A Model of Performance, Interaction, and Improvisation

Paul Hudak
Yale University
Department of Computer Science
`hudak-paul@cs.yale.edu`

Jonathan Berger
Yale University
Department of Music
`jberger@alice.music.yale.edu`

ABSTRACT: A formal model of performance, interaction, and improvisation is described using *mutually recursive processes.* The recursion captures feedback, mutual recursion captures interaction between players (or larger entities), and the architecture of the recursive network captures hierarchies of interaction.

With this model various classes of interaction can be categorized based on the internals of the engaged processes. These can range from simple articulations and embellishments to complex improvisations. The architecture also allows exploration of novel kinds of algorithmic composition. One of particular interest arises via application of *game theory*: engaged processes can be viewed as players in a game, where currency is manifested as aspects of musical aesthetics, and the rules relate to control of such aesthetics.

1 Introduction

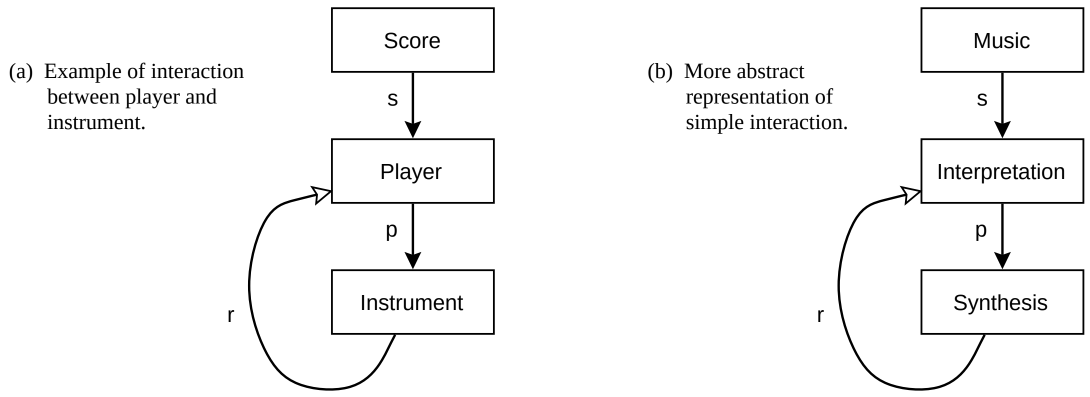
More often than not, computer music generation is viewed as a relatively *linear* process of composition and performance, with performance further broken down into interpretation and sound synthesis. In reality, these processes are interrelated in sometimes complex ways. For example, a performer is engaged in a constant "dialog" with her instrument: what is played depends greatly on what is heard. Similarly, in ensemble or orchestral playing, performers are affected by what the remainder of the ensemble is playing (as well as gestures by the conductor).

In the above examples the effects on performance arise from *interpretation* of a given score. But the effect does not have to be so limited. In particular, in *improvisational* contexts, elements of melody, rhythm, and harmony may be affected as well. In a jazz ensemble, the soloist may play off of rhythmic or melodic motifs introduced by the bass player; or the pianist may "hear" a chord substitution implied by the soloist, and adjust her "comping" (chord voicings) accordingly. In addition, not all interactions are created equal. In an orchestral setting, there is an *hierarchy* of interactions: from interactions between instrument and musician, to those between musicians in a section, those between entire sections, and those between the conductor and individual musicians.

With these observations in mind, we have developed a formal model of performance, interaction, and improvisation using *mutually recursive processes.* The recursion captures feedback, mutual recursion captures the interaction between players (or larger entities), and the architecture of the recursive network captures hierarchies of interaction. We will first describe this model in its most abstract form, then, as an example of application, consider uses of the model in applying *game theory* to describe certain kinds of interaction.

Our approach to capturing the salient features of interaction is fundamentally *algebraic* in nature. To express the algebraic specifications we will use the functional language *Haskell* [HPJWe92, HF92], whose syntax resembles mathematical notation, and whose algebraic datatypes and rich type system have a strong algebraic flavor.[1] This has the additional advantage of quickly yielding an *implementation* of our ideas (Haskell is sometimes referred to as an *executable specification language*). The

[1]We have in fact developed an entire body of "musical ideas" in Haskell, as well as interfaces to midi files, csound, and the NeXTStep MusicKit. We refer to this collection of tools as *Haskore*, and it is freely available via anonymous ftp. The interested reader is referred to [HMGW94] for more information.



