Handle: Engineering Artificial Musical Creativity at the “Trickery” Level

Simon Ellis, Alex Haig, Naveen Sundar G., Selmer Bringsjord, Joe Valerio, Jonas Braasch, and Pauline Oliveros

Abstract We embrace a “middle-standard” view of creativity in AI, according to which the driving goal is to engineer computational systems able to “trick” humans into regarding them to be human-level creative. We then report upon three versions of our system of this type in the realm of music: Handle. One of the important hallmarks of our engineering is a commitment to exploiting the power of formal computational logic within the highly intuitive domain of music; accordingly, coverage of the still-incomplete but fast-maturing music calculus is included.
Key words: creativity, music, conducting, artificial intelligence, computation, modelling, logic, intuition, cognitive simulation, spontaneity, music generation

1 Introduction and Plan

We briefly articulate a “middle-standard” position on machine creativity as measured against human creativity (§2), and report on the status of our decade-long attempt to concretize that position in the realm of music, specifically in the form of a multi-talented intelligent agent: Handle. Our middle-standard position falls between what we see as two extreme positions on machine creativity: on the one hand, Cope’s position, which (for reasons we briefly explain below) is acutely “low-standard,” and on the other, a position articulated by Bringsjord et al.: a “high-standard” one according to which it’s probably impossible that any standard computing machine be creative.

After placing our stake in the “middle ground,” we summarize three versions of Handle, one for each of three types of musical creativity from among the full array we ultimately seek to reach. Our ultimate goal is simply stated: For every significant and determinate type of musical creativity seen in the associated human sphere of music, and every level reached by humans in each type, Handle will match the type and level — in the “middle-standard” sense of ‘match.’

As to the three types discussed in the present, short chapter: First is Handle as conductor (Handle$_{cond}$), introduced at C3GI 2012 in Montpellier, France (§5.1). Handle$_{cond}$ is designed to be able to interpret and provide feedback on a human performance of a score-based, classical solo piece of music. Second is Handle as jazz musician (Handle$_{jazz}$), summarized in section 5.2, engineered to able to join humans in the production of improvised, “free” jazz. Third is Handle as film composer (Handle$_{film}$); here our engineering, still very active and more suggestive than mature, centers around the composition of music to suitably accompany narrative expressed cinematically. Handle$_{film}$ is summarized in section 5.3.

One of the important hallmarks of the Handle project is an unwavering commitment to exploiting the power of formal computational logic within the highly intuitive domain of music. Accordingly, the music calculus, $\mathcal{M}$, a first-order sorted modal logic, has been invented as a means of modeling certain aspects of musical creativity and cognition. A significant portion of the sequel (viz., §3) is devoted to the presentation of parts of this calculus.

We turn now, as planned, to staking out our aforementioned “middle standard.”

2 Our “Middle-Standard” View of Creativity

In earlier work, Bringsjord, joined by Ferrucci and Bello, argues that while machines can’t be genuinely creative, at least in the literary sphere they can nonetheless be en-
engineered to *seem* to be creative [6]. This two-part position is partly philosophical in nature (based as it is upon *a priori* reasoning), and partly engineeringish (based as it is upon producing a computational artifact capable of generating compelling short-short stories: Brutus). On the philosophical side, in order for a machine to be genuinely creative (*creativity*)$_G$, it would need to pass the so-called “Lovelace Test” (LT), which entails that what the machine does cannot be anticipated by the designer of this machine [6]. On the engineering side, it’s enough for the storytelling machine to trick human readers, in Turing-testing-style, into believing that the stories produced by this machine were produced by creative humans (*creativity$T$*). The Handle project is, like Brutus, based on a direct analogue of this two-part position: *viz.*,

\[\text{P1} \quad \text{Computing machines can’t be genuinely creative in the musical sphere. This means that no AI system or agent can be a genuinely creative conductor or composer. Nonetheless …}\]

\[\text{P2} \quad \text{from an AI-engineering point of view, (a) it’s enough to aim for a machine conductor or composer able to trick human listeners, in Turing-testing-style, into believing that the music produced/guided by this machine was produced by genuinely creative humans; and (b) such a creative$T$ conductor and/or composer can in fact be engineered within the foreseeable future.}\]

The work described herein is of course directly in line with P1 and P2, and is intended to empirically demonstrate, eventually, the truth of P2 (b).

Our “middle-ground” approach to mechnical creativity differs radically from the approach advanced by Cope, a longtime researcher of the first rank working in the intersection of AI and musical creativity, who abides by a “lower” definition of creativity. To confirm this, we need only turn to Cope’s *Computer Models of Musical Creativity* [10], where he tells us that for him creativity is merely “[t]he initialization of connections between two or more multifaceted things, ideas, or phenomena hitherto not otherwise considered actively connected” (Cope 2005, 11). Immediately after giving this latitudinarian definition, Cope provides a series of examples of his brand of creativity in action. His last example is the solving of the following puzzle:

> “I have three sons whose ages I want you to ascertain from the following clues. Stop me when you know their ages. One, the sum of their ages is thirteen. Two, the product of their ages is the same as your age. Three, my oldest-in-years son weighs sixty-one pounds.”

