CHAPTER TEN

Conclusions and Future Work

In the preceding chapters I have described the architecture of Musicat, recounted the history of the program's development, and given numerous examples of the program running. In this chapter I aim to synthesize the results of this work: I review the original aims of the project, and then I describe what Musicat accomplishes in terms of modeling music-listening, as well as what it fails to accomplish. I also list what I consider to be the main contributions of this project.

Musicat is a work in progress, and I have many ideas about how to continue the development of the program, given the lessons learned so far. At the end of this chapter I list a series of suggestions for further steps, ranging from obvious and relatively simple improvements to quite speculative ideas.

Aims of the project

The ambitious long-term goal I have in mind for Musicat is to make the program simulate human listening for a wide range of musical pieces of varying complexity and styles, including not only monophonic melodies but also works for multiple voices and instruments. The current version of the program has the restricted but still incredibly difficult

goal of "listening" to monophonic, tonal, Western, song-like melodies. Not only is music-listening incredibly complex, it is also an idiosyncratic and essentially human activity, making it hard to evaluate a model of it. How should one judge the accomplishments of Musicat?

Although the long-term goal is to simulate human listening, a more central role of cognitive models such as this one is to examine possible mechanisms that form the basis of human thought. In the case of Musicat, the modeling effort hopefully provides some insight into certain aspects of human music-listening. Also, because it is a computer model, Musicat makes concrete some heretofore theoretical aspects of music cognition (such as the importance of analogies in melodic structure).

What Musicat Does Well

The following list of "things Musicat does well" has an intentionally optimistic flavor: I describe some of the best things about the Musicat program, even though I freely admit that in every one of these instances it still has an unbelievably long way to go before it can claim to listen in more or less the same way that humans do. It is easy to see that the latest version of Musicat does much better than the earlier versions (as described in Chapter 9), but it is just as obvious that much more work is needed.

Musicat Behaves Differently on "Bad" Melodies

In Chapter 5, I gave several examples of "bad" melodies. These were "bad" in the sense that they didn't have any large-scale structure: individual notes or measures were quite random. I ran Musicat on those melodies after finishing the current version of the program, and I had some concerns that it might generate just as many groups and analogies for these melodies as it did for well-structured, "good" melodies. Musicat did make more groups than I had hoped, and they were usually based on its default preference to make a perfectly regular

hierarchy of groups with powers-of-2 lengths. Encouragingly, however, it rated many of these groups as quite weak. Moreover, it spotted far fewer analogies in these melodies than it did in "good" melodies, where were covered in the following chapters.

In general, Musicat seems to find less structure in "bad" melodies than in "good" melodies. While this may seem unimpressive, it is an important sign that the program is "hearing" structure in a reasonable way. It is easy to imagine a program that would take "garbage" melodies and would generate complex structures in its listening performances, but happily, this does not seem to characterize Musicat's behavior, at least not for these examples. However, as discussed below, Musicat's understanding of tonal implications in music is very limited, so it does indeed respond in essentially the same way to a melody that is "bad" in terms of pitch structure as it would to a normal melody. If I were to insert random "errors" in a melody, say, by adding accidentals in tonally inappropriate places, the program would not notice the mistakes. Despite this blatant weakness in the pitch domain, the program's different types of performances on "good" versus "bad" melodies is encouraging.

Musicat Generates Hierarchical Grouping Structures

Music theories such as GTTM have emphasized the importance of hierarchical grouping structure in music. The analytical techniques of GTTM, in particular, result in tree structures that describe musical works in terms of hierarchically-nested groups of various durations, ranging from groups of just a single note or two all the way to groups spanning the entire duration of a piece. To the best of my knowledge, few computational models generate hierarchical structures of this nature (with some exceptions, such as explicit attempts to generate GTTM analyses by computer (Hirata et al., 2007)). Recent computer modeling more typically aims to generate single-level segmentations; that is, music is partitioned in time but groups are not nested within other groups.

Musicat generates hierarchical grouping structures as it "listens" to a melody. Such structures are important not only because they appear in music notation (beams and phrases, for instance) and in music theory, but also because they are crucial in music cognition: limitations of human memory necessitate chunking of musical structures. Chunking allows people to make sense of musical structures of widely varying durations. More specifically, chunking allows people to understand and remember musical segments that, at the level of notes, are far too large and complex to represent at one time in working memory.