(a) Example of interaction between player and instrument.
Score
s
Player
p
Instrument
r
(b) More abstract representation of simple interaction.
Music
s
Interpretation
p
Synthesis
r

Figure 1: Simple Interactions

amount of Haskell syntax used in this paper is minimal, and should present no problem to readers unfamiliar with functional programming.

It is helpful, especially when studying recursive forms of interaction, to draw *diagrams* that illustrate the key communication paths in an interaction. In general these diagrams are isomorphic to their algebraic counterparts, but they provide useful intuition about the nature of interactions that is not as obvious in a lexical specification.

2 Basic Categories of Interaction

To begin, let us consider the simple (at least architecturally) interaction between a solo *player* and her *instrument*, along with the interaction between the player and the *score*. Figure 1(a) shows this interaction in diagrammatic form, where here (and in future diagrams) forward edges are shown with solid arrow-heads, and back edges with un-filled arrow-heads. The equivalent algebraic specification is given by:

$$\mathbf{r} \;=\; \mathbf{instr}\ (\mathbf{player\ s\ r})$$

where r is the *realization* of the performance, and s is the score. Note the recursive nature of this equation, reflecting the recursive interactions between the player process (*player*) and the instrument process (*instr*). Technically speaking, the *solution* to this equation is the least fixpoint of the function:

$$\lambda\mathbf{r} \;\rightarrow\; \mathbf{instr}\ (\mathbf{player\ s\ r})$$

An expression of the form $\lambda x \;\rightarrow\; exp$ is called a *lambda expression*, and evaluates to a (nameless) *function*; it is similar to the form `(lambda (x) exp)` in Lisp.

Because of Haskell's non-strict semantics ("lazy evaluation") the above equation is in fact well-defined (i.e. executable), even if it yields a potentially infinite performance (in which case, of course, only a finite portion of it could ever be realized).

This level of interaction is limited to controlling an instrument's sound, for the purpose of realizing fundamental parameters such as pitch in addition to more subtle issues of articulation, dynamics, and phrasing. Although simple in structure, the processes involved in the loop can obviously be arbitrarily complex, and the overall behavior is perhaps best derived or analyzed using traditional control theory. Although interesting in its own right, this level of interaction is not our main concern; rather, we shall consider several modes of generalization and abstraction.

To begin, a slightly more abstract rendition of this interaction is shown in figure 1(b), where we have re-labelled the "score" box as "music" (to free us from thinking too concretely about traditional scores), "player" as "interpretation" (to permit, for example, computer interpretations of music), and "instrument" as "synthesis" (to admit artificial, in particular computer generated, sounds). The separation of interpretation from sound synthesis is still an important element to this architecture, even in computer generated music. On the other hand, the boundary between the two may vary somewhat from that used by human performers. For example, in our Haskore implementation, interpretation is an instrument-independent concept: instruments "know" how to achieve certain articulations such as legato and staccato in an object-oriented sense. The architecture is rich enough to accommodate a variety of such designs.

Adding More Players One way to generalize our architecture is to consider interactions between more than one player. Figure 2 shows such a situation, whose lexical expression (ignoring the *merge* box, which we will return to later) is given by:

$$\begin{array}{lcl} \mathbf{r}_1 & = & \mathbf{instr}_1\ (\mathbf{player}_1\ \mathbf{s}_1\ \mathbf{r}_1\ \mathbf{r}_2\ \cdots\ \mathbf{r_n}) \\ \mathbf{r}_2 & = & \mathbf{instr}_2\ (\mathbf{player}_2\ \mathbf{s}_2\ \mathbf{r}_1\ \mathbf{r}_2\ \cdots\ \mathbf{r_n}) \\ & \cdots & \\ \mathbf{r_n} & = & \mathbf{instr_n}\ (\mathbf{player_n}\ \mathbf{s_n}\ \mathbf{r}_1\ \mathbf{r}_2\ \cdots\ \mathbf{r_n}) \end{array}$$

We can eliminate the ellipses – i.e. rewrite the equations in closed form – by assuming that the problem specification has as input three *lists* – a list of scores, a list of players, and a list of instruments – and produces as output a list of results (i.e. the r_i). The above equations can then be rewritten as:

```
results  =  aux scores players instrs
  where aux (s : ss) (p : ps) (i : is)  =  i (p s result)  :  aux ss ps is
        aux _        _        _         =  []
```

An expression of the form e : es is a *list* whose head is e and whose tail is es; thus : is like `cons` in Lisp. The first equation for *aux* uses list *pattern-matching* on the left-hand-side to match its arguments, and constructs a new list on the right-hand-side. If the first equation does not match the arguments in a call to a function, subsequent equations are tried in order. The second equation for *aux* matches anything – the underscore character is a "wildcard" pattern.

