DS.coda Coda E $ {
  S.code = L.code | E0.code | E1.code |
           E0.code | E2.code;                   }
```

The leftmost group L: If there are no non-paired right repeats in the score, L should be an E. Or this L can produce more L, right repeat sign followed by an E and also an ending structure, the leftmost multiple endings.

```
L -> L :| E {
  L.code = L1.code | L1.code | E.code;          }
L -> # E {
  L.code = E.code;                              }
L -> L :| {
  L.code = L1.code | L1.code;                   }
L -> LEND E {
  L.code = LEND.code | E.code;                  }
L -> LEND {
  L.code = LEND.code;                           }
```

The score element E can be a simple block, a repeated structure, an ending or a DS.al.Coda structure.

```
E -> b {
  E.code = (b, b.beat, b.dur);                  }
E -> |: E :| {
  E.code = E1.code | E1.code;                   }
E -> |: E ND [ E ] {
  For n = 1 to ND.count
    E0.code = E0.code | E1.code
            | ND.ending[n];
  E0.code = E0.code | E1.code | E2.code;        }
E -> E E {
  E0.code = E1.code | E2.code;                  }
E -> E Segno E ToCoda E DS.Coda Coda E $ {
  E.code = E1.code | E2.code | E3.code |
           E2.code | E4.code;                   }
```

ND and LEND: Because of the complication of this structure, we need to record each ending to the top level by creating a new attribute ND.ending with a list structure. We also need to use a loop to pair different endings with the constant part.

```
LEND -> # E ND [ E ] {
  For n = ND.count downto 1
    LEND.code = LEND.code | E0.code
              | ND.ending[n];
  LEND.code = LEND.code | E0.code | E1.code; }
ND -> ND [ E :| {
  ND.count = ND1.count | 1;
  ND.ending[ND.count] = E.code;                 }
```

```
ND -> [ E :| {
  ND.count = 1;
  ND.ending[1] = E;                        }
```

It can be shown that this grammar can be parsed by LR(1) without ambiguity ("Reduce-reduce" error or "Shift-shift" error). "Reduce-shift" errors occur but can be solved by "preferring reduce then shift."

6. IMPLEMENTATION

We implemented a Java-based API called MCFC (music control flow compiler) that creates a score compiler from a user-defined lexical definition and grammars at start-up and translates string-format score symbols to a flattened score based on the given grammar. Unlike lex and yacc [11], which are used frequently for generating code frameworks for making compilers, MCFC generates the entire compiler directly from the CFG and attribute grammar, and users can switch among different grammars without closing the application. This feature is especially useful when dealing with notation from different domains.

MCFC does not assume any model of the score and works in the string representation. The first step for any application to have this API embedded is to convert the score to a list of symbols as demonstrated in Section 5.1. To make the compiler understand the semantics of these symbols, the user needs to provide an additional file declaring all the symbols, which is known as lexical definition, and is accomplished using regular expressions similar to those used in lex [11].

6.1. Grammar Definition

The grammar definition is put in an additional file with extension ".g." The content of the grammar definition is essentially identical to the attribute grammar shown in Section 5.6.

6.2. The Intermediate Grammar

The score compiler supports a built-in grammar that uses a small set of instructions to make control flow more flexible. The basic instructions are shown in Table 2.

Table 2: Default Intermediate Language Grammar

Symbol	Meaning
(&, <CV>, <N>)	A section mark; $\boxed{A2}$ is (&,A,2)
(loop, L<N>)	A loop mark labelled by L, similar to left Repeat
(rep, L<N>, T<N>)	Repeat to Label L for T times, similar to Right Repeat
(lb, B<N>)	A Jump mark labelled by B
(jmp, L<N>, T<N>, B<N>)	jump to B at the T-th repetition for loop L

The intermediate grammar can be used to annotate word-defined scores as in Figure 3. It is very useful to compile the original score to this intermediate form and then use the built-in translator to further compile it to a flattened score. The compiler for the intermediate grammar has sophisticated functions such as loop mapping, which helps to relate static score positions back to multiple performance positions (k in $f(k)$). For example, one might want to select score position as it occurs in the "2nd repeat after the D.S." For example, consider the score