Musicat Listens in Real Time with Limited Working Memory

Not only does Musicat *generate* hierarchical grouping structures, but it does so *in real time*. If we were not interested in cognitive modeling, then there are obvious ideas from computer science that might be used to partition a melody into segments if the entire melody were given to an algorithm as a whole — see, for example, my own work on audio partitioning (Nichols & Raphael, 2006) — and one can imagine using a similar algorithm to create a hierarchical grouping structure. However, this type of algorithm is not cognitively plausible, obviously, because it requires the whole melody to be analyzed at once. (Furthermore, without a mechanism for analogy-making, I doubt that such an idea would work well in any case.) Human music-listening involves making hierarchical grouping structures as a piece is heard, despite the limited capacity of working memory.

Musicat generates hierarchical structures in real time without modifying grouping decisions made far in the past. Small changes to recent groups are allowed (and indeed are necessary, to provide Musicat with its fluidity), but as groups fade into the past, they become fixed and are eventually treated as inviolate. These chunked memory items from the past retain the ability to be referred to by analogies or incorporated into larger groups. Thus, even as time flows, with new notes being "heard" and older notes fading into the past, Musicat can

form ever larger structures whose constituent elements would be too large to store in working memory. It is remarkable that people can make sense of musical works whose durations are orders of magnitude greater than that of the phonological loop, and with number of notes orders of magnitude greater than Miller's magic number seven (plus or minus two); it is chunking that makes it possible. Musicat does not model the *specifics* of human working-memory limitations in great detail, but it does give a concrete implementation of mechanisms that can build large hierarchical structures despite the sorts of constraints imposed by human memory.

I designed Musicat to model real-time listening, but "real time" was meant to be simulated: Musicat was given notes one at a time at a constant rate, but I did not worry about the specific details of computation time or processor speed. I would have been happy if the model could "listen" to a melody even if the melody had to be presented at an extremely slow tempo. However, because Musicat is constrained to use a small amount of working memory and considers chunked groups from the past instead of "hearing" all the notes of a melody at once, it is quite efficient. Indeed, when the computationally intensive graphics-drawing code in the user interface is disabled, Musicat can "listen" much faster than real time (*e.g.*, it can simulate listening to a 20-second long melody in a fraction of a second). This speed is possible because a fixed number of codelets are run for each beat, and no codelets have loops that will run for a long time because the size of any loops has an upper bound determined by the small number of items in working memory along with a small number of accessible groups in the past. Every codelet runs quickly, and the time complexity of the program for each incoming note is constant — $O(1)$ in computer science's "big-O" notation — because of the small upper bound on working memory size. For long melodies, the number of groups in the past does grow, but access to these could be, in principle, constrained so as to ensure constant time complexity per note even for very long melodies.

This is necessary for a cognitively-plausible model of human listening — obviously, people do not require more computation time per note as a melody gets longer. Musicat's design allows quite naturally for scaling-up to long melodies: melody-length is a non-issue, thanks to the constraints imposed by real-time listening.

Musicat Notices Different Things on Different Runs

Since Musicat is a stochastic program, each run is different. While randomness is a core component of the FARG architecture, it has unique implications for perception in the temporal domain of music. First, in each run, Musicat focuses its perceptual attention on slightly different structures. While this is also the case in other FARG programs such as Copycat, which might pay more attention to certain aspects of letters in one run than in another, or Capyblanca, whose simulated random eye saccades cause the program to pay attention to different squares on a chess board in different runs, Musicat has a fixed amount of time to allocate to structure perception during each run. Copycat or Capyblanca can run for quite a while on a given problem, and the programs themselves regulate (at least to some extent) the amount of processing time that is used per problem. Musicat, however, has its processing-time determined by the rate at which new musical notes are heard. This relatively limited amount of processing time results in runs that are very diverse, because of Musicat's different focus of attention for different runs. A second implication of Musicat's randomness combined with real-time listening is that the perceptual choices Musicat makes early in a run can have significant consequences later in the same run. For example, an analogy can only be formed between measures 1–2 and measures 9–10 for some melody if the group **(1–2)** was formed while Musicat was listening to the first several measures; after enough musical time passes, the program can no longer go back and modify the grouping structure of measures heard earlier.