Using Haskell's pre-defined higher-order function *zipWith3*,[2] we can express our result even more succinctly as follows:

```
results  =  zipWith3 (λs p i  →  i (p s result)) scores players instrs
```

Note that this equation is in closed form, and holds *regardless* of the number of players, scores, and instruments. It is a well-defined recursive equation with a fixpoint semantics similar to that given for the single-player situation.

From Real Time to Virtual Time Another dimension of generalization concerns the treatment of "output" of the synthesis process. Note that the result of the ensemble interaction is a *list* of realizations r_i – but how are these realizations to be combined? The implicit assumption in the previous section was that they represented real-time performances, and would thus be combined in

[2] *zipWith3* is defined by:

```
zipWith3 f (a : as) (b : bs) (c : cs)  =  f a b c  :  zipWith3 as bs cs
zipWith3 f _        _        _         =  []
```

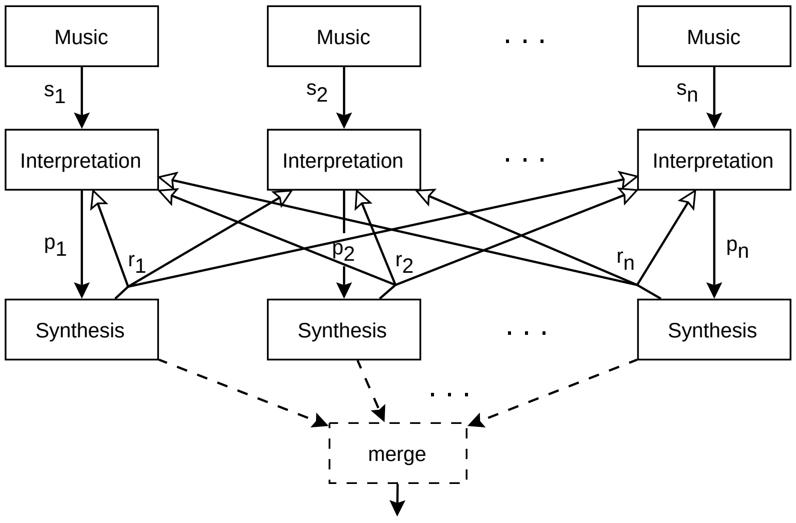


Figure 2: Ensemble Interaction

the obvious way. But to be precise about this we should specify the combining function explicitly, which we shall call *merge*. Thus the real result that we are after is:

merge results

where *results* is as previously defined; figure 2 shows the *merge* operator diagrammatically.

To be even more precise, we must say something about the nature of a realization. If we assume that a realization is a time-stamped sequence of *events*, the purpose of *merge* then becomes clear: it merges n time-stamped sequences into one. The important aspect of this assumption is that now the realizations observed by other players are within a virtual time that does not necessarily correlate with any one player's notion of real time. In particular, it allows the possibility of one player observing the *future behavior* of another player before deciding on its own behavior! By controlling how far each player is allowed to "peak into the future" of each other player, many interesting scenarios for interaction can be imagined. At one extreme, a player observing completely the performance of another allows one to achieve perfect, pre-composed accompaniment; at the other extreme, players can be limited to observations only a fixed time in the *past*; and in the middle we find conventional real-time interaction. Note that it is also possible for a system to *deadlock* if, for example, the current action of each of two players depends on future actions of the other.

Hierarchical Interactions Further generalizations of our model are possible that capture more complex interactive structures, such as the hierarchical structure of an orchestra described in the introduction. Space limitations preclude inclusion of the complete specification of such a structure, but it should be clear from our presentation thus far how to go about defining it.

