A Transformational Grammar Framework for Improvisation

Alexander M. Putman and Robert M. Keller

Abstract— **Jazz improvisations can be constructed from common idioms woven over a chord progression fabric. Prior art has shown that probabilistic generative grammars are one effective means of achieving such improvisations. Here we introduce another approach using transformational grammars instead. One advantage that transformational grammars provide is a form of steering from an underlying melodic outline. We demonstrate by showing how idioms can be defined in a transformational grammar and how the placement of idioms conforms to the outline and chord structure. We illustrate how transformational grammars can provide unique and varied improvisations that are suggestive of the outline. We illustrate the real-time application of this approach in an educational software tool.**

Index Terms — **improvisation, grammar, transformation, substitution, idiom.**

I. INTRODUCTION

Teaching and learning improvisation is a topic of interest to jazz educators. There are several different theories, each with its own strengths ([1]-[3]). Here we follow the suggestion of Shelton Berg [1] that much of jazz improvisation is based on the use of common jazz *idioms*. In this paper, we use the word "idiom" to represent any of an assortment of ideas commonly found in jazz improvisations. These include various digital patterns, approach tones, enclosures, *etc*. [4]. The exact set included is open-ended due to the extensibility of the language we use to define them. The playing of some performers can be recognized by signature idioms. While professional players strive for novelty, it is widely understood that they also rely on a variety of practiced idioms as backup.

As we are interested in providing educational software tools that help users understand jazz solo improvisation, we have developed an extensible framework based on transformational grammars that permits both the definition and application of melodic *substitutions* representing jazz idioms to an otherwise plaintive solo or merely a chordal outline. For example, some jazz educators emphasize "guide-tone" lines, which are relatively static melodies that progress from the 7th scale degree of one chord resolving to the 3rd scale degree of the next, or the altered 5th degree to an altered 9th [5], [6]. Each substitution is defined as a set of one or more specific *transformations*, which specify how the idiom is realized in various harmonic and melodic contexts. A transformation is defined as a specification for rewriting a sequence of melody notes into a sequence that could be considered more interesting.

To show what our transformational grammar system is able to achieve, Figs. 1 to 3 show how an outline melody may be transformed. Fig. 1 shows a simple repeating pattern involving scale tones that are consonant with the accompanying chords. Fig. 2 shows an example of a melody created from the simple melody in Fig. 1 by transformations. The transformed melody is still consonant with the chord structure but can be regarded as more unique and "jazz sounding". The choice of transformations can include non-determinism. Fig. 3 shows a different result from the same substitution (transformation set). By understanding the concept of transformations, improvisation students can improve their repertoire of improvisation techniques.

Previous work on applying transformations to music is described in Section II. While the outlines to which our transformations can be applied are not limited in complexity, simpler outlines provide the most possibilities for added nuance. If an outline is too complex, the set of applicable transformations will be smaller. Outlines can be constructed by the user, generated by a separate generative grammar, or by "flattening" the melody of an existing solo. An advantage of the transformational grammar approach is that it allows new substitutions and their contained transformations to be defined and modified by the user.

As the substitutions (sets of related transformations for an idiom) are applied to different styles of jazz music, their definitions and ideal placement may change. As it is impossible to create a single set of substitutions for every style and type of music or every idiom, we devised a grammatical framework that allows for the creation of new transformations and for the modification of already existing ones. Just as importantly, our grammars have a readable textual representation, allowing students to follow along and understand how and when notes are being transformed.

While the transformational grammar approach is intended to be general enough to apply to basic musical definitions, we built a specific instance of a transformation system by augmenting Impro-Visor [7], an open source jazz solo generator and notation tool that already provides improvisation using generative grammars. Our work extends Impro-Visor by adding the ability to define, and then automatically apply, a transformational grammar.

II. RELATED WORK

Numerous researchers have discussed the possibility of improvisation using grammars. Johnson-Laird [8] discussed possible roles of grammar in jazz improvisation. Keller and

The authors are with Harvey Mudd College, Claremont, California, 91711 USA. This work was supported in part by NSF CISE REU award #1359170.