Musicat's ability to focus on different parts of a melody on different runs reminds me of my own listening experience: I know that I pay attention to different things during different sessions of listening to the same piece. Sometimes this is more consciously controlled than is the case for Musicat's listening: for example, I might decide to pay attention to an inner voice or a bass line while listening to polyphonic music. Or, in the case of monophonic melody, I might decide to focus my attention on a particular musical motif or even an individual pitch. Musicat does not control its own attention in this way (perhaps a future meta-Musicat could do such a thing), but it does notice different things on different runs, and this also happens with my listening: I might listen to the same recording of a song dozens of times, but, quite haphazardly, I might notice different aspects of the music during different listenings. I suspect that this is one of many things that make some melodies so much more amenable to repeated hearing than one might expect: rather than getting bored from over-familiarity, listeners can discover new things on new occasions.

MUSICAT FORMS ANALOGIES

The most unique thing about Musicat, compared with other models of music cognition, is that it makes analogies between musical structures. Musicat sometimes misses analogies that seem obvious to people, and other times it makes analogies that seem non-justifiable, but on the whole it makes many analogies that seem reasonable to me. For instance, in "Row, Row, Row Your Boat", Musicat generated the chain of analogies **(1–2)↔(3–4)↔(5–6)**, which is a sophisticated and human-like way of hearing the first six measures: measures 3–4 are heard as a development of measures 1–2, and likewise measures 5–6 are heard as a development of measures 3–4. The program also makes larger-scale analogies, such as **(5–8)↔(9–12)** or the even-larger **(1–8)↔(9–16)** in "On the Street

Where You Live", thus illustrating that it is able, to some degree, to make sense of musical structure.

Analogy-making is a crucial component of the human experience of listening to music. Even though Musicat's ability to form analogies is terribly limited compared with a human's, the mere fact that it is able to "hear" analogies at all is a success. Additionally, Musicat's ability to form *multiple* analogies within a single melody gives it the potential for hearing melodies in a rather sophisticated way that is evocative of the complexity of human listening. Much of the meaning of a musical work (especially in the case of non-programmatic "absolute" music) derives from the very specific and unique internal vocabulary of musical ideas that exists only within that particular work. Except for trivial cases of exact repetition, such a vocabulary is recognizable thanks to analogies formed between elements of the vocabulary. Analogy-making, then, gives Musicat a preliminary, but promising, way of understanding certain aspects of musical semantics.

What Musicat Does Badly

Admittedly, there are many aspects of musical listening that Musicat does not model well, or at all. Music is highly complex, and Musicat just scratches the surface of music-listening, so the list of deficiencies in this section could be made arbitrarily long. Therefore, I focus on things that Musicat currently does badly but that I believe it *should* be able to do with some improvements but without any radical changes to the current architecture. Musicat doesn't attempt to model the emotional experience of music-listening, for instance, so that topic — among countless others — is not addressed here.

Musicat Forms Groups at Measure Boundaries Only

As I have mentioned many times in previous chapters, in the latest version of Musicat (in contrast with the earlier versions, MusicatH and MusicatW), groups must start or end at measure boundaries (in the case of melodies with pickup notes, this is true if we consider the measure boundaries to be shifted to the left by the total duration of the pickup notes). As a result of this simplification, Musicat became much better at forming large groups and analogies, but this constraint is too restrictive. After all, salient groups (from a human point of view) can occur at the sub-measure level. One particularly important class of examples of this problem is *melodic sequences*: if a sequence is built on a half-measure-long motif, for instance, Musicat is not able to recognize the sequence because it cannot build analogies among such short passages.

Additionally, groups often have a duration that is equivalent to a non-integral number of measures. For instance it is common for a single note to serve a dual function as the end of one group and the start of a new group, as was discussed in Chapter 2 in the section about GTTM. The current restriction prevents this sort of group overlap.

Musicat Misses Too Many Simple Things

There are many musical relationships or places that have group boundaries that are exceedingly obvious to a human yet seem to be imperceptible to Musicat. For example, in "Younger than Springtime", Musicat usually failed to make the analogy **(1–2)↔(3–4)**, linking together the phrases for the lyrics "Younger than springtime are you" and "softer than starlight are you". Even without the lyrics, the analogy between the phrases is blatantly obvious to a human listener. The phrases share such a distinctive rhythm and pitch contour that the analogy seems to scream out at the listener. The program did make the analogy sometimes, which is a relief to me, but it should have made it *every* time. In general, Musicat

does not notice these sorts of extremely salient relationships nearly as easily as it should; it seems that it works too hard trying to form more complicated groups and analogies, at the expense of simple structures.