An Example: the Jazz Quartet It is worth studying at least one form of interaction in detail to help concretize, and later extend, our ideas. Consider the highly interactive *jazz quartet*, consisting (typically) of piano, saxophone, double bass, and drums. In traditional jazz (say, of the bebop era) the individual roles of these instruments are fairly stylized and well developed. The typical performance of a jazz standard is structured by first playing the tune (or "head") from a lead sheet which contains only the melody and chord names; considerable liberty is taken with regard to embellishments and rhythmic alterations to the melody, voicings for and temporal placement of chords, etc. The head is played either once or twice, after which the musicians take turns[3] *improvising* on the basic harmonic structure (the so-called "changes") of the tune. The improviser ("soloist") usually plays through the changes at least 2 or 3 times, but sometimes much longer. Depending on who is soloing, the roles of the remaining members vary considerably:

1. During a saxophone solo, the bass, piano, and drums form the "rhythm section," with the bassist outlining the roots of the changes, and the pianist accompanying ("comping") via voicings that outline the upper structure of the chords (roots and fifths are almost never played).
2. During a piano solo, the saxophonist rarely plays anything, but the pianist's left hand continues its role of comping, with the right hand acting as "soloist."
3. During a bass solo, the piano returns to its full-time comping role, but now in a generally sparser and quieter manner. The drums likewise usually become more subdued.
4. During a drum solo all other players are usually silent; only in this case is the term "solo" used accurately.

After the solo section, the ensemble returns to the head one more time, thus completing the tune.

The jazz quartet thus defined matches the architecture given in figure 2, where $n = 4$ and the *score*s take the form of the lead sheet plus the general song-form schema: head-solos-head. Some notable work has been done on the computational nature of jazz improvisation [AD92, Bag92, JL91], but our work differs in its emphasis on the structure of interactions, especially when combined with the game-theoretic ideas presented in the next section.

3 Game-Playing Interactions

The interactions between the jazz soloist and those accompanying her can be quite involved. Generally the soloist is "in control" of the improvisation, and will introduce melodic motifs, imply harmonic substitutions, and induce rhythmical themes that the remaining players try to accompany in interesting ways. But the communication is bi-directional: the soloist may play off of rhythmic or melodic motifs introduced by the bass player, for example, or a chord substitution used by the pianist may suggest an altered scale to the soloist. The rhythm section in fact often controls gross features such as tempo, dynamics, and intensity.

Although the goal of those involved in improvisation is generally one of *cooperation*, there is also a certain amount of *conflict*. Some of it is unintentional, but a lot of it isn't. This conflict should not be taken in a pejorative sense, but rather in terms of the stakes perhaps associated with a "friendly parlor game." Many such games can be imagined, and we would like to formalize them for the purpose of understanding them better and ultimately engaging a computer in playing them.

Games between musicians need to be expressed, of course, in musical terms. Nevertheless, it is helpful to use terminology and structure given to us by *game theory* [Rap70, LR57]. From this perspective there are three fundamental components to the definition of a game:

1. The game's *currency* is the commodity used to measure a player's success. Conflict is manifested by the fact that when one player gains currency, one or more other players lose currency.

[3]Sometimes more than one player solos simultaneously, but this is more common in larger ensembles, say with an additional horn player.

2. The *rules* of the game specify the allowable *moves* of a player under all possible game *situations*, including a specification of the currency lost or gained by each player for each move.

3. A player's *strategy* is an algorithm (not necessarily deterministic) for playing a game. Thus a strategy specifies, for every conceivable game situation, what a player's move will actually be.

The goal of each player, of course, is to maximize one's profit (or minimize one's loss). This is done at the expense of the other players, thus revealing the nature of the conflict. An *optimal* strategy is one which guarantees, on average, the most profit, *regardless of the other player's moves.*

Game theory is a branch of mathematics that has studied many specific *kinds* of games to determine, for example, conditions under which maximum profits can be guaranteed. For example, one of the most-studied forms of games is the *two-player zero-sum game*, in which the sum of gains and losses of two players is always zero. A fundamental theorem of game theory guarantees the existence of optimal strategies for both players in any finite two-player zero-sum game.

The simplest and most intuitive examples of *musical* games arise under circumstances of relative parity between the players. For example, we can imagine a game between two horn players soloing simultaneously in a jazz quintet, each vying for the center of attention on *aesthetic* grounds, but constrained by rules that prevent a player from "hogging the show" in inappropriate ways (for example playing so loudly so as to drown out the competitor, or purposely interrupting a particularly compelling phrase by the competitor; these are examples of *cheating*). More complex games, but of a similar nature, may occur between players of different instruments.