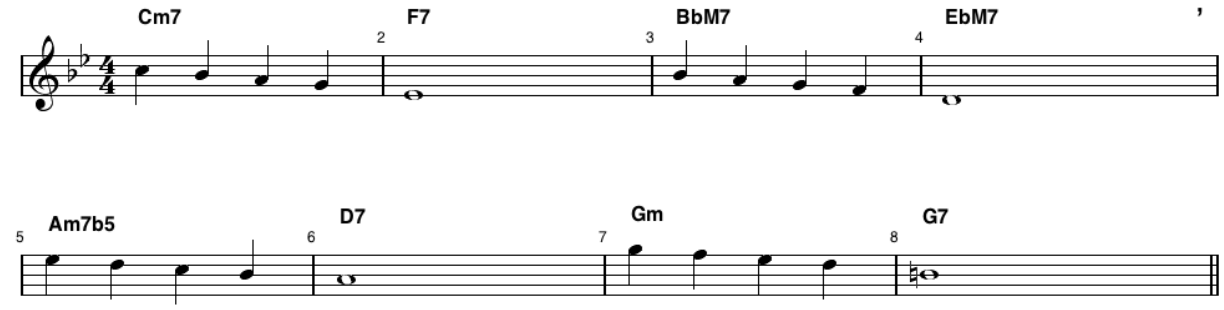


Fig. 1. Outline melody

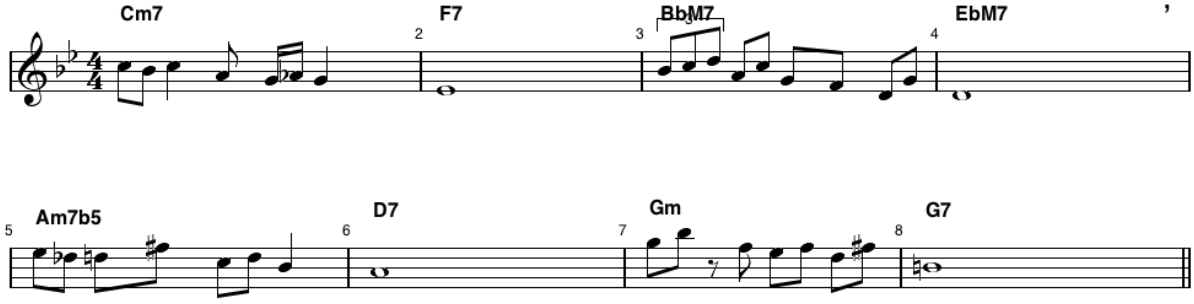


Fig. 2. Outline melody transformed

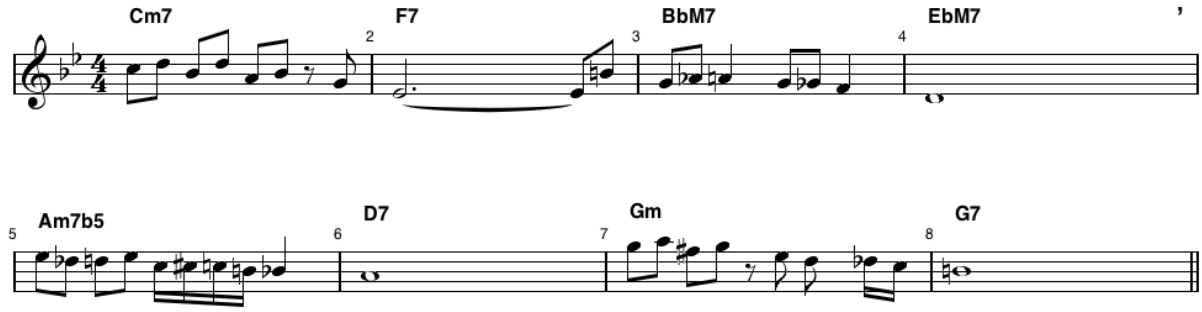


Fig. 3. One of many possible alternate transformed melodies

Morrison [9] described the use of probabilistic context-free grammars in generating jazz melodies. Gillick, Tang and Keller [10] described a method for machine learning of such grammars from solo transcriptions.

The cited works use grammars to *generate* improvisations *ab initio* based on chord progressions. In contrast, the present work proposes the use transformational grammars to *transform* existing melodies to create improvisations.