Musicat Makes Errors in Grouping and Bar-line Thicknesses

Musicat makes two important types of errors in grouping: determining starting and ending points of groups and determining how to combine two groups into a meta-group.

Musicat relies on its perception of bar-line thicknesses (in other words, its perception of hypermetric hierarchy) to make many decisions about grouping. Sometimes it is not flexible enough and doesn't allow a group to cross a pre-established bar line even though that would result in a better group (and as a result, it might result in changing the bar line thicknesses — bar-line thicknesses and groups influence each other in Musicat). For example, in "Sun and Moon", a natural group boundary is *after* the whole note on the word "sky" in measure 9, but Musicat hears a thick bar line *before* measure 9 and always creates groups that end before the whole note. On the other hand, sometimes Musicat is too flexible, and allows groups (especially large groups) to straddle thick bar lines when doing so is inappropriate. Often the program correctly hears the bar line between measures 8 and 9, for example, as being quite thick — after all, many melodies have a strong cadence on measure 8 — but then it will ignore this clue and form a group such as **(7–10)**.

Musicat also has a tendency to make meta-groups whose constituent groups are of mismatched size. For example, sometimes an 8-measure group such as **(1–8)** and a small group such as **(9–10)** will be used to make an out-of-balance meta-group **(1–10)**, without any obvious justification for combining groups of such different durations. Musicat penalizes meta-groups for such a size mismatch, but with the program's current parameters, there are

often other positive factors that override the penalty, and allow the awkward group to be formed.

Musicat Makes Too Many Analogies

Although Musicat sometimes fails to make some nearly trivial analogies, on the whole it errs on the side of making too many analogies. For example, recall the second run of "On the Street Where You Live", in which Musicat made 12 analogies in the first 16 measures, all between various 2-measure groups. Most of these analogies were defensible, and some were stronger than others, but this abundance of analogies doesn't seem to capture the most important things that I hear when I listen to the melody. One problem, I believe, is that this version of Musicat makes analogies only between elements in the Workspace. In contrast, MusicatW could create motif nodes in its Slipnet (*i.e.*, in motivic memory), and then analogies could be formed between groups in the Workspace and groups stored in motivic memory. Many of the analogies between the groups in the Workspace that seem to be extraneous in this version might be better represented as analogies to nodes in the Slipnet. In other words, a revised version of Musicat form *categories* in the Slipnet to represent motifs, and any time it would hear a new instance of a motif it would be reminded of the category. Categorization is analogy-making; reminding is analogy-making. These processes would complement the current analogy-making that only takes place among elements in the Workspace.

Another issue leading to the overabundance of analogies is that Musicat is simply too flexible in forming correspondences between two structures. Often, such flexibility is exactly what is necessary for high-quality analogy-making, but in some cases the program seems to be finding analogies based on far-fetched connections between structures. Analogy-making is,

of course, much subtler than what Musicat can currently do with its analogy objects and analogy-finding codelets.

Musicat Makes Too Few Analogies

Even though I have just finished describing how Musicat makes too *many* analogies, in another sense it makes too *few* of them. Specifically, recall Chapter 4, in which I described how people make analogies across a huge range of levels, from the note level (or even below), up to the level of entre pieces (and even above — Shostakovich's 24 Preludes and Fugues were composed by analogy with Bach's 24 Preludes and Fugues in the *Well-Tempered Clavier*). Musicat makes analogies in which each side of the analogy is at least one measure long, but as discussed above, many more analogies can be found at the sub-measure level. And even at the level Musicat works at, it does sometimes miss analogies that would be obvious to humans. In sum, Musicat makes too many analogies overall at its usual level of analysis, but too few analogies overall when considering smaller levels as well.

Musicat Ignores Many Crucial Aspects of Pitch

Musicat's development followed a curious trajectory: I initially gave the program melodies in which all notes had the same duration (*i.e.*, they were all quarter notes), and in which pitch was the most important component of each note. By contrast, in RhythmCat (and also, to a great extent, the latest version of Musicat), rhythm is king and pitch is a treated as a second-class citizen. This reversal was necessary to make Musicat recognize the more primal similarity of rhythmic patterns, but it makes the program much less sensitive to pitch than desired. For example, people are quite good at detecting "errors" in pitch in simple tonal contexts — a single stray F♯ in a C major folksong that is unknown to the listener will

not go unnoticed. Musicat, however, will be only slightly affected by strange accidentals inserted into a piece, especially if the piece has a repetitive rhythmic structure.