> “Stop,” says the second man, “I know their ages.”

> What are their ages?

Under the assumptions that: (i) the second man is an adult, and hence—in our culture—at least 21 years of age; (ii) the second man couldn’t deduce the answer after the second clue; and (iii) the second man knows his own age, it’s possible to provide an outright proof that the correct answer is 2, 2, and 9. In an informal nutshell here, the reasoning runs as follows: Of the permutations of three numbers $n$, $m$, and $k$ that sum to 13 and have a product that’s at least 21, the only two that produce the same product (36) are: 1, 6, 6 and 2, 2, 9. Since in the former case there is no oldest, we are left with the latter as the only possibility. Since, using standard formalisms in logic-based AI [4], we have engineered a machine able to
find and certify a formal proof of the argument just given, it’s clear that a theorem-prover-based program able to solve this problem would not be creative. The reason is that the designer of such a computer program wouldn’t be surprised in the least when a formal proof expressing the argument is found. In addition, such a program wouldn’t be creative, for the simple reason that cracking such puzzles is precisely the kind of thing humans expect computers to be able to do, while humans, save for a select few trained in formal logic, have quite a bit of trouble with such puzzles.

Despite the fact that we anchor our r&d to “trickery,” we do make one assumption about the internals of a creative machine, and one general assumption about the epistemic context of any computational artifact that we produce.

Regarding internals, we assume that the computing machine that is a candidate for creativity have at least the minimum representation-and-reasoning power of quantified epistemic logic. This minimality condition, and the methodology that accompanies it, have been defended elsewhere [3]. The condition reflects Bringsjord’s affirmation of Piaget’s seminal position that mature, general-purpose human cognition (including, most prominently, problem-solving) consists of processes operating largely on formulas having at least the expressivity of formulae in full first-order logic [14]. Given the affirmation of the condition in question, the present work reflects a desire to engineer machines that are creative: they are both creative and their internal processing conforms to Piagetian concepts of general intelligence and creative problem-solving. For more on this machines that embody the first condition, see [8].

What is our second assumption? We assume that computational creativity cannot be formalized and engineered without yet another nod in the direction of logic, and specifically epistemic logic. To see our second assumption, consider a person/agent A who has produced an artifact a through some creative process p. Any formal model of A’s cognition before, during, and after the creative process should by our lights have sufficient representational capacity to let us conclude, or at least consider, whether:

1. A knows that A itself has produced a in a creative manner.
2. A believes that a has not been produced by any other agents.
3. A knew what existed and what did not exist before it started the process that produced a.
4. A desired that it needed to create an artifact to fulfill some need.
5. A intended to engage in a series of actions making up the process p.

The above (informal) micro-theory is admittedly incomplete, but does seem unavoidable, and suggests that higher cognition surrounding creativity that can be succinctly captured with an expressive formal logic. We do not claim here that we have invented and specified this logic, but we have taken appreciable steps toward doing so, by way of our modeling framework, $\mathcal{DCEC}$, upon which $\mathcal{M}$ is based. We turn now to $\mathcal{DCEC}$ and $\mathcal{M}$. 
3 The Music Calculus

While considerable work has been devoted to modeling music at various levels, from the raw signal-processing stage to representing hierarchical structures, modeling of the cognitive, social, and doxastic dimensions of music has not been carried out. We provide a small glimpse of the foundations of our approach to constructing the music calculus $\mathcal{M}$; these foundations give at least a provisional account of these three dimensions. Why do we need such a formalism? As we begin to examine the act of musical conducting in a bit more detail, we begin to see why:

Consider a simple situation in which there is a composer $c$, a performer $p$, a listener $l$, and a conductor $h$. The composition, or score, in question is $\text{score}$. The performance of the score by $p$ is $\text{performance}$. Composer $c$ creates $\text{score}$ with the intention of inducing a single emotional effect $\text{effect}_1$ in the listener of the piece, $l$. Performer $p$ has a belief that the composer intends the music to draw out $\text{effect}_1$ in $l$, but performer $p$ might want his performance to have effect $\text{effect}_2$ on $l$. The conductor $h$ might in turn have beliefs of what the composer and the performer intend, and $c$ might have their own intentions for the performance. Each participant in such a scenario can have further iterative beliefs: for example, the conductor believing what the performer believes the composer intended the performance should be. The conductor should also have an understanding of emotional effects and their inter-relations. For example, $h$ should know that a melancholic effect is incompatible with a joyous effect. Such knowledge of effects should allow the conductor to dynamically alter a performance to elicit compatible effects. …

Obviously, even this simple, informal analysis reveals that cognitive, social, and doxastic factors are quite real, and quite central. Our music calculus, designed to allow formal capture of such factors, is based on the cognitive event calculus ($\mathcal{CEC}$), which we review briefly now.