Transformations have been previously applied to music by transforming audio signals [11], evolving counterpoint using simple note transformations [12], and finding common rhythm and transposition motifs [13]. These methods provided modest changes to the melodic structure. As we seek to make melodic structures more complex, the rules for transformations need to take more information into account. Section III explains how we define these rules in a transformational grammar to achieve this goal.

III. SUBSTITUTIONS AND TRANSFORMATIONS

In our model, a transformation both defines how a group of notes is modified, as well as when it is appropriate to apply the modification. From an improviser's perspective, this could be viewed as having an awareness of a base outline and knowing how and when to change existing groups of notes.

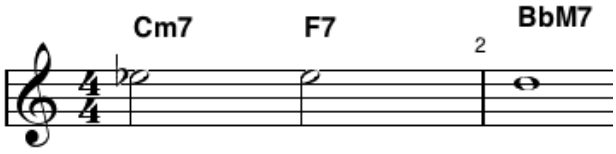
Tables I-II illustrate the possible effects of two distinct substitutions. In Section III.A, Tables III-IV show how we specify a transformation using a *textual* representation. Transformations also contain a *guard condition* that is used to determine applicability, as well as a *weight* that is used to determine the likelihood that the transformation will be applied. These are explained further in Section III.D.

A. Definitions of Transformations

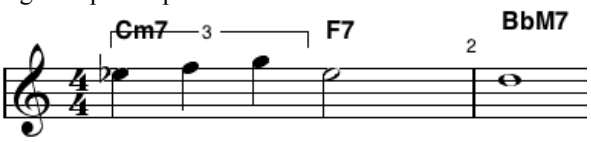
The transformative aspect of a transformation is expressed using two lists: *source-notes*, which are notes in the original melody, and *target-notes*, which are notes that replace them. For maximum generality, the notes in the source-note list are not absolute, but rather have variable names corresponding to each of the notes. For example, in Table III, there are two adjacent notes being transformed, the first being represented by *n1* and the second represented by *n2*. The target-notes will allow for definition of new notes to replace the source notes. Examples in Tables III-IV show how individual notes, such as *n1* and *n2*, are used to define the target notes.

Table I: A substitution based on half-notes with two of its transformations

source: split-half substitution



target: triplets-up transformation



target: triplets-down transformation

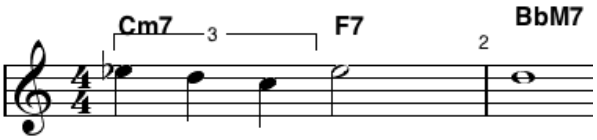
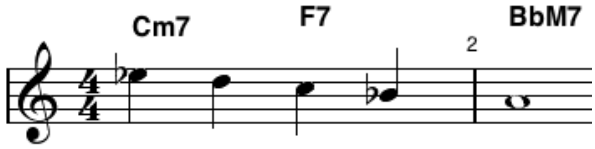
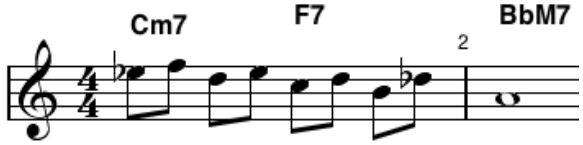


Table II: A substitution based on quarter notes, with some transformations

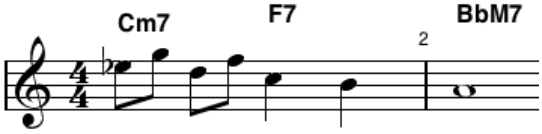
source: split-quarter substitution



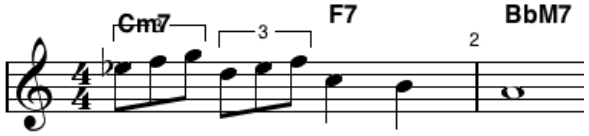
target: eighths-up transformation



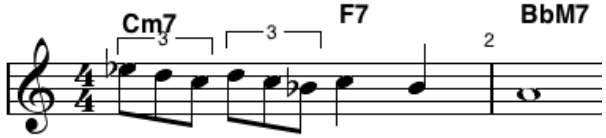
target: eights-up-third transformation



target: triplets-up transformation



target: triplets-down transformation



Note manipulation functions accept source notes and possibly other parameters and return transformed notes. In Table III, one can see that note manipulation functions *subtract-duration*, t*ranspose-chromatic*, and *set-duration* easily tell the reader how each note in the target is created.