The architecture for ensemble interaction presented in figure 2 also serves well as a basis for game-playing. To illustrate, we will consider the simple two-player case, whose equational form is given by:

$$\begin{aligned}\mathbf{r}_1 &= \mathbf{instr}_1\ (\mathbf{player}_1\ \mathbf{s}_1\ \mathbf{r}_1\ \mathbf{r}_2)\\ \mathbf{r}_2 &= \mathbf{instr}_2\ (\mathbf{player}_2\ \mathbf{s}_2\ \mathbf{r}_1\ \mathbf{r}_2)\end{aligned}$$

where, as before, we consider r_1 and r_2 to be sequences of time-stamped musical events. Assuming that $instr_1$ and $instr_2$ are fixed, $player_1$ and $player_2$ will embody the *strategies* of the players.

Suppose now that some quantity of type *Profit* is our measure of the profit accumulated in a game (a negative value corresponds to a loss), and the function *util* computes, from the merged result of r_1 and r_2, the profits for each player as a pair $(profit_1, profit_2)$. We can compute a sequence of *instantaneous* profit pairs by applying *util* to successive subsequences of the result, as follows:

$$\mathbf{profits} \;=\; \mathbf{map\ util}\ (\mathbf{scanl}\ (\lambda \mathbf{l\ x} \rightarrow \mathbf{l}{+\!\!+}[\mathbf{x}])\ [\,]\ (\mathbf{merge}\ \mathbf{r}_1\ \mathbf{r}_2))$$

The operator ++ concatenates 2 lists; *scanl* and *map* are best explained by example:

$$\begin{aligned}\mathbf{scanl\ f\ z}\ [\mathbf{x}1,\ \mathbf{x}2,\ ...] &= [\mathbf{z},\ \mathbf{f\ z\ x}1,\ \mathbf{f}\ (\mathbf{f\ z\ x}1)\ \mathbf{x}2,\ ...]\\ \mathbf{map\ f}\ [\mathbf{x}1,\ \mathbf{x}2,\ ...] &= [\mathbf{f\ x}1,\ \mathbf{f\ x}2,\ ...]\end{aligned}$$

For this game to be *finite* there must first of all be a well-defined ending to the game. In the jazz improvisation context described earlier, we could define a game as simply one verse of a solo; the player with the highest score at the end of each round is the "winner" of that round. There must also be a finite number of possible moves at each step in the game. There are certainly a finite number of possible notes on a given instrument, and one might try to argue that note durations are from a finite set of "conventional" values. But once various nuances of loudness, pitch bend, timbre, anticipation and delay, stretching or shortening of note length, etc. are taken into account, a finiteness argument begins to weaken. If one *could* constrain the rules well enough to achieve finiteness, then our goal would be to find values of $player_1$ and $player_2$ such that the pair:

$$(\mathbf{profit}_1, \mathbf{profit}_2) \;=\; \mathbf{last\ profits}$$

is maximized (*last* returns the last element in a list). However, even though game theory tells us that such strategies must exist, the computational aspects of this particular game would be infeasible: the number of choices grows exponentially, just as it does in chess.

Thus we take the approach of finding strategies that perform well over a finite *window* of time. In other words, we constrain the depth of the game tree search to positions only a finite distance into the future. This may in fact correspond to what happens in human interactions, but in any case results in a computationally feasible optimization problem.

Extending the Application of Game Theory: Self Fulfilling Prophecies and Musical Expectations In an earlier paper [Ber91] we proposed a model of music cognition in which a quantifiable degree of realized expectation (dre) is determined by comparative evaluation of a (heuristically determined) "guessed next event" to a constantly updating "next event." The crux of the model was realizing a sudden shift in dre to be a musical surprise which forces a listener to recontextualize the manner in which the music is being perceived. Such shifts (called ambiguous events) are classified by a number of factors including: whether it is patent or latent (i.e., if the event made sense within the context or not); whether it is incipient (that is, it instantiates a musical idea) or is rhetorically prepared by a clearly stated argument embedded within a rhetorical context; its dynamic structural role in its degree of structural import.

Our model of individual listening can be used to examine how multiple players interact by describing first how such expectations dynamically affect certain performance behaviors and then how these individual player's behaviors affect an ensembles overall behavior. We return to our incorporation of game theory and define our jazz ensemble as a multiagent system in which each performer is an agent with independent and autonomous mechanisms to dynamically create contexts.

Adopting terminology from [HH94] we define a jazz ensemble as a system of collective action displaying sporadic regions of instability within a generally stable framework.

