Machine Improvisation with Formal Specifications

Alexandre Donzé[1], Rafael Valle[1,2], Ilge Akkaya[1], Sophie Libkind[3], Sanjit A. Seshia[1], David Wessel[1,2]

[1]EECS Department, UC Berkeley [2]Music Department and CNMAT, UC Berkeley [3]Swarthmore College

Corresponding author: donze@berkeley.edu

ABSTRACT

We define the problem of machine improvisation of music with formal specifications. In this problem, one seeks to create a random improvisation of a given reference melody that however satisfies a "specification" encoding constraints that the generated melody must satisfy. More specifically, we consider the scenario of generating a monophonic Jazz melody (solo) on a given song harmonization. The music is encoded symbolically, with the improviser generating a sequence of note symbols comprising pairs of pitches (frequencies) and discrete durations. Our approach can be decomposed roughly into two phases: a generalization phase, that learns from a training sequence (e.g., obtained from a human improviser) an automaton generating similar sequences, and a supervision phase that enforces a specification on the generated sequence, imposing constraints on the music in both the pitch and rhythmic domains. The supervision uses a measure adapted from Normalized Compression Distances (NCD) to estimate the divergence between generated melodies and the training melody and employs strategies to bound this divergence. An empirical evaluation is presented on a sample set of Jazz music.

1. INTRODUCTION

Music can be automatically generated either at the audio or at a symbolic level. The former involves processing and synthesizing sound waves, whereas the latter is concerned only with generating *scores*, i.e., sequences of (groups of) symbols, the notes [1], each of them being an abstract representation of a particular sound that can be instantiated in many different variations by different instruments or, generally speaking, by sound synthesizers. Thus, at the symbolic level, generation of music can be reduced into generating sequences of letters, each of which corresponding to a note.

Music improvisation is a special case of music generation where one generates a random variant of a given melody. The field of computer music improvisation, also termed as *machine improvisation*, has been well studied [1]. One approach to improvisation is *data-driven*, wherein recurrent patterns are inferred from the reference melody, and then replicated and recombined to form the improvisation. Different data structures and algorithms have been proposed for this purpose such as incremental parsing (IP) [2] inspired from dictionary based compression algorithms from the Lempel-Ziv family [3], probabilistic suffix trees (PST) [4], and factor oracles (FO) [5]. Another approach is *rule-based*, where an expert encodes rules in a formal system such as a stochastic context free grammar (CFG), using these to control which sequences are generated [6]. These two approaches, while very effective in many situations, lack certain desirable properties. First, certain rules need to be enforced always, not just probabilistically, and can often be captured using automata-theoretic formalisms. Second, while formalisms such as stochastic CFGs capture some rules, it can be desirable to separate out the generation mechanism from the language in which rules are expressed. Third, it can also be desirable to control the amount of "creativity" in the improvisation, using some kind of divergence measure. [2]

In this paper, we present a new approach to machine improvisation of music that incorporates the notion of a *formal specification*. A formal specification is a mathematical statement of what a system must or must not do, often expressed in mathematical logic or using automata-theoretic formalisms. It is central to certain fields of computer science and electrical engineering, such as program verification or supervisory control. The latter problem bears some resemblance to the problem of music improvisation, so we elaborate the connection here. *Supervisory control* refers to the problem of designing a controller (aka "supervisor") that will guarantee that a given system (aka "plant") always satisfies a set of formal specifications. If we think of formal specifications as music rules and the "plant" as a random improviser, then the supervisory controller is similar to a controlled improviser of music. We formalize this connection in the present paper and apply it to the machine improvisation of Jazz music.

Specifically, we consider the scenario of generating a monophonic (solo) melody over a given Jazz song harmonization, similar to a given reference (or "training") melody. The improvised sequence has to be synchronized with another sequence, usually the chord progressions, considered as fixed and called hereafter the accompaniment. The improviser then has to be a function of the training sequence, the accompaniment, and other imposed constraints such

[1] Inharmonic and aperiodic sounds are beyond the scope of this paper.

[2] We present a more detailed discussion of related work in Section 6.

as "safety" rules and divergence measure. We present a generic strategy for solving this problem in three phases. The first phase, *generalization*, learns from the given melody an automaton generating a set of melodies containing the original. We implement this phase using factor oracles [5]. The second phase, *safety supervision*, enforces rules on the generalized automaton so that it plays in harmony with the accompaniment. The rules are analogous to "safety properties" that a control system must always obey. The third and final phase, *divergence supervision*, ensures that sequences produced by the improviser automaton lie, with high probability, within a specified "similarity" divergence from the original. This phase is implemented by assigning probabilities to transitions of the improviser automaton based on a given divergence measure. For music, several divergence measures have been proposed often in the purpose of genre classifications; amongst these, Normalized Compression Distances (NCDs) have been effectively used [7], and so we employ a variant of an NCD in this paper.

In summary, the main novel contributions of this paper are:

- The formal notion of *controlled machine improvisation* for formal specifications (Section 2);
- An approach to solve the controlled machine improvisation problem based on generalization, safety supervision, and divergence supervision (Section 3), and
- An instantiation and application of our approach to improvisation of Jazz melodies (Sections 4 and 5).

A preliminary version of some of the ideas in this paper have appeared in a technical report [8]. That report, not formally published, is written from the viewpoint of *control theory* and introduces a broader notion termed *control improvisation*. In this paper, we customize the ideas for machine improvisation of music. We further extend our implementation and experimental evaluation for Jazz improvisation.