After the note manipulation functions are evaluated, the notes in the target-notes list are inserted in place of the source notes. To ensure that a transformed melody will not be longer or shorter than its original length, the insertion is only done if the total duration of the notes in source-notes list equals the total duration of the notes in the evaluated target-notes list.

B. Guard Conditions

The source and target notes define how an idiom is created, but they do not dictate where it is placed. Placement is determined by the "guard-condition" of a transformation. The *guard-condition* is a Boolean-valued expression that determines whether or not a transformation will be applied to a sequence of source-notes. Note attributes include pitch and duration, as well as the chord over which it is being played. The guard condition typically checks certain attributes of the source-notes that are viewed to have a significant impact on the quality of the resulting target-notes.

Guard conditions are constructed from a base of music specific operators such as *duration*>, *pitch*<, and *chord*=, which may be combined using common relational operators, such as *and*, *or*, and *not*. Guard conditions may also use functions that return certain attributes of notes, such as whether they are part of the current chord, and attribute-manipulating functions that operate on attributes of notes.

Another reason for keeping notes abstract is to make the grammar rules as accessible as possible. Keeping notes as abstract variables allows for better flow in reading and understanding. In the examples of Tables III and IV, the grammar explains that the first note being transformed is *n1* and the second is *n2*. Reading through, one can then follow descriptive functions used on each note. Each time one sees a note variable used, it will be the same note. This permits a simplified mental model, defining "this is what we have", "should it be transformed?", and "this is how I want it transformed".

Table III: Textual specification of transformations for a grace-note transformation

```
(transformation
  (description single-ascending-tuple-grace-note)
  (weight 1)
  (source-notes n1 n2)
  (guard-condition
    (and
      (not (triplet? n1))
      (= (note-category n2) Chord)
      (duration>= n1 4)
      (not
        (and
          (rest? n1)
          (= (duration n1) 8)
          (duration>= n2 8)))))
  (target-notes
    (subtract-duration 16 n1)
    (set-duration 16 (transpose-chromatic -1/2 n2))
    n2))
```

Table IV: Textual specification of transformations for a triplet arpeggio transformation

```
(transformation
 (description ascending)
 (weight 1)
 (source-notes n1 n2 n3)
 (guard-condition
   (and
     (member (relative-pitch n2) (1 3 5))
     (= (duration n2) 4)
     (pitch< n2 n3)
     (not (= (chord-family n2) dominant))))
 (target-notes
   (subtract-duration 8 n1)
   (set-duration 8 (transpose-diatonic -2 n2))
   (scale-duration 1/3
     n2
     (transpose-diatonic 3 n2)
     (transpose-diatonic 5 n2))
   n3))
```

An application of a given transformation to a particular group of notes will always yield the same result, *i.e.* individual transformations are always deterministic. However, *substitutions*, as sets with possibly more than one transformation, are not deterministic. For a given substitution, the specific transformation selected depends on the relative value of the weights indicated for each transformation, in addition to a randomization element.

C. Substitutions

Substitution is the term we use to group together transformations that implement the same idiom. As transformations are implementations of idioms, substitutions can be considered to represent the idioms themselves. This allows us to categorize transformations and give specific attributes to idioms as a whole. These attributes include a name, weight, type and an arbitrary number of transformations. Table V shows how substitutions are structured in the textual notation.

Table V: Specifying an idiom that consists of multiple transformations. Ellipses are used to indicate suppressed details of transformations.

```
(substitution
  (name grace-note)
  (weight 1)
  (type embellishment)
  (transformation … specification of first transformation …)
    ...
  (transformation … specification of last transformation …))
```

Substitutions divide into two types. The first type, representing a significant change to the flow of a piece, is called a *motif*. An example of a motif is a triplet arpeggio. The second type, being an ornament that just goes on top of a line and doesn't drastically change the color, is called an *embellishment*. An example of an embellishment is a grace note. Because embellishments and motifs aren't used equally or for the same purpose, our implementation first transforms a melody using motif substitutions to make all the significant changes. The result is then further transformed with all the embellishment substitutions to add the ornaments on top of the melody.