The $\mathcal{CEC}$ is a first-order modal logic. The formal syntax of the $\mathcal{CEC}$ is shown in Figure 1. The syntax specifies sorts $S$, signature of the function and predicate symbols $f$, syntax of terms $t$, and the syntax of sentences $\phi$. We refrain from specifying a formal semantics for the calculus as we feel that the possible-worlds approach, though popular, falls short of the tripartite analysis of knowledge (Pappas [16]), according to which knowledge is a belief that is true and justified. The standard possible-worlds semantics for epistemic logics skips over the justification criterion for knowledge.\footnote{The possible worlds approach, at least in its standard form, also suffers from allowing logically omniscient agents: agents which know all logically valid sentences.} Instead of giving here a full formal semantics for our calculus based in a formalization of justification, we specify a set of inference rules that capture our informal “justification-based” semantics.

We denote that agent $a$ knows $\phi$ at time $t$ by $K(a,t,\phi)$. The operators $B$, $P$, $D$, and $I$ can be understood to align with belief, perception, desire, and intention, respectively. The formula $S(a,b,t,\phi)$ captures declarative communication of $\phi$ from agent $a$ to agent $b$ at time $t$. Common-knowledge of $\phi$ in the system is denoted by $C(t,\phi)$. Common-knowledge of some proposition $\phi$ holds when every agent knows $\phi$, and every agent knows that every agent knows $\phi$, and so on ad infinitum. The Moment sort is used for representing timepoints. We assume that timepoints are isomorphic with $\mathbb{N}$; and function symbols (or functors) $+,-,$ relations $>,$ $<,$ $\geq,$ $\leq$ are available.
The CEC includes the signature of the classic event calculus (EC) (see Mueller’s [15]), and the axioms of EC are assumed to be common knowledge in the system [1]. The EC is a first-order calculus that lets one reason about events that occur in time and their effects on fluents. The CEC is versatile: it provides a formal account of: mendacity [9], the false-belief task (modeled by Arkoudas and Bringsjord in [1]), and the mirror test for self-consciousness, described in [7]. The latter can be consulted to read more about the calculus.

Our preliminary music calculus has at its core an EC-based hierarchical representation of the syntax and semantics of music. To our knowledge, this work represents the first attempt at modeling the hierarchical structure of music in the event calculus, or any augmentation thereof.

While the syntactic hierarchical structure of music has been commented upon in [18, 17], there has been precious little study of the compositional or hierarchical semantics in music. Our calculus is intended to remedy this. Our representation also draws upon observations that music exhibits syntactic structure similar to that found in natural language. The alphabet of our music consists of events representing idealized notes combining information about the pitch, time, duration, and timbre of the note. This is exactly similar to the CHARM representation described in [18]. The CHARM system allows much leeway in how such events can be combined together to form hierarchical structures. We impose some constraints that stipulate that such structures must correspond to some abstract syntax:

1. Events in music must have some syntax with which they can combine with other events in music;
2. events in music must have semantics or meaning which interact with the meaning of other events to produce a composite meaning for the whole musical piece.
To this end, we use a representation inspired by the Combinatory Categorial Grammar approach to modeling meaning in natural and formal languages. (See [23] for a quick introduction to the CCG formalism.) Informally, each word in a language is assigned an expression in the typed lambda calculus. The types also specify one of two possible directions in which the lambda function can take arguments. The types allow certain parses of sentences to be ruled out. The meaning of a piece of text is one of the many functional reductions that can be carried out.

The following example illustrates this. The word ‘John’ has syntactic type $NP$, that is, noun phrase, and has semantic value $john$; similarly, ‘Mary’ has syntactic type $NP$ and semantic value $mary$. The word ‘loves’ is a bit more complex. It has syntactic type $(S/NP)ackslash NP$, which means that the word ‘loves’ combines with an $NP$ on the left to give a phrase with type $(S/NP)$; it then combines with an $NP$ on the right to give a phrase of type $S$, which is of course a complete sentence. The word ‘loves’ has a lambda function as its semantic value; this function indicates the operations we just described. The following is a parse tree for “John loves Mary”, which results in an analysis that gives us $\text{loves}(\text{john}, \text{mary})$ as the meaning of the whole sentence at the bottom of the parse.

![Parse Tree](image)

We observe that the CCG formalism can be adapted to music to enable us to handle semantically rich theories of music which can go beyond shallow syntactic forms in music to the deep meaning of musical pieces.