2. CONTROLLED MACHINE IMPROVISATION

2.1 Notation and Background

As our goal is to generate symbolic musical improvisations, we work with traditional score notations, which are based on discrete sets, namely, a discrete set of pitches (e.g., `a4`, `c2`, `g3`, etc), and a discrete set of durations (quarter notes ♩, eighth notes ♪,etc). As a consequence, the formal background can be set up in terms of finite state automata.

Definition 1. *A finite state automaton (FSA) is a tuple* $\mathcal{A} = (Q, q_0, F, \Sigma, \rightarrow)$ *where* Q *is a set of* states*,* $q_0 \in Q$ *is the* initial state*,* $F \subset Q$ *is the set of* accepting states*,* Σ *is a finite set called the* alphabet *and* $\rightarrow \subset Q \times \Sigma \cup \{\epsilon\} \times Q$ *is a transition relation. for which we use the usual infix notation* $q \xrightarrow{\sigma} q'$ *to mean that* $(q, \sigma, q') \in \rightarrow$*, and* ϵ *is the* empty word*.*

We interpret letters of the alphabet as observable events of the system under consideration. A *word* $w \in \Sigma^*$ is either ϵ (transition with no observable event) or a finite sequence of letters in Σ, i.e. $w = \sigma_1\sigma_2 \ldots \sigma_k$ for some integer $k \geq 1$. The length of a word is defined inductively as $|\epsilon| = 0$ and $|w\sigma| = |w| + 1\ \forall \sigma \in \Sigma$. A word is a *trace* of a FSA $\mathcal{A}$ iff there exists a sequence of states $q_i \in Q$ such that $q_0 \xrightarrow{\sigma_1} q_1 \xrightarrow{\sigma_2} \ldots \xrightarrow{\sigma_{n-1}} q_{n-1} \xrightarrow{\sigma_n} q_n$. It is an *accepting trace* of $\mathcal{A}$ iff q_n is in F. The *language of* $\mathcal{A}$, noted $\mathcal{L}(\mathcal{A})$ is the set of accepting traces of $\mathcal{A}$.

Definition 2. *(Synchronous Product) The synchronous product of* $\mathcal{A} = (Q, q_0, F, \Sigma, \rightharpoonup)$ *and* $\mathcal{A}' = (Q', q'_0, F', \Sigma, \rightharpoonup)$*, noted* $\mathcal{A}||\mathcal{A}'$*, is defined as the FSA* $\mathcal{A}||\mathcal{A}' \triangleq (Q \times Q', (q_0, q'_0), F \times F', \Sigma, \rightarrow)$ *where* $\forall \sigma \in \Sigma \cup \{\epsilon\}$*,* $(q_i, q'_i) \xrightarrow{\sigma} (q_j, q'_j)$ *if and only* $q_i \xrightarrow{\sigma} q_j$ *and* $q'_i \xrightarrow{\sigma} q'_j$*.*

Intuitively, $\mathcal{A}||\mathcal{A}'$ is a finite state machine where $\mathcal{A}$ and $\mathcal{A}'$ take transitions synchronously, with the constraint that for a transition to be possible, both current states of $\mathcal{A}$ and $\mathcal{A}'$ must have an outgoing transition driven by the same event.

2.2 Problem Definition

FSAs equipped with the synchronous product are then sufficient to define a "controller synthesis" problem. Assume that $\mathcal{A}$ models the behavior of a system for which some states are labeled as "bad". Synthesizing a controller amounts to finding $\mathcal{A}^c$ such that the product of $\mathcal{A}$ and $\mathcal{A}^c$ will naturally *disable* transitions leading to bad states. Note that for modeling convenience, $\mathcal{A}$ is often itself decomposed into a *plant* $\mathcal{A}^p$ and a specification $\mathcal{A}^s$ so that $\mathcal{A} = \mathcal{A}^p||\mathcal{A}^s$. A bad state is typically one from which no accepting state of $\mathcal{A}^s$ is reachable.

A controller is said to be *non-blocking* if it always allows the system $\mathcal{A}||\mathcal{A}^c$ to reach an accepting state. It is said to be *maximally permissive* when it does not disable more transitions than strictly necessary. There is a simple algorithm [9] for finding a non-blocking, maximally-permissive, memoryless controller, when one exists. Informally, the algorithm is based on locating "bad" states in the composite automaton and then iteratively pruning away transitions to such states, while marking as "bad" states any predecessors of existing "bad" states or new blocking states. The framework of supervisory control, while relevant, is not sufficient for our setting of improvisation. There are two main differences:

(i) *Randomness:* To improvise is to incorporate some randomness ("unpredictability"), whereas traditional control seeks to find safe, deterministic strategies, and

(ii) *Bounded Divergence:* The improvisation is created from a *reference trace* w_{ref}, and is typically "similar" to it. The problem definition should capture this constraint.

We therefore defined a new controller synthesis problem, termed as the *control improvisation* problem. The goal is to randomly generate traces among a family of "safe" traces which are equivalent based on some *divergence measure*. We assume that the latter is given by a non-negative function $d_{w_{\text{ref}}}$ on words, such that $d_{w_{\text{ref}}}(w_{\text{ref}}) = 0$ and $d_{w_{\text{ref}}}(w)$ increases as w gets "further" from w_{ref}. A controller solv-

ing the control improvisation problem influences the behaviors of $\mathcal{A}^p$ in two ways:

1. When several transitions of $\mathcal{A}^p$ are safe with respect to $\mathcal{A}^s$, one is picked following a random distribution in accordance with the similarity criterion;
2. When no safe transition is available, one transition of $\mathcal{A}^p$ is modified (replaced with alternative transitions to the same end state but labeled with a different event) to prevent blocking while still preserving safety.