```
(&,A,1) |: |: (b,0,4) :| (&,A,2) (b,4,4) :|
```

which is translated into intermediate presentation

```
(&,A,1) (loop,0) (loop,1) (b,0,4)
(rep,1,2) (&,A,2) (b,4,4) (rep,0,2)
```

and finally to the flattened score

```
(&,A,1) (b,0,4) (b,0,4) (&,A,2) (b,4,4)
(b,0,4) (b,0,4) (&,A,2) (b,4,4)
```

Output symbols are marked with labels (below) called flags that show the mapping from the symbol to loop. The format is [L1,count1;L2,count2,...] where count$n$ is the number of repetitions of loop Ln. Notice that (b,0,4) appears with four distinct loop labels

```
[] [L0,1;L1,1] [L0,1;L1,2] [L0,1] [L0,1]
[L0,2;L1,1] [L0,2;L1,2] [L0,2] [L0,2]
```

6.3. Rearrangement

The other feature of this built-in translator is its ability to handle rearrangement. In the example above, the score is separated into subsections by section marks.

```
A1-[]      (b,0,4) (b,0,4)
A2-[L0,1] (b,4,4) (b,0,4) (b,0,4)
A2-[L0,2] (b,4,4)
```

One can input a rearrangement based on the section marks and the loop mapping.