Use of motifs *vs*. embellishments is illustrated in Figs. 4-6. We start with the outline melody in Fig. 4. When we apply a set of substitutions to the outline melody, the motif substitutions are applied first, as shown in Fig. 5. That result is then transformed with the embellishment substitutions from the set, giving the fully transformed melody, as shown in Fig. 6. As shown, the more significant alterations are done with the motif substitutions, while the embellishment substitutions just add minor ornaments.

D. Transformation Weight within a Substitution

A transformation's weight is used when the substitution containing it is being applied. As certain implementations of an idiom might be more common than others, when a substitution is being applied to a melody, it will sort its transformations, giving priority to transformations with higher weights. We then try to apply each transformation in sort order, but randomly within a given weight. This will continue until a transformation successfully transforms the melody, at which point the transformed melody is returned. If none of the transformations are able to transform the melody, the original melody is left unchanged at that point.

IV. SYSTEM IMPLEMENTATION

A. Application of Transformational Grammar

Now that we have defined the grammar framework, we can understand how it is applied to a melody line. The grammar specifies a set of substitutions we want applied to a melody. The system is first given a melody, with accompanying chords, to transform. As with the application of transformations described in Section III, the implementation will rank the applicable substitutions in the grammar by weight and try to apply them to the melody. Each substitution and transformation is tried starting with the first note in the given melody. When one of the selected substitutions successfully applies a transformation, the transformed section is added to the transformed melody, which originally starts empty. Then the process is repeated on the remaining notes of the melody line that were not transformed. If no substitution can be applied successfully, the note of the original melody will be added to the transformed melody and the process repeated on the rest of the melody. This is done until there is no more melody to transform, at which point the transformed solo is returned.

As described in Section III.C, the process described above is first done to a melody by the motif substitutions, and then repeated on the result of that by the embellishment substitutions in the grammar. The result is the fully transformed melody.

B. GUI for Transformational Grammars in Impro-Visor

To use our transformational grammars in Impro-Visor [7], we developed a graphical user interface (GUI) that shows the different pieces of information in each substitution, permitting the user to specify different options for applying the grammar in the program. It also allows users to modify any aspect of the grammar without having to alter the grammar file itself. Fig. 7 shows a grammar opened in the GUI with the substitutions from the default grammar file.

Two settings for applying a transformational grammar are *Rectify* and *Enforce Duration Equality*. *Rectify* will smooth the notes of the returned transformed melody that are neither chord tones nor color tones of their underlying chord into the closest note that does fit one of those categories. *Enforce Duration Equality* requires that the duration of the source of a transformation must equal that of the target notes.