An individual player's detection of an ambiguous event triggers a recontextualization of all players' understanding of the current state of the game. This prompts each player to respond accordingly by "picking up on" a musical attribute that gains prominence. This creates a social dilemma of sorts since the ensemble strives for common good while being careful to maintain a minimal (in terms of degree of control) and shifting (in terms of who is in charge) sense of central authority. Thus players oscillate between a contributory state (which is triggered by reaction to a sudden shift in the dre and reliance upon others to continue as long as the game is more or less stable (unless of course, this continues beyond an acceptable length, which we call the horizon length).

The principle characteristic of multiagent systems is the production of complex unified behavior through interactions of local constituents. Ensemble situations, including our jazz quartet, pose the problem of individual players with limited views of the system in its entirety. In an improvisational situation the individual player has a certain degree of control over the amount and import of her contribution. In terms of [HH94] the "interaction strength" is "proportional to the frequency with which they exchange information or 'hints' with each other."

In order to model the interaction of players we add to our localized dre a broader and more general time varying system-stability rating assessed by each player. This variable determines an expectation level ranging from static expectations (an assumption that the system is stable or changing very slowly) to trendy expectations (in which a player believes the system will continue to follow a perceived trend).

We noted above the shifting alliances that result from which agent assumes temporary leadership (i.e. the soloist). We can apply the same attributes to an individual agent as the level of cooperation among agents changes, collective defection from a single group and dynamic creation of sub groups are perceived. Each player has finite patience or ability to perceive a context. We measure this with a horizon length.

Ultimately we attempt to chart the changing knowledge of other's intents (the perceived balance between cooperation vs. competition) within and amongst agents.

Improvisation Viewed as a Special Case of Composition Improvisation can be viewed in some sense as "real-time composition." Indeed, we noted earlier that a suitably flexible notion of interaction allows players to look ahead arbitrarily far into the future. If we additionally allow an arbitrary amount of time to compute a behavior, and allow the score to vary from nothing at all to perhaps a simple description of a musical *concept*, then we have a purely compositional process. The interactions are still plausible exchanges of musical ideas, especially if the abstract view of "interpretation" is adopted as in figure 1(b), and even if considered only abstractly in the mind of the composer. In addition, compositions (or portions thereof) viewed as the *result* of a game are equally plausible. A good example of this might be a phrase with a cannon-like structure: for a given temporal displacement, harmonic considerations constrain the choice of melody, the goal of the game being to maximize the combination of melodic, rhythmic, and harmonic aesthetics.

References

[AD92] C. Ames and M. Domino. Cybernetic composer: An overview. In M. Balaban, K. Ebcioglu, and O. Laske, editors, *Understanding Music with AI: Perspectives on Music Cognition*, pages 186–205. AAAI Press/MIT Press, 1992.

[Bag92] D.L. Baggi. Neurswing: An intelligent workbench for the investigation of swing in jazz. In D.L. Baggi, editor, *Readings in Computer-Generated Music*, pages 79–94. IEEE Computer Society Press, 1992.

[Ber91] J. Berger. A theory of musical ambiguity. *Journal of Computers in Music Research*, II, 1991.

[HF92] P. Hudak and J. Fasel. A gentle introduction to Haskell. *ACM SIGPLAN Notices*, 27(5), May 1992.

[HH94] B. Huberman and T. Hogg. Communities in practice: Performance and evolution. Technical report, Dynamics of Computation Group, Xerox PARC, 1994.

[HMGW94] P. Hudak, T. Makucevich, S. Gadde, and B. Whong. Haskore music notation – an algebra of music, September 1994. To appear in the Journal of Functional Programming; preliminary version available via `ftp://nebula.systemsz.cs.yale.edu/pub/yale-fp/papers/haskore/hmn-lhs.ps`.

[HPJWe92] P. Hudak, S. Peyton Jones, and P. Wadler (editors). Report on the Programming Language Haskell, A Non-strict Purely Functional Language (Version 1.2). *ACM SIGPLAN Notices*, 27(5), May 1992.

[JL91] P.N. Johnson-Laird. Jazz improvisation: A theory at the computational level. In P. Howell, R. West, and I. Cross, editors, *Representing Musical Structure*, pages 291–325. Academic Press, 1991.

[LR57] D. Luce and H. Raiffa. *Games and Decisions.* John Wiley and Sons, New York, 1957.

[Rap70] A. Rapoport. *N-Person Game Theory.* University of Michigan Press, Ann Arbor, 1970.