Formally, the control improvisation problem is defined as follows:

Definition 3. *(Control Improvisation Problem) A control improvisation problem $\mathcal{P}_I$ is defined by: a plant FSA $\mathcal{A}^p$, a specification FSA $\mathcal{A}^s$, an accepting trace w_{ref} of $\mathcal{A}^p||\mathcal{A}^s$, a divergence measure $d_{w_{ref}}$, an interval $I = [\underline{d}, \overline{d}]$ and a pair of reals $(\varepsilon, \rho) \in (0, 1)$. A solution of $\mathcal{P}_I$ is a probabilistic automaton generating words w in Σ^* such that the following conditions hold for each w:*

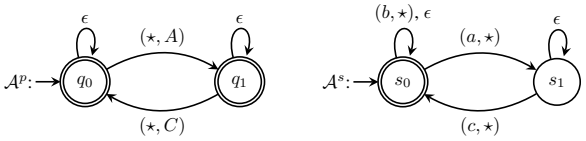
(a) Safety: *w is an accepting trace of $\mathcal{A}^s||\mathcal{A}^p$;*
(b) Randomness: *The probability measure of w* [3] *is smaller than ρ,*
(c) Bounded Divergence: $Pr(d_{w_{ref}}(w) \in [\underline{d}, \overline{d}]) > 1 - \varepsilon$.

Note that problem easily generalizes to the case where several reference words w_r are provided.

2.3 Running Example

We present below a running example to illustrate the definitions and approach:

- An alphabet composed of two sets of symbols $\Sigma = \Sigma_a \times \Sigma_A$, where $\Sigma_a = \{a, b, c\}$ and $\Sigma_A = \{A, C\}$
- A plant model $\mathcal{A}^p$ and a specification automaton $\mathcal{A}^s$:



  where we use the special symbol $\star$ as a "don't care" symbol. E.g., $(b, \star)$ represents either (b, A) or (b, C);
- A set of reference words given by

$$w_r = (b, \star)(b, \star)(a, \star)(c, \star)$$

  . Note that when projected on alphabet Σ_a, w_r maps to the unique word $bbac$. For ease of reading, $\star$ might be omitted in the following.

Anticipating the musical encoding described in more details in Section 4, one can roughly see Σ_a as a set of "notes", and Σ_A as a set of "chords". The plant model fixes the succession of "chords" as alternances of A followed by C, and the specification model puts constraints on "notes", such that an a has always to be followed by a c. The goal is thus to improvise sequences of "notes", variations of w_r so that the resulting word is compatible with $\mathcal{A}^p$ and $\mathcal{A}^s$, e.g., $(b, A)(a, C)(c, A)(b, C)$.

[3] Intuitively, the probability that w is picked among all admissible w.

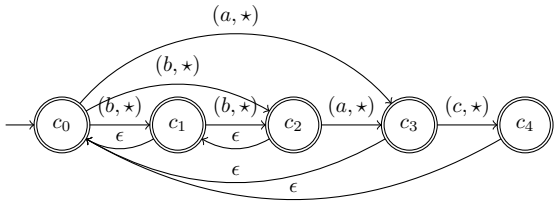


Figure 1. Factor Oracle improviser obtained from the reference words $(b, \star)(b, \star)(a, \star)(c, \star)$.

3. FACTOR ORACLE APPROACH

3.1 Generalization using Factor Oracles

The core of our improvisation approach is based on the factor oracle (FO) structure [5]. A factor oracle is a compact automaton representation of all contiguous subwords (factors) contained in a word $w = \sigma_1\sigma_2 \dots \sigma_n$. It has $|w| + 1$ states, all accepting, and its transitions can be categorized into

1. *Direct* transitions of the form $s_i \xrightarrow{\sigma_{i+1}} s_{i+1}$;
2. *Forward* transitions of the form $s_i \xrightarrow{\sigma} s_j$ where $j > i+1$ and σ is some letter in w;
3. *Backward* transitions, also called *suffix links*, of the form $s_i \xrightarrow{\epsilon} s_j$ with $j < i$.

The details of the construction of factor oracles, can be found in [10]. Some properties of FOs are as follows:

- An accepting word that takes only direct transitions is a prefix of w;
- Factors of w are accepting words taking only direct and forward transitions;
- Finite concatenation of factors of w are accepting words taking all three types of transitions.

These properties make the FO a suitable structure to generalize w_{ref}, so a first step to solve the control improvisation problem is to define $\mathcal{A}^g = \text{FO}(w_{\text{ref}})$. In Figure 1, we show the factor oracle obtained from the reference word w_r.

3.2 Enforcing Specifications

Even though w_{ref} is an accepting word for $\mathcal{A}^p||\mathcal{A}^s$, there is no guarantee that its generalization $\mathcal{A}^g$ composed with $\mathcal{A}^p$ is non-blocking for $\mathcal{A}^s$. However, assuming that there exists a non-blocking memoryless controller $\mathcal{A}^c_{\text{max}}$ for $\mathcal{A}^p||\mathcal{A}^s$ — something that can be checked using standard supervisory control [9] and which is guaranteed by the existence of w_{ref} — it is always possible to make $\mathcal{A}^p||\mathcal{A}^g||\mathcal{A}^s$ non-blocking by adding transitions as follows. Let (q, c, s) be a blocking state of $\mathcal{A}^p||\mathcal{A}^g||\mathcal{A}^s$. Since $\mathcal{A}^c_{\text{max}}$ is a non-blocking memoryless controller, there exists a non-blocking transition $(q, s) \xrightarrow{\sigma} (q', s')$ in $\mathcal{A}^p||\mathcal{A}^s$ for some $\sigma \in \Sigma$. Hence we can pick some state c' in $\mathcal{A}^g$ and add the transition $c \xrightarrow{\sigma} c'$ to the transition relation of $\mathcal{A}^g$. This effectively adds the transition $(q, c, s) \xrightarrow{\sigma} (q', c', s')$ in $\mathcal{A}^p||\mathcal{A}^g||\mathcal{A}^s$. This procedure is repeated until no blocking state can be found in $\mathcal{A}^p||\mathcal{A}^g||\mathcal{A}^s$.