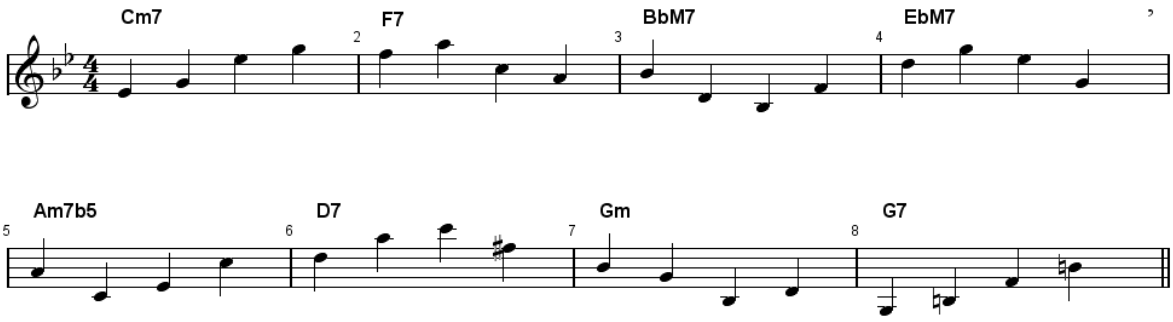


Fig. 4. Outline melody generated by a separate generative grammar

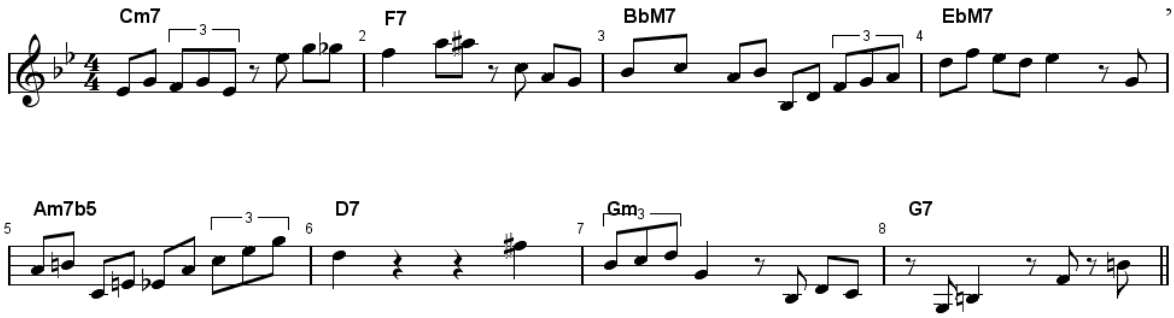


Fig. 5. Outline melody transformed with only the motif substitutions

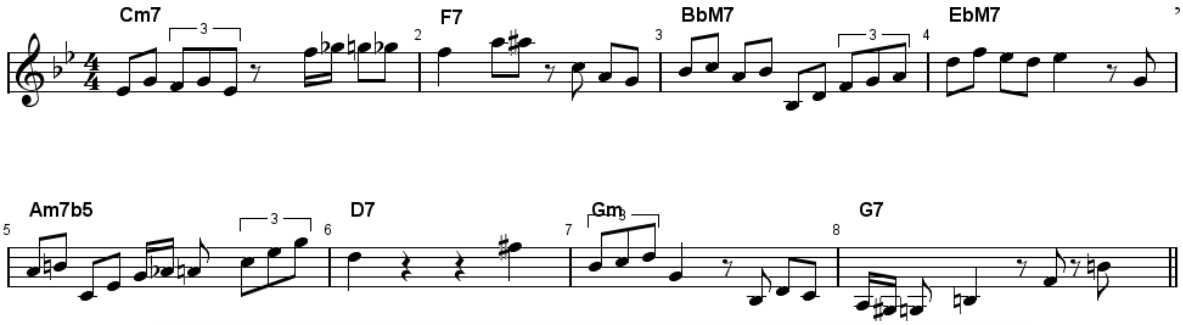


Fig. 6. Transformed melody having applied embellishment substitutions to Fig. 5

Enforce Duration Equality is normally always selected. One use for unselecting this feature could be to see how a transformation is being applied when it is not working, allowing a user to debug an incorrectly written transformation.

To view or edit the transformations that make up a substitution, a user has only to select the substitution, at which point its transformations will fill the *Transformations* list. Fig. 7 shows the different actions that are available when a substitution and transformation are selected. Fig. 7 also shows that the GUI allows the user to change every aspect of the grammar. Source-notes, condition-guard and target-notes of a transformation can be edited by clicking the *Edit Transformation* button, which opens the transformation in a text editor window in which the transformations can be edited, in the form shown in Tables III-IV.

The GUI provides a full visual understanding of the transformational grammar with the ability to edit all elements of it. It also adds the feature of allowing a *subset* of the full grammar to be enabled when applying to a melody. This enabled subset of the grammar is shown with checkboxes for each substitution and transformation. When a grammar is saved, the subset of enabled grammar pieces are saved so that users can easily save whichever substitutions and transformations work best for a piece.

The documentation of all functions the grammar supports is provided in Impro-Visor and can be seen by clicking the *Show Function Documentation* button in the *Transformations Functions* section. Impro-Visor also contains a short tutorial and explanation for the transformational grammar.