If Musicat listened to polyphonic melodies or to melodies that were paired with explicit harmonic progressions (as opposed to the harmonies that it infers behind-the-scenes), there are obvious techniques that could give Musicat more sensitivity to pitch. For example, the degree of dissonance for each melody tone could be calculated with respect to the harmony using existing techniques such as those in *Tonal Pitch Space* (Lerdahl, 2001). However, there is much more to pitch than consonance or dissonance, and even in the monophonic case (Larson, 2012), pitch movement has subtle and intricate ramifications. Overall, pitch is much more crucial to music listening than is implied by Musicat's current design.

Musicat Doesn't Generate Note-level Expectations

My original plans for Musicat included having the program generate real-time expectations about which notes might come next. I toyed with this idea in MusicatH, and even in MusicatW for a while, but the current version does not make this sort of explicit expectation. Instead of expectations involving specific notes, it makes *structural* expectations, which are simply expectations that a strong group structure will repeat as soon as the group reaches closure. Some amount of melodic repetition is implied here, but at the note-to-note level, the program doesn't care which way the melody might move. This is an important deficiency (although I believe it would be relatively easy to implement note-level expectation at this point), because a program with expectations for notes also has the ability to be "surprised" or "satisfied" with notes as it hears them. Real-time denial of or satisfaction with expectations is a crucial component of the affective listening experience.

Musicat Flails

One of the biggest problems for Musicat is what Douglas Hofstadter has called "flailing": the program sometimes creates excellent groups and analogies, only to destroy them moments later in favor of new and often terrible structures. The program often alternates between several competing grouping structures during a single run, and fails to settle down on what, to a human listener, is obviously the strongest choice. Although Musicat computes strengths for various structures and should be able to tell which among various rival grouping structures is the best, it keeps destroying strong structures. Although the program seems to flail less now than in previous versions, the problem is far from solved at this point. It is worth considering the issue for a moment so that this program (and others) might avoid the problem in the future.

Why does Musicat flail? A more precise question is: why is flailing a problem with Musicat, and not of FARG-architecture programs in general? After all, the notion of a subterranean "fight" between various competing pressures and between rival structures in the Workspace is fundamental to this type of architecture. At the *subcognitive* level, some degree of "flailing" is expected and even (when temperature is high) encouraged. But why don't these many tiny "fights" result in an unstable, chaotic Workspace that never settles down to a good representation? Why doesn't micro-flailing translate into macro-flailing?

In brief, the answer is that although individual codelet actions do indeed act in a rather frenzied, chaotic-seemed way, creating and destroying structures with great speed, a stable structure does tend to emerge as a result of all these actions (at least in well-behaved FARG programs!). Strong structures and regular behavior can result from a seemingly chaotic foundation when there are statistical regularities that bias the system in a particular direction. Take a gas, for instance. The individual motions and interactions of particles of a gas are far too numerous and chaotic to measure, but the aggregate behavior is not chaotic at all: the

temperature and pressure of a quantity of gas contained in a fixed volume are quite simple properties to understand and to measure. So what is it about Musicat that makes its aggregate behavior more chaotic than desired?

I believe that there are two main problems for Musicat: it does not pay enough attention to extremely obvious and even superficial cues, and it does not make use of Workspace temperature and simulated annealing in quite the same way that earlier programs such as Copycat did.

Regarding the first problem, there are certain cases where groups and analogies are exceedingly obvious, and the program should simply jump to a conclusion in each of these cases without spending much time considering other possibilities. Domain-specific knowledge (embodied in codelets) may be particularly helpful in this respect. For example, if a melody moves downwards through a scale in quarter notes and then ends on the tonic in a whole note, the program's default assumption should be to hear the scale and the whole-note tonic as constituting a group. Extremely strong pressure should be required to change that default grouping. The current version of Musicat, however, does not see obvious groups of this and other types quickly enough, and even when it does, it is far too eager to entertain alternate possibilities.