To illustrate this construction, consider automaton $\mathcal{A}^p$,

$\mathcal{A}^s$ defined in Sec. 2.3 and $\mathcal{A}^g$ on Fig. 1. Constructing the product $\mathcal{A}^p||\mathcal{A}^s||\mathcal{A}^g$, we find that the run

$$(q_0, s_0, c_0) \xrightarrow{(b,A)} (q_1, s_0, c_1) \xrightarrow{(b,C)} (q_0, s_0, c_2) \rightarrow$$

$$\xrightarrow{(a,A)} (q_1, s_1, c_3) \xrightarrow{\epsilon} (q_1, s_1, c_0)$$

leads to a blocking state (q_0, s_1, c_0), as $\mathcal{A}^s$ requires a c transition, whereas $\mathcal{A}^g$ only permits an a or a b. Hence we add the transition $c_0 \xrightarrow{(c,C)} c_1$.

One remaining question is how to pick the un-blocking transition when more than one choice is possible. One possibility is to pick a transition which is close in some sense to an existing transition of $\mathcal{A}^g$. For example in our musical application where transitions corresponds to note events, we can pick notes with the closest pitch or duration. Another possibility is to rely on a Markov model as suggested in the next section, to pick the most frequent valid successor note.

3.3 Divergence Control

The last step is to define transition probabilities satisfying the Randomness and Bounded Divergence requirements. We begin by concretizing the similarity divergence that we use, which is a variant of the Normalized Compression Distance introduced in [11] and is based on the theory of Kolmogorov complexity. The Kolmogorov Complexity of an object x (denoted $K(x)$) is defined as the length of the shortest compressed code to which x can be losslessly reduced. The Kolmogorov Complexity of y given x (denoted $K(y|x)$) is the length of the shortest compressed code to which y can be losslessly reduced assuming knowledge of x. In practice, $K(x)$ is not computable, and so is typically approximated by $C(x)$ where $C(x) = \text{length}(compress(x))$ for some compression algorithm $compress$ and $K(y|x)$ can be approximated by $C(y|x) = C(xy) - C(x)$[7], where xy is the concatenation of x and y. Then the Normalized Compression Distance (NCD) [11] between x and y is defined as

$$NCD(x, y) = \frac{\max(C(x|y), C(y|x))}{\max(C(x), C(y))}.$$

Informally, it estimates 1 minus the mutual information in x and y. In our case however, the amount of information in an improvisation that is not in the reference trace is of more interest than mutual information, hence we define the similarity divergence based on the asymmetric quantity $\frac{C(y|x)}{C(y)}$, as follows.

Definition 4 (Similarity Divergence $d_{w_{\text{ref}}}$)**.**

$$d_{w_{ref}}(w) = \frac{C(w|w_{ref})}{C(w)} + (1 - \frac{C(w_{ref}w_{ref})}{C(w_{ref})})$$

The second term in the sum ensures that $d_{w_{\text{ref}}}(w_{\text{ref}}) = 0$. In our application we used the LZW compression algorithm to compute $C(\cdot)$.

Finally, a simple way to assign probabilities to transitions in a FO is as follows. Recall that traversing the $n+1$ states in sequence by taking direct transitions reproduces w_{ref}. Improvisation, i.e., variation from the original sequence, is obtained by randomly taking forward transitions or backward transitions. Thus the higher the probability of taking direct transitions, the more similar the output is to the original sequence. In our implementation, we assign the probability p to each direct transition, so that the improviser replicates w_{ref} when $p = 1$, and probability $1 - p$ equidistributed to other outgoing forward or backward transitions. This provides for a simple parameter controlling how different the improvised sequence is from w_{ref}.

A more sophisticated way of assigning the probabilities is to build a first-order Markov Chain (MC) on the pitch sequence of w_{ref} and assign the probabilities of forward and backward transitions according to the transition probabilities of the MC, which is computed using frequentist inference. This substitutes the previously described uniform distribution of forward and backward transitions with a non-uniform distribution, with the purpose of generating sequences whose pitch distribution is more similar to the training sequence.

The overall process for generating an improvised sequence of n events is summarized below:

1. Maintain a sequence $(q_0, c_0, s_0)(q_1, c_1, s_0) \dots (q_k, c_k, s_k)$ of states of $(\mathcal{A}^p||\mathcal{A}^g)||\mathcal{A}^s$ and a word $w_k = \sigma_0\sigma_1 \dots \sigma_k \in \Sigma^k$,
2. If $k \geq n$ and s_k is accepting, return $w = w_k$
3. Else if (q_k, c_k, s_k) has outgoing transitions (non-blocking), assign probabilities according to replication probability p and pick σ_{k+1} and $(q_{k+1}, c_{k+1}, s_{k+1})$
4. Else if (q_k, c_k, s_k) is blocking, pick a safe σ_{k+1} and $(q_{k+1}, c_{k+1}, s_{k+1})$ as defined in Section 3.2.