```
A1-[] A2-[L0,2] A1-[]
```

Then the flattened score is

```
A1-[]      (b,0,4) (b,0,4)
A2-[L0,2] (b,4,4)
A1-[]      (b,0,4) (b,0,4)
```

In this way the rearrangement notation used in Figure 3 can be compiled.

7. APPLICATION

The score compiler is used in a music score display application that, in turn, can be used as part of a human-computer interface for score following, human computer music performance, music education, multimedia databases, etc., where the correct reading order is essential. As a demonstration, we built a system called Live Score Display (LSD for short) where performers import score images, annotate control flow symbols, re-arrange the score and play the score in real time. With the flattened score representation, the user can browse to any repetition of a repeat, even in heavily nested repeats.

7.1. Score Annotation

Ideally, scores would all be machine readable with consistent control flow notation. In reality, we often work with scanned score images, and optical music recognition (OMR) would at best require extensive manual editing to produce usable data. Our solution is to import scanned or photographed score images and manually annotate the elements that are critical for control flow and display.

The score annotation interface is provided to import score images and annotate the layout of the score and the control flow symbols. This interface is shown in Figure 7a. There are four main panels: the leftmost contains the score structure tools, which are used to draw the system boundary, barlines and places between the barlines where control flow symbols occur. These places and barlines are called time points. Next to the structure toolbar is the symbol toolbar, which is used to place symbols on time points. In the middle is the notation panel where users add annotations. The rightmost panel is the navigator where a small-size score is shown along with the string-format symbols that are used in the score compiler.

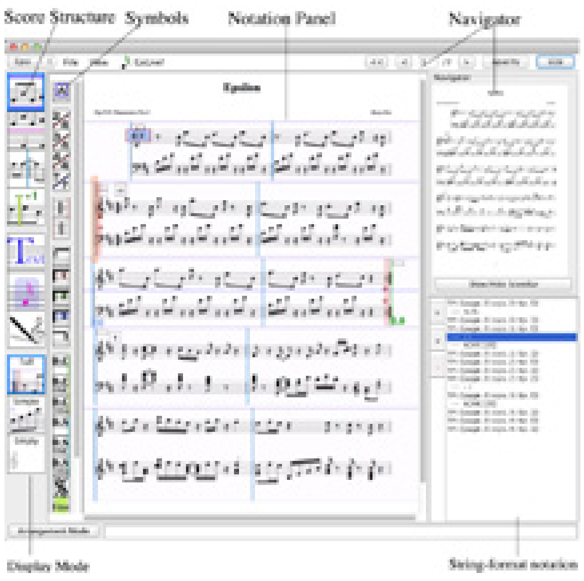
To compile the score and enter the arrangement mode, one can press the "GoLive" menu on the menubar. If there are notation errors, a prompt window will display an error message and suggestion. To switch between different grammars, one can find "switch grammar file" and "switch lex file" in the "GoLive" menu.

7.2. Arrangement Mode

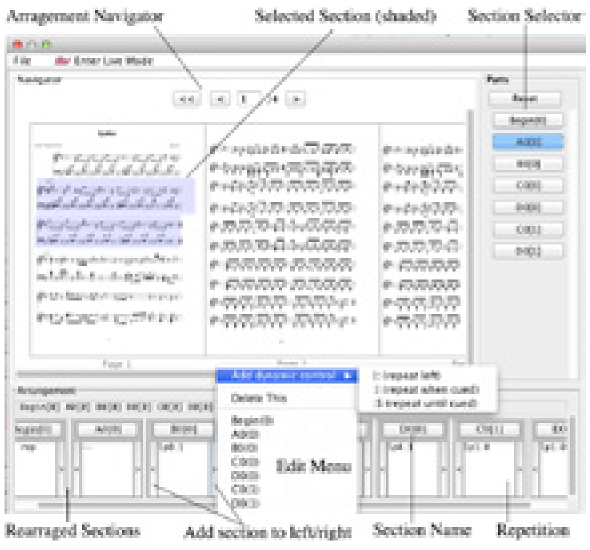
When the score is compiled, the program enters "arrangement mode" as shown in Figure 7b. The arrangement editor features a navigator which highlights the selected section in the "section selector" panel. In the bottom is the section panel where one can insert or delete more sections. The sections are shown in small square widgets with the name and corresponding loop marks (see Section 6.2). One can also add dynamic controls between sections, which are special repeat signs that can be dynamically controlled in the performance mode to model "repeat until cue" directions.

7.3. Performance Mode

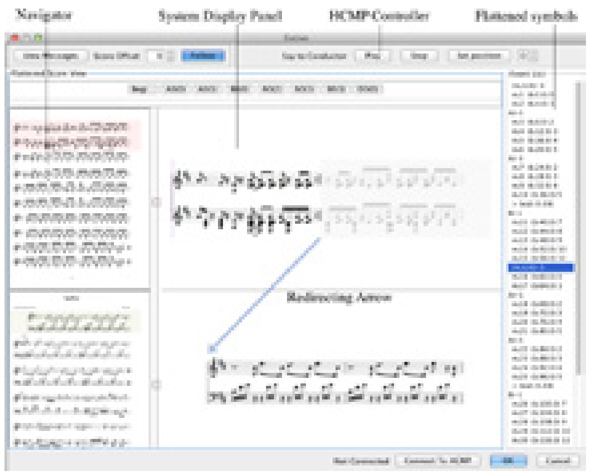
After the score is arranged, the user can enter Live mode and set up an HCMP conductor [4, 5] to play with a band. The HCMP conductor synchronizes sequencers, audio, video, and other media to live performers by broadcasting score position and tempo changes via OSC signals [18]. LSD registers with the conductor as a "player" object and displays the score's current and next system to the performer in a manner similar to the lyric display in a Karaoke system. When the control flow jumps, an arrow is shown to point out the direction and unplayed measures are shaded (Figure 7c).



(a) Notating Mode



(b) Arragement Mode



(c) Performance Mode

Figure 7: Live Score Display user interface

8. SUMMARY AND CONCLUSIONS

The interpretation of control flow symbols in music notation is an intricate problem that existing music theory solves only in a highly simplified and informal manner. We have described a framework that borrows from formal languages, formal semantics, and compiler theory. The framework allows us to describe precisely what scores have valid control flow directives and what these directives mean. Furthermore, we showed that we can model modern practices such as nested repeats and textual directives that have no conventional control flow notation. We can also model the practice we call "arrangement" in which additional directives override control flow conventions to perform the score in a new sequence, including on-the-fly arranging such as vamping a section until cued or taking an optional cut.