The second problem, having to do with temperature, is related to the previous issue. In Copycat and related programs, alternate possibilities were paid much greater attention when temperature was high. Musicat does use a type of temperature, but, as discussed in Chapter 8, it is computed locally for measures, groups, and the current working memory area, rather than globally, as in Copycat's Workspace. Global temperature in Copycat had an important effect on competitions between rival groups: high temperature made the program more likely to consider alternate groups, whereas low temperature resulted in increased stability for strong structures. The average temperature (or happiness) of the structures in

Musicat's working memory is used in an analogous way to global temperature in Copycat, but perhaps temperature is less useful in this case because the working memory (the past four to eight measures) is quite large and is the host to *many* groups and analogies. Temperature affects competitions between rival groups, but perhaps a more local notion of temperature would be more effective for Musicat. Additionally, simulated annealing in Copycat forced the temperature to descend over time. In Musicat there is no simulated annealing, and although the strengthening of structures as they fade away into the past fills a somewhat analogous role, perhaps the lack of simulated annealing is another contributor to the problem of flailing.

Musicat Destroys Large Structures

This issue might well be considered a version of the flailing problem, but I address it separately. Musicat sometimes creates quite strong and large structures, including groups, analogies, and expectations, but soon thereafter destroys them and lets them go forever. This sort of problem was addressed in the Tabletop program, and as I explained in Chapter 8, I implemented a solution based on that in Tabletop: large strong structures are retained in memory after they are destroyed and they have a chance to be brought back to the Workspace quickly, bypassing the need to recreate all their individual components. However, this idea didn't work very well at all, perhaps because of technical issues related to the temporal nature of the domain. This problem may turn out merely to be a subtle bug rather than a deep issue. I have no way of knowing, at this point.

In the case of expectations, however, there is a special problem to be solved that is somewhat related to the flailing issue. Whereas most of the listening architecture is quite dynamic, with groups and analogies being rapidly created and destroyed until the system (hopefully) stabilizes on a strong structure, expectations sometimes need to be more

persistent, in the sense that they need to remain for quite some time without being destroyed. For example, consider a situation in which the program has heard eight measures of a melody and has formed an expectation for an analogous 8-measure structure to follow. In the current version of the program, a large-scale expectation like this can readily be formed, but to be useful it needs to remain active in the Workspace for the full duration of those eight measures. Unfortunately, even when a very reasonable 8-measure expectation has formed, sometimes Musicat will temporarily form a group of a length that is incompatible with the expectation (such as a 3-measure group), and even if the temporary group turns out to have low strength and is quickly destroyed, its creation may result in destruction of the 8-measure expectation. Thus, for large-scale expectations, Musicat needs a way to balance its need for flexibility in trying out many possible group structures with the need for expectations that can persist long enough to have a tangible top-down effect on perception and can guide the formation of future expected groups.

Musicat Gets Confused by Long Melodies

As was demonstrated with the "Tennessee Waltz" melody (Chapter 7), Musicat gets confused when a melody is too long (say, longer than 16 measures). The program makes groups that span long portions of the melody in strange places without much apparent justification; moreover, meta-groups form and create alien-looking patterns, and analogies often link groups of disparate sizes together. Part of the problem is simply an abundance of options for the program's codelets to consider: in longer melodies, there are more levels of grouping hierarchy present, more options for making meta-groups, and more opportunities for the program to mistakenly make analogies between groups of quite disparate sizes. Most importantly, there is simply more material to consider. Although the size of working memory is limited, the program makes analogies to groups that occurred earlier, and it probably does

so too often. More-limited access to older groups, along with an implementation of motivic memory, should help improve the program's behavior on longer melodies. Because Musicat already can handle melodies that are 8 or even 16 measures long and because the focus of most codelets is on the most recent several measures only, I think that many of the problems associated with scaling up to "long" melodies have been handled already, and I expect that the remaining associated problems are manageable.

Contributions of Musicat

The Musicat program and its listening performances may be the most tangible product of this research, but the problems encountered along the way, the design choices made to address them, and even the problems that the program still has are also useful results. In this section I list what I consider the most important knowledge I can impart based on my work on Musicat. This includes lessons learned in designing the program, new ideas that extend the FARG architecture, and contributions to the modeling of music cognition.

Lessons Learned

The following lessons that I learned while working on Musicat apply to FARG models in general. The final item, however, pertains specifically to melody.