For a given choice of replication probability p and $\varepsilon > 0$, there is an interval $I = [\underline{d}, \overline{d}]$ such that the probability that the divergence between the improvised and reference word is in I is less than ε. This interval can be determined experimentally, and our experiences showed that I is usually narrow, which indicates that p is a suitable parameter to adjust to a desired divergence [8].

4. A FORMAL MODEL

4.1 Musical Notations

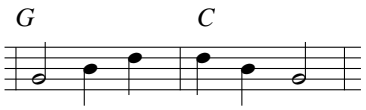
We abstract and formalize a piece of jazz music into a *melody*, a string of pitched notes and rests, aligned with an *accompaniment*, a looping sequence of chords with given durations. The time unit is the *beat* and the piece is divided into bars which are sequences of k beats. We assume that the accompaniment is fixed and our goal is to define an improviser for the melody. Hence, the plant FSA will model the behavior of the accompaniment, without constraining the melody, and the specification FSA will set constraints on acceptable melodies played together with the accompaniment. To encode all events in a score, we use an alphabet composed of the cross-product of four alphabets: $\Sigma = \Sigma_p \times \Sigma_d \times \Sigma_c \times \Sigma_b$, where

- Σ_p is the *pitches* alphabet, e.g., $\Sigma_p = \{\text{𝄽}, \texttt{a0}, \texttt{a\#0}, \texttt{b0}, \texttt{c0}, \cdots\}$,

- Σ_d is the *durations* alphabet, e.g., $\Sigma_d = \{\text{𝅘𝅥𝅯}, \text{♩}, \text{𝅗𝅥}, \ldots\}$ with ♩ = 1 beat. Note that Σ_d also includes fractional durations, e.g., for triplets, as discussed below;
- Σ_c is the *chords* alphabet, e.g., $\Sigma_c = \{$C, C7, G, Emaj, Adim, $\ldots\}$,
- Σ_b is the *beat* alphabet. E.g, if the smallest duration (excluding fractional durations) is the eighth note, i.e., half a beat, then $\Sigma_b = \{0, 0.5, 1, 1.5, 2, 2.5, 3, 3.5\}$.

All automata in the following will use implicitly the full alphabet Σ. However, each component alphabet is meant to address one particular aspect of the music formalization, and we will construct the specification automaton by the composition of different sub-automata using these different component alphabets. Also, note that this encoding of music is of course not unique nor meant to be canonical, and other types of alphabets can be used in replacement to or to complement the one we propose and used. E.g., we do not consider here note *velocity* (i.e., the intensity of the sound of the note).

Example 1. *we provide an example encoding of a simple score using the formalism defined in Section 4. Consider the following extract:*



It contains a melody and a chord progression which is represented by the following word in our alphabet:

(g4,𝅗𝅥,G,0) (b4,♩,G,2) (d5,♩,G,3)
(d5,♩,C,0) (b5,♩,C,1) (g4,𝅗𝅥,C,2)

4.2 Encoding Chord Progressions

The harmonic context of the melody is given by the chord progression (accompaniment). The plant FSA $\mathcal{A}^p$ then encodes the events of specified chords at specified times. The basic idea of the encoding is to define as many states as there can be events of the minimal possible duration in a bar, i.e., in four beats, and replicate those states for as many bars as needed. Then transitions from one state q to another state q' is possible when a note of the proper duration is possible and if in the duration of this note, there is no chord change. This construction is illustrated in Figure 2.

4.3 Rhythmic and Harmonic Specifications

The specification FSA encodes rhythmics and harmonic (tonal and modal) constraints involving notes in the melody which enforce some general structure and basic musical consistency. The following specifications are adapted and simplified from the generic guidelines found in [12]. We structure Jazz melodies into *licks* defined informally as short melodic phrases of pitched notes separated by either rests or long notes. Then we impose that licks start on specific beats. E.g., start beats can be 0.5, 1.5, 2.5 or 3.5, i.e., *off-beats*. This specification can be encoded in the automaton (a) on Fig. 3.

The second specification has to do with durations which are not multiples of the smallest duration. In that case, we require that it be repeated until the total duration is such a multiple. The typical example of this situation is the triplet, e.g., 

, which is the concatenation of three notes of duration $\frac{1}{3}$, noted $\overset{3}{\text{𝅘𝅥𝅯}}$. Without loss of generality, we model only this case, shown as the FSA (b) in Fig. 3, as other fractional durations are dealt with in a similar manner.

Finally, we define constraints on the pitches of the notes in the melody. The pitched notes are classified based on their accompanying chord. We follow the three primary tone classifications as described in [12]:

- **Chord tone:** a pitch belonging to the current chord;
- **Color tone:** a pitch that does not belong to the current chord but complements and creates euphony with the current chord;
- **Approach tone:** neither a chord nor color tone that is followed by pitched note that differs by exactly 1 semitone;

This classification provides a set of "good" pitches for each chord. Color tones can be defined by a scale, i.e., a set of pitches, which is overall "compatible" with the whole song, and to which we remove potential "avoid" notes for the current chord. As an example, consider a song in the key of C. All notes in the C major scale $\{\texttt{c}, \texttt{d}, \texttt{e}, \texttt{f}, \texttt{g}, \texttt{a}, \texttt{b}\}$ are safe to be played in general, however if an F chord is played (composed of pitches $\texttt{f}, \texttt{a}, \texttt{c}$), we need to avoid $\texttt{b}$ which is highly dissonant with $\texttt{f}$. Hence the set of good notes in this situation is $\{\texttt{c}, \texttt{d}, \texttt{e}, \texttt{f}, \texttt{g}, \texttt{a}\}$. The assignment of individual scales and modes, e.g. pentatonic, octatonic, ionian, lydian, to chords in the chord progression is also possible, thus increasing the temporal granularity of harmonic constraints, which is specially relevant for tonicization, modulation and modal harmony. The approach tones make it possible to deviate "temporarily" from these good notes: if a note not classified as good is played, it must be short and followed by a good note immediately and not further than a semi-tone away from it. We simplify this into the automaton (c) in Fig. 3.