V. SAMPLE RESULTS

Using a set of common idioms described in [1] and [14], we created a transformational grammar including substitutions that define the following idioms:

- grace note
- mordent
- triplet arpeggio
- passing tone
- neighbor tones
- triplet ornament

An application of this transformational grammar to the outline in Fig. 4 is shown in Fig. 6. The outline was generated using Impro-Visor with a grammar that generates only chord-tone quarter notes.

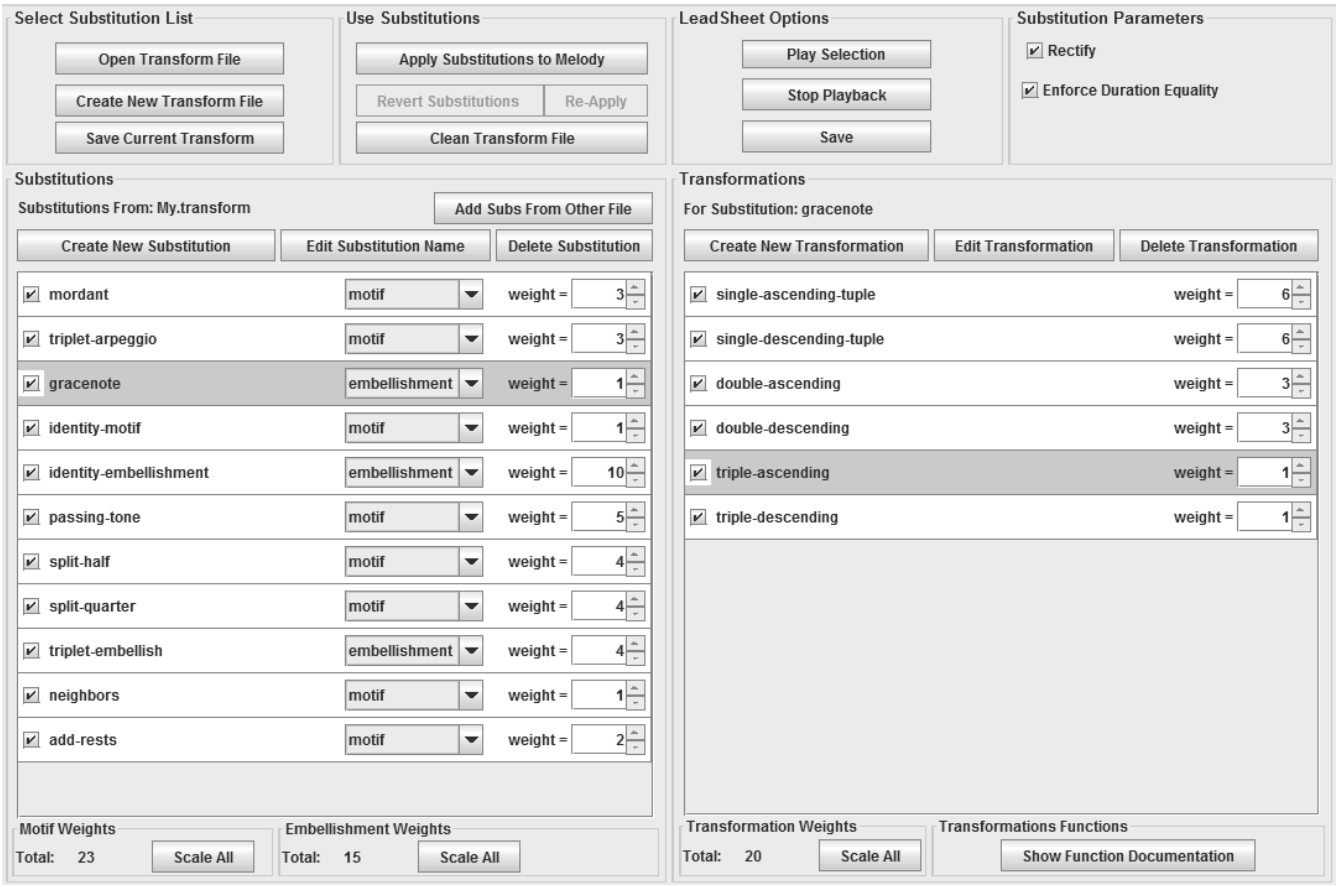


Fig. 7. A substitution and one of its transformations are selected in the transformational GUI

VI. FUTURE WORK

The use of a transformational grammar requires an outline, or simple melody, as a base for transformation. Outlines that will allow for good transformation results are not always easily created, so we seek to *automate* the "flattening" of complex solos into basic outlines. This will require detecting the important notes, or goal notes [1] [3], of a solo and building the outline from them.