![Signature of Music Calculus](image)

Figure 2 shows a part of the signature of our preliminary music calculus. We have self-explanatory sorts for representing different aspects of the musical universe. Note has its usual standard interpretation. A $\text{Score}$ is a sequence of notes formed using the function symbol $\text{add}$. A $\text{MusicParticle}$ is a note played at a particular moment and can be considered the simplest $\text{MusicPhrase}$. Simpler $\text{MusicPhrases}$ combine in myriad ways to form complex $\text{MusicPhrases}$; this is represented using $\text{combine}$. The rendition of a $\text{Score}$ using a $\text{Recommendation}$ in a performance re-
sults in a MusicPhrase. The music phrases have meanings Meaning which form a subset of the lambda expressions; the meanings combine using reduce. The phrases have abstract types Type; the types combine using apply. Allowed combinations of music phrases are represented using allowed. Recommendations by the conductor are represented using objects from the sort Recommendation. Simple machinery for representing affects is achieved using the sort Affect and the predicate symbol feels. We model affects as a subset of fluents in the event calculus. The way the meaning of a music phrase produces an affect is to be captured by translates.

With this syntactic machinery we can account for different agents interpreting a piece of music differently. What might be the meaning of a musical piece? It definitely includes affects produced in a listener. In addition to affects, the meaning can include objective properties of the music, such as its tempo, which the current version of Handle can process.

**The General Problem of Conducting:** The general problem of conducting can be stated as follows: Given a score score and the composer’s intention that the listener listener should feel affect a, is there a music phrase p which is the result of performing score with the conductor’s recommendation r such that the meaning of the phrase p translates into affect a in listener?

\[
I(h,t,\text{feel}(\text{listener},a)) \Rightarrow \\
\exists p : \text{MusicPhrase} \ r : \text{Recommendation}.
\]

\[
(B(h, t, \text{performance}(r, \text{score}) = p) \land \text{translates}(\text{meaning}(p), a))
\]

What axioms do we need to enable the conductor to determine his actions? At a minimum, we need a rule specifying combination of the music particles into music phrases. Axiom \(A_1\) states that two music phrases can combine if and only if their syntactic types let them combine. If they combine, the combined phrase has syntax and semantics dictated by the original pieces.

\[
\forall m_1, m_2 : \text{MusicPhrase} \\
\quad \text{allowed}(m_1, m_2) \\
\quad \iff \\
\quad \exists m : \text{MusicPhrase}. \ \text{combine}(m_1, m_2) = m \\
\quad \text{type}(m) = \text{apply(type}(m_1), \text{type}(m_2)) \land \\
\quad \text{meaning}(m) = \text{reduce(meaning}(m_1), \text{meaning}(m_2))
\]

We need knowledge capturing how the meaning of music translates into affects in agents. Before formalizing this, we need an axiom stating that musical meanings produce affects. Axiom \(A_2\) states that if a piece of music has some meaning, there is an event that causes an affectual response in some person. Here start is a defined function symbol giving us the start of a music phrase.

\[
\forall m : \text{MusicPhrase} \ \exists e : \text{Event} \ a : \text{Affect} : \text{Moment} \\
\quad \text{initiates}(e, a, t) \land \text{translates(meaning}(m), a) \land t > \text{start}(m)
\]

Axiom \(A_3\) states a basic property of affects: affects have to be instantiated or associated with agents.
The \textit{translates} predicate is supposed to capture the translation or production of affects in agents via the semantic properties of music. Upon some reflection, the reader may suspect that we have swept under this predicate symbol the hard-to-formally-model processes that operate in the production of affects. We expect that, when axiomatized, determining whether \textit{translates}(m,a) holds could be as hard as general-purpose deductive reasoning. Let the axioms governing \textit{translates} be \( \Gamma \). The problem of conducting can be now stated as finding an \( r \) such that:

\[
\{ A_1, A_2, A_3, \ldots \} \cup \Gamma \vdash \exists p : \text{MusicPhrase} \ r : \text{Recommendation}. \\
(\text{B}(h,t,\text{performance}(r,\text{score}) = p \land \text{translates}(\text{meaning}(p),a)))
\]

4 The Handle Trajectory

Handle was originally a rather humble logico-mathematical component of a music-creating entity known as CAIRA [for more information on CAIRA and “humble” Handle, see [12]]. CAIRA is powered in significant part by human musical creativity, and, as explained above, our goal is a standalone AI that demonstrates across-the-board musical creativity. The first step in the engineering devoted to creating such as AI was reported by by Ellis and Bringsjord; see Figure 3 for the simple, initial architecture of Handle, conceived as an artificial conductor. As promised, we now describe three versions of Handle, beginning with the conductor version.

to the work reported on herein is given in Figure 3.
5 Three Versions of Handle

5.1 Handle as Conductor

Handle\textsubscript{cond} is both a microcosmic version of the logic-based parts of CAIRA and a standalone creative\textsuperscript{T}\textsuperscript{+} machine conductor. Can a computing machine “understand” music and reason from that understanding to the direction of a great conductor, itself issued in real time so as to improve the performance in question? While we are confident the answer is Yes, the only way to provide via engineering an existence proof of this affirmative answer is to start humbly, and gradually scale up. Accordingly, Handle\textsubscript{cond} was created around a single pianist playing a short piece, and specifically to investigate the understanding and “conducting” of this playing. A screenshot of Handle\textsubscript{cond} in action is shown in Figure 4.