4.4 Improviser Architecture

The automaton obtained by composing the specifications above with the accompaniment automaton is non-blocking; thus, we can apply the approach proposed in Section 3. However, our early experiments showed that a single viewpoint system in which the model predicted note duration and pitches together was too inflexible, in that the control mechanism would have to add too many edges to the factor oracle generator in order to avoid blocking states. Therefore, we adopted a multiple viewpoint system which improvises rhythms and melodic pitches separately. The architecture presented in Figure 4 has been implemented in Python, using the Music21 library. [4] We present some re-

[4] `http://web.mit.edu/music21`

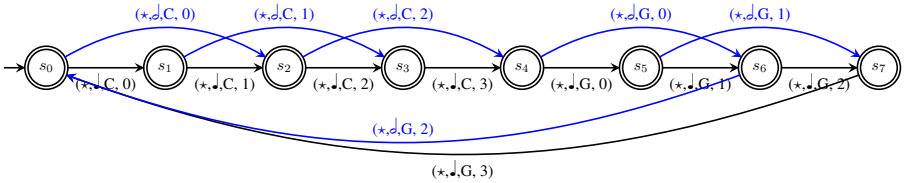
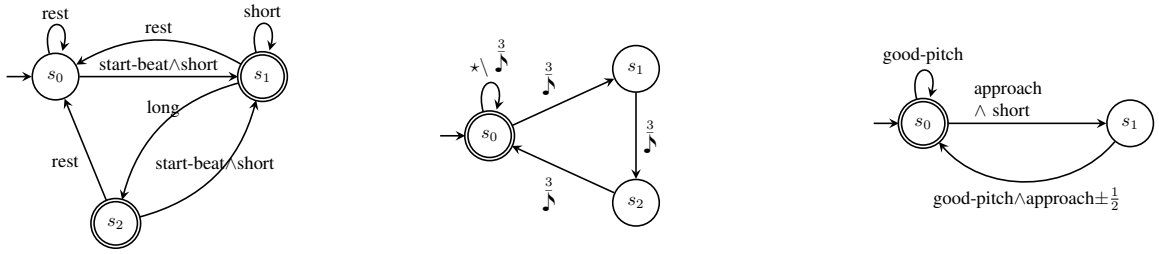


Figure 2. Chords progression automaton $\mathcal{A}^p$ of the example. It consists in an accompaniment looping on chord C during 4 beats (1 bar) and chord G during 1 bar, with duration alphabet restricted to quarter notes and half notes.



(a) $\mathcal{A}^s_1$:Licks (b) $\mathcal{A}^s_2$: Triplets (c)$\mathcal{A}^s_3$: Pitches

Figure 3. Specification automata $\mathcal{A}^s = \mathcal{A}^s_1||\mathcal{A}^s_2||\mathcal{A}^s_3$. "rest" indicates a rest in the melody of any duration. "start-beat" indicates a label of the form $(\star, \star, \star, b)$ where b is a beat value for which a lick can start. "short" indicates a note in the melody of short duration, e.g., of duration less or equal to a beat (♩). Conversely, "long" indicates a note of a longer duration, e.g., strictly more than ♩. "good-pitch" indicates a note with a pitch which is either a chord or a color tone. "approach" indicates an approach tone. "approach$\pm\frac{1}{2}$" indicates an approach tone plus or minus a semi-tone.

sults in the next section, as well as on a dedicated webpage [5].

5. RESULTS

We evaluated our improviser using a melody generated by Impro-visor ([6]) over the standard 8-bar blues chord progression, and the first verse melody of *The Girl from Ipanema* composed by Tom Jobim. Using the Impro-visor generated melody as the reference trace, we generated an improvisation with and without specification both with probability of direct transitions assigned to be $p = 0.8$ (Figure 6). The reference melody and the controlled improvisation share several similarities, however the improvisation also deviates sufficiently from the original to be considered unique. The uncontrolled improvisation contains several notes (highlighted in red) that are not chord, color, or approach tones and which are therefore undesirable.

Using the verse melody of *The Girl from Ipanema* as the reference trace, we generated over the B section of the same piece one improvisation from a supervised factor oracle with specifications based on one scale per chord and another improvisation with specifications based on one key for all chords. Due to the modulatory nature of the chord progression and different tonicization every four bars, the key-based method uses color tones that do not fit the chords or the local harmonic field implied by them. On the contrary, the scale based method produces notes that better fit

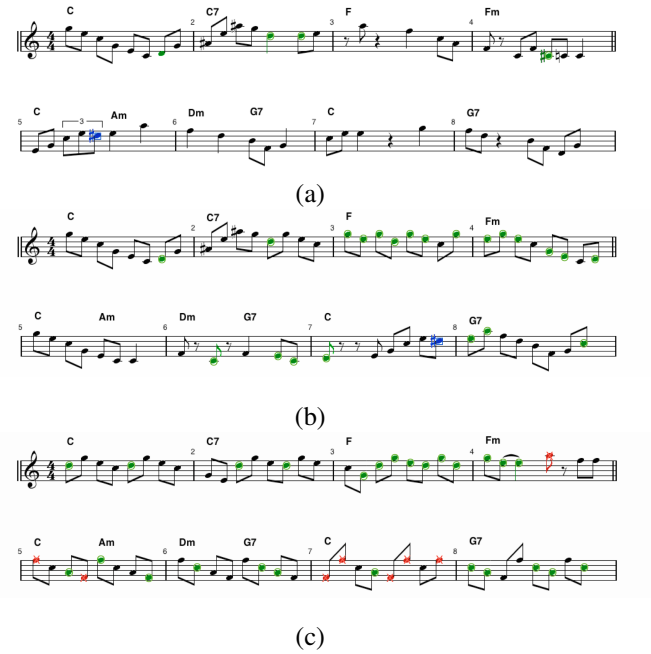


Figure 5. (a) Training melody, (b) improvisation generated with specifications (c) improvisation generated without specifications. Black notes are chord tones, green (circled) notes are color tones, blue (squared) notes are approach tones, and red (crossed) notes are other (undesirable) tones.