As transformations can be time consuming to write and can change based on musician and style of music, we also seek a way to *learn* transformations from transcriptions of solos. This would allow students to extract transformations from solos of interest.

VII. CONCLUSION

We have created a transformational grammar framework to define substitutions that produce idioms from simpler note groups. Although our particular focus is jazz, the grammar is sufficiently general to be applicable across a wide range of music. With a sufficient list of substitutions and simple starting melody, novel melodies are automatically generated from the transformation application process. These melodies can be generated from the same grammar, so if the substitutions are written to represent a certain set of styles, the generated melodies should all fit in that set of styles. The implementation of the grammar in a free software tool provides the ability to show students how to transform a plaintive melody idiom by idiom. The grammar rules have a textual notation readable by users who are trying to understand how to apply the transformations or create new ones.

REFERENCES

[1] S. Berg, *Jazz Improvisation – The Goal Note Method*, Second Edition, Kendor Music, Delevan, NY, 1992.

[2] J. Bergonzi, *Melodic Structures*, Advance Music, Mainz, Germany, 1992.

[3] J. Riposo, *Target and Approach Tones*, Jamey Aebersold, New Albany, IN, 2011.

[4] J. Coker, *Elements of the Jazz Language for the Developing Improvisor*, CPP/Belwin, Inc., Miami, FL, 1991.

[5] T.D. Mason, *The Art of Hearing*, Hal Leonard Corporation, Milwaukee, WI, 1997.

[6] T. Pease, *Jazz Composition: Theory and Practice*, Berklee Press, Boston, MA, 2003.

[7] R. Keller, http://www.cs.hmc.edu/~keller/jazz/improvisor/, last consulted December 2014.

[8] P. Johnson-Laird, "How Jazz Musicians Improvise", *Music Perception*, vol. 19, no. 3, pp. 415-442, 2002.

[9] R.M. Keller and David R. Morrison, "A Grammatical Approach to Automatic Improvisation", Proc. Fourth Sound and Music Computing Conference, Lefkada, Greece, July 2007.

[10] J. Gillick, K. Tang, and R. Keller, "Machine Learning of Jazz Grammars", Computer Music Journal, Fall 2010, vol. 34, no. 3, pp. 56-66.

[11] E. Gómez, G. Peterschmitt, X. Amatriain, and P. Herrera, "Content-Based Melodic Transformations of Audio Material for a Music Processing Application", Proc. of the 6th Int. Conference on Digital Audio Effects (DAFX-03), London, UK, September 8-11, 2003

[12] J.P. Jacobs and J.A. Reggia, "Evolving Musical Counterpoint", http://arxiv.org/pdf/1207.5560.pdf, 2012, last consulted December 2014.

[13] C. Yoste, "A New Model for Algorithmically Improvised Jazz", unpublished report, Willamette University, 2013.

[14] M. Voelpel, *The Best of Charlie Parker*, Hal Leonard Corporation, Milwaukee, WI, 2003.

Alexander M. Putman is from Austin, TX, USA. He is currently a junior pursuing a Bachelor of Science in computer science at Harvey Mudd College in Claremont, California.

His research interests include domain specific languages, music software and artificial intelligence. He worked as a researcher on the Intelligent Music Software project, where he implemented the system described in this paper.

Robert M. Keller received the B.S. in Engineering Science from Washington University in St. Louis and the Ph.D. in Electrical Engineering and Computer Science from the University of California, Berkeley. He held faculty positions at Princeton University and the University of Utah prior to his current position of professor of computer science at Harvey Mudd College, where he directs the Intelligent Music Software project.

In addition to music software, his research interests also include logic, neural networks, parallel computing, machine learning, and jazz education. Teaching a course in jazz improvisation for many years motivated and advanced some of the ideas leading to the present paper.