Handle\textsubscript{cond} has two major components; the first is an audio analysis module running within MATLAB that controls low-level audio and signal processing routines on incoming live or recorded audio. This module passes information, using a predetermined protocol and format, to the musical calculus system, which runs via Common Lisp. Handle\textsubscript{cond} is capable of computing the tempo of live or recorded audio using the system described in [11]. The result of this computation is then passed on to the reasoning system, which in turn determines whether the song is being played at a tempo appropriate for the audience and context. Figure 4 shows Handle\textsubscript{cond} responding to a performance of the Prelude in C major from Book 1 of Bach’s The Well-Tempered Clavier by asking for it to be replayed at a slightly faster tempo.

Fig. 4: Sample Output from Handle\textsubscript{cond}
5.2 Handle as Jazz Musician

Whereas Handle\textsubscript{cond} is designed to work with score-based, structured music with minimal deviation from a known form, Handle\textsubscript{jazz} is designed to explore the field of unstructured music in the form of improvised, free “fusion jazz.”

As formerly, the architecture of Handle\textsubscript{jazz} consists of two major components, one running in MATLAB and the other in Common Lisp. The most notable change is that the MATLAB component is the controlling client, rather than the Lisp system as formerly. The MATLAB client interfaces with FILTER and MaxMSP via a simple set of OpenAudio commands sent periodically via the network, which provided information on the current tension level of the three players as integers in the range 1-7. These values are used to look up the current ‘perceived state’ of the performance for FILTER and appropriate information is returned. If, however, the state had not been encountered, a best guess is made, derived from a weighted nearest-neighbor heuristic, and the state is marked for formal calculation using an implementation of the music calculus in the Lisp-based SNARK automated theorem prover. Due to the length of time required to perform such operations, all logic calculations are currently done off-line between performances. An overview of the instantiation of the Handle architecture corresponding to the work reported on herein is given in Figure 5.

![Fig. 5: Instantiation of Handle\textsubscript{jazz} Architecture](image)

Handle\textsubscript{jazz}, as the logical reasoning portion of the larger CAIRA system, accepts inputs from three audio sources: two of these correspond to two human players and the third is the output from FILTER. Computational Audio Scene Analysis (CASA) is used on all three audio streams at various points during the piece to extract data on various features of each player’s performance. These features —
Table 1: Ensemble states as determined by TensionB. A, B, and C represent the calculated state of Musicians A, B, and CAIRA, respectively. CAIRA may disregard the recommendation and respond differently. CAIRA must decide which state to prefer. States are ordered hierarchically such that overlap goes to the higher-ranking state [2].

<table>
<thead>
<tr>
<th>Ensemble States</th>
<th>Musician A</th>
<th>Musician B</th>
<th>CAIRA C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solo A</td>
<td>A &gt; B + 1</td>
<td>B + 1 &lt; A</td>
<td>C + 1 &lt; A *</td>
</tr>
<tr>
<td>Solo B</td>
<td>A + 1 &lt; B</td>
<td>B &gt; A + 1</td>
<td>C + 1 &lt; B *</td>
</tr>
<tr>
<td>Solo C</td>
<td>0 &lt; A &lt; 5</td>
<td>0 &lt; B &lt; 5</td>
<td>??</td>
</tr>
<tr>
<td>Low Level Tutti</td>
<td>0 &lt; A &lt; 5</td>
<td>0 &lt; B &lt; 5</td>
<td>??</td>
</tr>
<tr>
<td>High Level Tutti</td>
<td>A &gt; 4</td>
<td>B &gt; 4</td>
<td>C &gt; 4 *</td>
</tr>
<tr>
<td>Ending</td>
<td>A = 0</td>
<td>B = 0</td>
<td>C = 0 *</td>
</tr>
<tr>
<td>Uncertain</td>
<td>None of the above</td>
<td>??</td>
<td></td>
</tr>
</tbody>
</table>

$tension_B$, tensionMIR, dynamic, tempo, valence and activity — are mapped to an integer scale in the range 0–6 [2]; using combinations of 3-tuples of these features, information on the current state of the performance is derived. States used in this work and their definitions may be found in Table 1: possible states are solos for players A, B and C, low- and high-tension tutti, ending and an “uncertain” state. The latter state indicates there is insufficient information for HandleJazz to supply meaningful data. When CASA information is supplied to Handle, it returns an integer value representing current performance state of the piece, using the definitions in Table 1. Based on this value, FILTER then adjusts its own parameters to take the recommendation into account in its performance. In the work described, only tensionB (as defined in [2]) is used for state calculations.