[5] http://www.eecs.berkeley.edu/~donze/impro_page.html

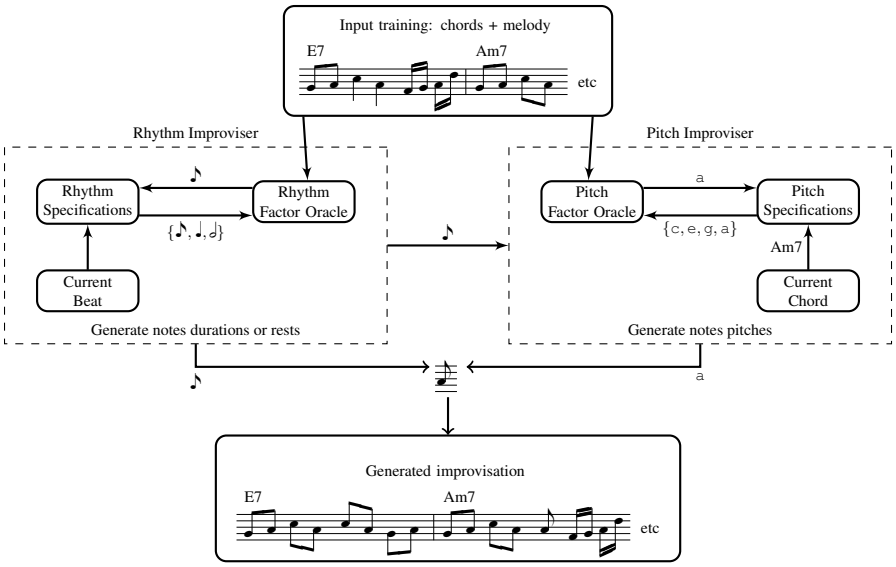


Figure 4. Architecture of the improviser with multiple view points.

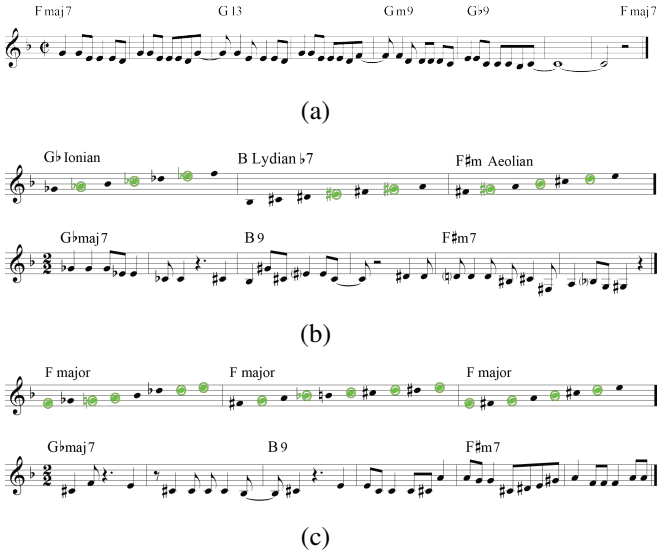


Figure 6. (a) Girl From Ipanema training melody, (b) improvisation generated with one mode specification per chord, (c) improvisation generated with global key specification. Black notes are chord tones, green (circled) notes are color tones.

the temporal granularity of the chord progression and gives the user control over the color tone specifications.

6. RELATED WORK

Broadly speaking, there are two approaches to automatic music improvisation: *rule-based* and *data-driven*. Rule-based approaches attempt to define the rules of "good" improvisations and generate pieces of music that follow those rules. However, it has been observed that it is difficult to come up with the "right" rules, resulting in systems that are either too restrictive, limiting creativity, or too relaxed, thereby allowing musical dissonance [13, 14, 6]. Consequently, recent musical improvisers tend towards *data-driven* or "predictive" approaches that employ machine learning. These approaches learn a probabilistic model from music samples, and use that model to generate new melodies. Examples of such models include stochastic context-free grammars (SCFGs) [15, 12], hidden Markov models (HMMs) [16], and universal predictors [4, 14, 5, 17]. Some approaches combine rule-based and data-driven approaches; e.g., the Impro-visor system [6] based on SCFGs has rules learned from training licks through the grammatical inference [15].

It has been found that certain universal predictors outperfom other stochastic models in producing stylistically appropriate music [14]. Universal predictors vary based on the data structures and algorithms used, such as incremental parsing (IP) [2] inspired from dictionary based compression algorithms from the Lempel-Ziv family [3], probabilistic suffix trees (PST) [4], and factor oracles (FO) [5]. Amongst these, it has been found that the latter has some advantages. Unlike both IP and PST, the factor oracle is both complete (contains all factors of the given word) and can be constructed on the fly. Due to this, factor oracles are at the core of the OMax improvisation system [6] de-

[6] http://repmus.ircam.fr/omax/home

veloped at IRCAM and which has been used in a number of performances. The work closest to ours is presented in [18] which describes ImproTek, a variant of OMax which implements a notion of harmonic context influencing the FO-based improvisation.