Beyond theory, we created a flexible implementation for displaying music notation in live performance, mapping the performance position to the score position, looking ahead to page turns and non-sequential jumps in the score notation, and allowing the user to add annotations and arrangements. The Live Score Display system has a meta-notation system in which new control flow syntax can be introduced.

In the absence of interactive computer music, control flow semantics is mostly a theoretical problem. Regardless of theory, the "meaning" of a score is really whatever the performers choose to perform. However, the problem is much more concrete in the context of interactive systems. Computers and humans must agree on control flow semantics if we expect to perform conventional scores with computers. Computers can "understand" scores or we can "tell" computers what they mean. Either way, we need formal descriptive models to express our intentions. This work offers an initial step toward this goal.

9. ACKNOWLEDGEMENTS

The authors would like to thank Nicolas Gold, Gus Xia, and Dawen Liang for discussions and earlier work that set the stage for this research. This work was partially supported by the National Science Foundation under Grant No. 0855958.

10. REFERENCES

[1] A. Aho, M. Lam, R. Sethi, and J. Ullman, *Compilers: Principles, Techniques and Tools*, 2nd ed. Prentice Hall, 2007.

[2] W. Apel, "Harvard Dictionary of Music," in *Harvard University Press*. Harvard University Press, 1970, p. 696.

[3] A. W. Crosby, *The Measure of Reality: Quantification and Western Society, 1250-1600*. Cambridge University Press, 1997.

[4] R. B. Dannenberg, "New Interfaces for Popular Music Performance," in *International Conference on New Interfaces for Musical Expression*, New York University, New York, Jun. 2005, pp. 130–135.

[5] ——, "A Virtual Orchestra for Human-Computer Music Performance," in *Proceedings of the International Computer Music Conference 2011*, San Francisco, Aug. 2011, pp. 185–188.

[6] T. Gerou and L. Lusk, "Repeat Signs," in *Essential Dictionary of Music Notation*. Alfred Pub Co, 1996, p. 110.

[7] N. Gold and R. Dannenberg, "A reference architecture and score representation for popular music human-computer music performance systems," in *International Conference on New Interfaces for Musical Expression*, Oslo, Jun. 2011, pp. 36–39.

[8] Hal Leonard Corporation, *The Real Book: Sixth Edition*, 6th ed. Hal Leonard Corporation, 2004.

[9] W. B. Hewlett, "MuseData : multipurpose representation," in *Beyond MIDI*. MIT Press Cambridge, Oct. 1997, pp. 402–447.

[10] D. Knuth, "On the translation of languages from left to right," *Information and control*, vol. 9, pp. 707–639, 1965.

[11] J. Levine, T. Mason, D. Brown, and B. Cupac, *Lex & yacc*, 2nd ed. O'Reilly Media, Oct. 1992.

[12] Michael Good, "MusicXML for Notation and Analysis," in *The Virtual Score: Representation, Retrieval, Restoration,*, Walter B. Hewlett and Eleanor Selfridge-Field, Ed. Cambridge, MA: MIT Press, 2001, vol. 12, pp. 113–124.

[13] J. Paakki, "Attribute Grammar Paradigms Language Implementation A High-Level Methodology," *ACM Computing Surveys (CSUR)*, vol. 27, pp. 195–255, 1995.

[14] D. Pager, "A practical general method for constructing LR(k) parsers," *Acta Informatica*, vol. 7, no. 3, pp. 249–268, 1977.

[15] G. Read, *Music notation a manual of modern practice*. Boston: Allyn and Bacon, Inc., 1964, pp. 224–231.

[16] B. Schottstaedt, "Common music notation," in *Beyond MIDI*. MIT Press Cambridge, Oct. 1997, pp. 217–221.

[17] K. Stone, *Music notation in the twentieth century: a practical guidebook*. W. W. Norton & Company, 1980.

[18] M. Wright, "Open Sound Control: an enabling technology for musical networking," *Organised Sound*, vol. 10, no. 3, p. 193, Nov. 2005.