As is the case for human players, HandleJazz has gaps in its knowledge when asked for its opinion on a state it has not encountered. In such cases it attempts an “intuitive leap,” using a predetermined heuristic to calculate a “best-guess” response. If such a guess is made, it is stored in the knowledge-base together with a marker indicating it was a guess. At the end of the performance, all states where a guess existed are passed to SNARK for formal proof, and the resultant values are stored permanently in the knowledge-base. Figures 6 and 7 show output from HandleJazz operating over a short piece of music and the beginning of the resulting SNARK proof resulting from an unknown state, respectively.

HandleJazz operates primarily by logical deduction of state using axioms derived from predetermined rules. The rules given in Table 1 are used to construct premises in first-order logic by which one may prove each state. Numerical values received via the CASA system are mapped to a corresponding player-level value (0 = A, 1 = B, etc.); these are used for comparisons during theorem-proving, replacing the ordinary arithmetic operators which SNARK cannot reason over.

The following are some key examples of premises in FOL used to define the music states [22], some of which are visible as SNARK axioms in Figure 7.
1. A person has a solo iff they have the highest level and there are no other players within one level.

\[ \forall p_1 : \text{Person} \ Solo(p_1) \leftrightarrow \exists l_1 : \text{Level} \ \text{MaxLevel}(p_1, l_1) \land \\
\neg \left( \exists p_2 : \text{Person} \ \exists l_2 : \text{Level} \left[ \text{Level}(p_2, l_2) \land \text{WithinOne}(l_1, l_2) \land (p_1 \neq p_2) \right] \right) \]

2. There is a low-level tutti iff every player has a level greater than zero and less than five, and no player has a solo.

\[ \text{Tutti}(\text{low}) \leftrightarrow \forall p : \text{Person} \exists l : \text{Level} \left( \left( \text{Level}(p, l) \land \text{LessThan}(l, f) \right) \land \text{LessThan}(a, l) \land \neg \text{Solo}(p) \right) \]

3. There is a high-level tutti iff every player has a level greater than four.

Fig. 6: Sample Output from Handle\text{jazz}
Fig. 7: Sample Output from SNARK

Tutti\textsubscript{(high)} \leftrightarrow \forall p : \text{Person} \exists l : \text{Level}(\text{Level}(p, l) \land \text{LessThan}(e, l))

4. It is the end iff all players are at level zero.

End \leftrightarrow \forall p : \text{Person} (\text{Level}(p, a))

By determining states logically in this fashion, the agent is guided to play in a fashion that is expected on a social level. Given that the state determined certain weightings within the FILTER machine-learning aspect, however, the agent remains able to improvise freely within the expected range as a response to the other players; in effect, this is the same as a human player reasoning about what the other players are doing and what they intend, and reacting accordingly. Similarly, when the agent is first initialized, it has no knowledge about what combinations of inputs relate to which state and no knowledge about which are required to learn by reasoning about new inputs over time.

Handle\textsubscript{jazz} uses axioms provided and computed in SNARK to derive state values matching to combinations of 3-tuples of musical features. However, again, in cases where precomputed information is unavailable Handle is designed to make an “intuitive leap,” by using one of a number of possible heuristics to guide its judgment.

We note at this point that for the given three-player scenario we may represent each state as a color corresponding to the most extreme coordinates for a feature 3-tuple. For example, a solo for player A corresponds to red (6, 0, 0 \rightarrow 255, 0, 0), Solo B is 0, 6, 0, which encodes to green, and Solo C is 0, 0, 6, blue. Low-tension tutti
is represented by grey (calculated by finding the centroid of the state), high-tension \textit{tutti} is white, end is black, and uncertainty is determined by the coordinate values themselves. After computing all state combinations, such a representation appears as in Figure 8.