Our approach extends this state of the art by providing a way to (i) enforce certain rules in a flexible and modular way on the generated melody, and (ii) bounding the similarity divergence from the original melody. Also, many of the improvisers discussed rely on a single viewpoint system. In other words they attempt to encapsulate and improvise all aspects (rhythm, pitches, volume, etc.) of an improvisation simultaneously. For example, in [14] the alphabet of the prediction model is the cross product of the beat each note starts on, the note's pitch, and the note's duration. Following Conklin and Cleary [19] we implemented a more flexible multiple viewpoint approach to music generation in which note aspects are predicted separately and then aggregated.

7. CONCLUSION

We introduced the concept of control improvisation and presented an approach to solve it. Our approach shows promise for automatic improvisation of Jazz music but we believe this paper is just a first step, and there is plenty of room for futher work. More work is required to investigate the full space of possible similarity divergence measures for different styles of music and there is room for improvement over the base approach we present in this paper, e.g., to provide stronger theoretical guarantees for the "bounded distance" condition. Also one can consider inferring the specification automaton from examples of "good" and "bad" melodies. Further, it would be interesting to consider real-time improvisation (a preliminary implementation was done in Ptolemy [7] [20]) and improvising collectively on a set of melodies rather than just a solo piece.

Acknowledgments
This research was supported in part by the TerraSwarm Research Center, one of six centers supported by the STARnet phase of the Focus Center Research Program (FCRP) a Semiconductor Research Corporation program sponsored by MARCO and DARPA.

8. REFERENCES

[1] R. Rowe, *Machine Musicianship*. MIT Press, 2001.

[2] S. Dubnov, G. Assayag, and R. El-Yaniv, "Universal classification applied to musical sequences," in *Proceedings of the International Computer Music Conference*, 1998, pp. 332–340.

[3] G. Assayag, S. Dubnov, and O. Delerue, "Guessing the composer's mind: Applying universal prediction to musical style," in *Proceedings of the International Computer Music Conference*, 1999, pp. 496–499.

[4] S. Dubnov, G. Assayag, and O. L. G. Bejerano, "A system for computer music generation by learning and improvisation in a particular style," *IEEE Computer*, vol. 10, no. 38, 2003. [Online]. Available: http://articles.ircam.fr/textes/Dubnov03a/

[5] G. Assayag and S. Dubnov, "Using factor oracles for machine improvisation," *Soft Comput.*, vol. 8, no. 9, pp. 604–610, 2004.

[6] R. M. Keller and D. R. Morrison, "A grammatical approach to automatic improvisation," in *Proceedings SMC'07, 4th Sound and Music Computing Conference*, Lefkada, Greece, 2007, pp. 330 – 337.

[7] R. Cilibrasi, P. Vitanyi, and R. De Wolf, "Algorithmic clustering of music," in *Web Delivering of Music, 2004. WEDELMUSIC 2004. Proceedings of the Fourth International Conference on*. IEEE, 2004, pp. 110–117.

[8] A. Donze, S. Libkind, S. A. Seshia, and D. Wessel, "Control improvisation with application to music," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2013-183, Nov 2013. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2013/EECS-2013-183.html

[9] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[10] L. Cleophas, G. Zwaan, and B. W. Watson, "Constructing factor oracles," in *In Proceedings of the 3rd Prague Stringology Conference*, 2003.

[11] M. Li, X. Chen, X. Li, B. Ma, and P. M. Vitányi, "The similarity metric," *Information Theory, IEEE Transactions on*, vol. 50, no. 12, pp. 3250–3264, 2004.

[12] R. M. Keller, *How to Improvise Jazz Melodies*, 2012. [Online]. Available: http://www.cs.hmc.edu/~keller/jazz/improvisor/HowToImproviseJazz.pdf

[13] D. Cope, *Computers and Musical Styles*. Oxford University Press, 1991.

[14] G. Assayag and S. Dubnov, *Mathematics and Music: A Diderot Mathematical Forum*. Berlin: Springer, 2002, ch. Universal Prediction Applied to Stylistic Music Generation. [Online]. Available: http://articles.ircam.fr/textes/Dubnov02a/

[15] J. Gillick, K. Tang, and R. M. Keller, "Learning jazz grammars," in *Proceedings of the SMC 2009 - 6th Sound and Music Computing Conference*, Porto, Portugal, 2009.

[16] J. R. Gillick, "A clustering algorithm for recombinant jazz improvisations," 2009.

[17] G. Cabral, J.-P. Briot, and F. Pachet, "Incremental parsing for real-time accompaniment systems," in *The 19th International FLAIRS Conference, Special Track: Artificial Intelligence in Music and Art*, Melbourne Beach, USA, 2006.

[18] J. Nika and M. Chemillier, "Improtek, integrating harmonic controls into improvisation in the filiation of omax," in *International Computer Music Conference Proceedings*, 2012.

[19] D. Conklin and J. G. Cleary, "Modelling and generating music using multiple viewpoints," in *Proceedings of the First Workshop on AI and Music*. University of Calgary, 1988.

[20] A. Donzé, I. Akkaya, S. A. Seshia, E. A. Lee, and D. Wessel, "Real-time control improvisation for the smartjukebox," November 2013. [Online]. Available: http://chess.eecs.berkeley.edu/pubs/1065.html

[7] `http://ptolemy.eecs.berkeley.edu`