Fig. 8: Chromatic Representation of Complete 3D State Cube for tension_B

Our approach for resolving an unknown state at runtime is to use a \textit{nearest-neighbor} algorithm. Given the (safe) assumption that the states for all 8 corners of the state cube are absolutely known \textit{ab initio}, the value of any unknown state may be derived by taking the average of the sum of the distance vector between it and each (fully known) corner. As more and more states are derived and the knowledge-base becomes progressively more complete, however, we would prefer to use more accurate information; that is, state values which are more proximate to the unknown state. These 8 cubes, nearest to the unknown state in diagonal distance, are determined using a simple vector approach and the mean of their values returned. Thus, given an unknown state $\alpha$ with coordinates $(\alpha_x, \alpha_y, \alpha_z)$ and $\langle x \rangle, \langle y \rangle, \langle z \rangle$ vectors of the co-ordinates of all known states, the 8 nearest neighbors of $\alpha$ can be found as follows:

\begin{align*}
\text{index}_1 &= \text{find}(x[i] \geq \alpha_x \& y[i] \geq \alpha_y \& z[i] \geq \alpha_z) \\
\text{index}_2 &= \text{find}(x[i] \geq \alpha_x \& y[i] \leq \alpha_y \& z[i] \geq \alpha_z) \\
\text{index}_3 &= \text{find}(x[i] \geq \alpha_x \& y[i] \leq \alpha_y \& z[i] \leq \alpha_z) \\
\text{index}_4 &= \text{find}(x[i] \leq \alpha_x \& y[i] \leq \alpha_y \& z[i] \leq \alpha_z) \\
\text{index}_5 &= \text{find}(x[i] \leq \alpha_x \& y[i] \leq \alpha_y \& z[i] \geq \alpha_z) \\
\text{index}_6 &= \text{find}(x[i] \leq \alpha_x \& y[i] \geq \alpha_y \& z[i] \leq \alpha_z) \\
\text{index}_7 &= \text{find}(x[i] \leq \alpha_x \& y[i] \leq \alpha_y \& z[i] \geq \alpha_z) \\
\text{index}_8 &= \text{find}(x[i] \leq \alpha_x \& y[i] \leq \alpha_y \& z[i] \leq \alpha_z)
\end{align*}
While admittedly simplistic, this approach nonetheless conforms to our definition of ‘intuition.’ In order to determine the efficacy of this approach, a simulation was run of over 400 sessions to determine the percentage error of the nearest-neighbor approach by picking 10 random player inputs each session. It can be seen that, as new states were accumulated in the knowledge-base, the accuracy of the nearest neighbor method improves (Figure 9). A comparator is also provided, based on an approach which guesses randomly on uncomputed states.

Fig. 9: Nearest-Neighbor Error vs. Random Guess

We acknowledge that we may stand accused of not properly addressing, or exploring, the concept of ‘intuition’ in Handle\textsubscript{jazz}. We do not deny this. However, in our own defense, we would reply by stating that ‘intuition,’ as it is commonly understood, requires a much larger dataset than the one we were able to use, one that in all likelihood contains a certain amount of conflicting data, together with an appropriate means of accessing and using those data. Handle\textsubscript{jazz} currently functions using only one musical feature, tension\textsubscript{B}, thereby yielding a dataset containing a paltry $7\times7\times7=343$ states; as a consequence, ‘intuitive leaps’ are inevitably barely distinguishable from mere arithmetic computation. But, should we ever have the additional information, the problem will inevitably arise of how we manage it.

One of the most interesting effects to emerge from explorations with Handle\textsubscript{jazz} is the concept of the state cube, shown above in Figure 8. Originally developed by Valerio based on work by Ellis, it appeared to have considerable implications for computation using multiple musical features, even leading to a method of mapping for the use of visual stimuli in the composition of music. (For full information, see [22].) Most importantly for Handle\textsubscript{jazz}, however, it offers a resource to manage and easily explore the large data-space resulting from the many possible combinations of musical features captured hitherto, as well as any we hope to capture in the future.
5.3 Toward Handle as Film Composer

Handle\textsubscript{film} reflects a turn toward a new component of the creative process: composing music for film. Our starting point in this endeavor parallels the starting point of film music as a whole. When silent films first started using music, there was a pianist or organist who would play music along with the video. This music was often improvised, but for those who were not as comfortable with improvisation there were books of music printed with basic instructions indicating what music was appropriate for many different types of scenes; initially, we take a similar approach \cite{20}. Taking a story in the form of video, we represent it in a form that Handle\textsubscript{film} can reason over and gather knowledge about. Then, using this knowledge, Handle\textsubscript{film} can select a piece of music from a predefined database of music that contains both a list of the pieces available and a description of when each piece is most suitable. Once a piece of music is selected, it is then matched to the video. The entire high-level process can be seen in Figure 10.

egin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{process.png}
\caption{Process for Creating Music from Narrative}
\end{figure}

5.3.1 Story Foundation

In 1944, Fritz Heider and Marianne Simmel carried out a seminal and now-famous experiment to study human perception of the behavior of others. The experiment used videos portraying abstract shapes — triangles and circles to be exact — interacting in an ambiguous but highly suggestive manner \cite{13}. Heider and Simmel found that most people watching the video assign personalities to the shapes and a storyline to the video to explain the interactions between these shapes; but they also found that people invent similar personalities and storylines. Handle\textsubscript{film} is currently based on stories we invent for a type of action seen in this original experiment. These stories are relatively short, but are still interesting; this makes them a good starting point for story representation and film composing. The fact that the stories
are natively in a video format also helps with aligning music to the story. In addition, since there is substantial consensus regarding what the stories are “about,” we can partially measure how successful our music generation is by analyzing whether the music enforces the perceived storyline or deprecates it. Lastly, by using the Heider-Simmel stories we open up the opportunity for collaboration with other research teams (see [21]). Figure 11 shows two frames from a Heider-Simmel based story we have created.

Fig. 11: Two Frames from a Heider-Simmel-based Video

(a) Triangle “Examining” Box   (b) Triangle “Celebrating”

5.3.2 Story Representation

In order for Handle\textsubscript{film} to generate music to match a given story, we need a representation scheme for stories that is in a form Handle\textsubscript{film} can use to reason about the story, and to make decisions about what music is most appropriate. Unsurprisingly, we have opted to use formal logic, starting with a proper fragment of the music calculus discussed above; viz., the event calculus (EC), elegantly presented, in connection with stories (specifically with the field of story understanding) in [15]. EC allows us to represent a variety of simple stories and then reason about the events in them, and about some basic emotional constructs related to those events. This information can then be used to determine what kind of music is best suited for the story. Figure 12 shows the event-calculus representation of the story in Figure 11.

As the stories and our descriptions of them become more complex, EC is no longer sufficient to represent all the narratological input that Handle\textsubscript{film} needs in order to compose music to match the relevant story. For example, EC cannot properly represent the beliefs and knowledge of the agents in the story. To represent these facets of the story, we must supplement the EC by using the CEC described previously (also see [7]). Figure 12 shows our first steps in representing the Triangle’s beliefs and emotions via the following simple conditionals:

\[
\begin{align*}
P(Triangle,t,Fireworks) &\Rightarrow B(Triangle,t,Celebration) \\
B(Triangle,t,Celebration) &\Rightarrow Happy(Triangle)
\end{align*}
\]
These conditionals give us a general model of the Triangle’s “psyche,” allowing us to determine what the Triangle’s reactions are to events in the story, and to then generate music based on these reactions using the method described in the next section. Currently, the process of taking the story as we would understand it and representing it in the forms described above is done manually, but Handle_film will later be able to perform this task on its own. This autonomy will require an ambitious level of story understanding, but such capabilities are likely required for creativity in the realm of film composing.

5.3.3 Music Generation

Once the story has been formally represented, we can use a theorem prover to determine what emotions or beliefs are present at any given timepoint by simply requesting a series of proofs. One challenge in this method is that if we ask the theorem
prover to prove that an emotion is present at a given timepoint when it is not, the theorem prover may run forever. However, since our stories are relatively simple, a proof is usually found quickly if one exits; and we can confidently declare that an emotion is absent if it is not found after a reasonable time period; so we still obtain good results. As we scale up to more complex stories, this technique may not be valid unless we are willing to wait a long time for every proof, but for now it achieves our desired goal. Once the emotions, beliefs, or other factors important to the music are determined for every timepoint, we query a knowledge-base using these descriptive factors to determine what song is most appropriate at that time. Currently, our knowledge-base is in accord with *Motion Picture Moods for Pianists and Organists*, a book of songs for silent films like one mentioned previously [19].

For the fireworks scene, Handle knows that the Triangle will perceive fireworks going off and can therefore use the formulae presented above to infer that the Triangle will believe there is a celebration and that the Triangle will be happy. Handle then queries the knowledge-base for music that best matches the “happy” and “celebration” descriptors and returns Giacomo Meyerbeer’s *Coronation March*, the first line of which can be seen in Figure 13.

Fig. 13: Music Selected by Handle for the Fireworks Scene

As we move toward modern-day film music, the interaction between the story and the music becomes much more complex, and therefore our method for generating music will have to become correspondingly intelligent. For example, while most silent music is parallel in nature — meaning that it directly correlates with what is happening in story — it is just as common for modern-day film music to be counterpunctual — meaning that it contradicts what is happening in the story, often to emphasize some specific aspect of the story. To cope with such techniques, we need to expand our representation of the story to include not only what is happening in the story, but what the story is trying to convey. We could then reason about what music best matches the purpose of the story rather than just the contents of the story. Our hope is that a “reversal” of the story-generation approach in [5] will serve as a basis for this deeper form of story understanding.
Acknowledgements: This project is made possible by generous sponsorship from both the NSF (grant no. 1002851, to Braasch, Bringsjord, and Oliveros), and the John Templeton Foundation, to Bringsjord and Govindarajulu. The authors would additionally like to thank the anonymous referees of this paper for their insights.

References