Aim for Deep Superficiality

Perception is incredibly complex and subtle. However, there are some simple perceptual cues that are quite salient and important, and a flexible model should rapidly and reliably exploit them, even if they seem superficial. For instance, Musicat should easily notice and pay attention to such things as exactly repeated notes or patterns, very abrupt changes of

rhythm, extremely high or low notes, passages that form scales or arpeggios, tonics and dominants, and so forth. This strategy is especially crucial in a real-time domain because the time available for perception is limited. Unlike Copycat, which had ample time to consider many alternate ideas for each analogy problem, Musicat must rapidly and accurately discover the most important things. In so doing, the program can, paradoxically, listen in a more sensitive way than it would otherwise, because by very rapidly perceiving some things thanks to knowing how to take advantage of surface-level cues, it frees up more time for deeper, subtler, and more meaningful analogy-making. Douglas Hofstadter refers to this strategy as "deep superficiality".

Don't Downplay Top-down Pressures

A common issue we encountered during Musicat's development is that the program would get hung up on details of low-level perception, flailing about in making small-scale structures such as groups of just a few notes or a few measures. However, top-down pressure can greatly help to organize and prioritize lower-level perceptual processes. Many of the biggest advances in the quality of Musicat's listening performances came when I added new types of top-down pressure. For example, creating codelets that would take an existing analogy and look for a *parallel* analogy was particularly effective. The *high-level* search for the second analogy would be undertaken by additional codelets that would bias the program's *low-level* perception so that it would tend to look for the key groups and relationships in the most natural, fairly predictable places.

Rapid Feedback during Implementation is Essential

All runs of a stochastic program such as Musicat are different, and each run takes quite a while to complete when the visual display of the program's Workspace is enabled.

Therefore, understanding the effect of recent changes to a codelet or to parameter settings by simply running and rerunning the program requires a significant amount of time: I might need to watch the program running three, four, or even ten times to understand the effect of such changes. Luckily, I was able to streamline this process and to understand the effects of changes much more easily once I had designed a rapid-feedback tool (the Visual Studio add-in described in Chapter 8) that allowed me to see the effects of a change very quickly.

Experiment with Simple Examples

When I started developing Musicat, I used what I considered to be simple melodies, but much later I realized that I could make far simpler melodies. Running the program on extremely simple — indeed, nearly trivial — melodies often yielded surprising and key insights, while more complicated melodies often had a way of obscuring fundamental issues.

Have a Pressure-based Programming Mindset

Abhijit Mahabal coined the phrase "pressure-based programming" to describe his development of the Seqsee program. He and I spoke about this phrase only briefly, but I appreciate the idea. Programming a FARG model is quite unlike traditional programming because so much that happens is random. Instead of trying to think about the detailed sequence of events that might occur for desired structures to be built in the Workspace, it is helpful to think in terms of hundreds or thousands of codelets collectively exerting *pressure* on how structures will emerge over time in the Workspace.

During development, my use of a programming trick made it slightly easier for me to think in terms of pressures: I made it extremely fast and simple to add new codelets to the system using a technical feature of the C♯ language called a "class attribute". A class attribute is a simple marker that can be attached to a class to provide useful information that the

program can use at runtime. I defined a "CodeletAttribute" that the Coderack could detect when it needed to populate itself with more codelets. Then, any time I wanted to create a new type of codelet, I simply wrote a new class (deriving from the base "Codelet" class), and attached the CodeletAttribute to the class, along with a default urgency level. Then, simply recompiling the library of codelet classes would make copies of this new codelet appear in the Coderack.

Keep Melody and Rhythm Separate

On several different occasions in the past several years, I have come across one particular way of simplifying musical problems: separating the two primary elements of music — melody and rhythm — from each other. RhythmCat, for example, totally ignored pitch and focused just on rhythm. In the Seek Well domain, on the other hand, all notes have the same duration but vary in pitch. Melody and rhythm can interact in complex ways, but temporarily treating them as independent components can be surprisingly effective.

Architectural Contributions

Musicat is based on the FARG architecture, but it contributes several ideas that are novel with respect to other FARG programs. A short list of the most significant ideas follows.

Temporal Domain

Time is central to Musicat's domain. Of course, in other FARG programs, time passes as the program runs, but Musicat is unique in that input is given to the program in real time. Seek Whence and Seqsee have a somewhat temporal nature because they process number sequences, and the numbers arrive one at a time, but there can be an arbitrarily long gap between one number and the next, so time is not nearly as critical a pressure. Musicat

works solely in the music domain, but some of its architecture and some aspects of its codelets might be applicable to other temporal domains.

Motivic Memory in a Slipnet

The motivic memory introduced in MusicatW can be seen as a rudimentary learning mechanism: when new motifs are heard, they are added to the Slipnet, which constitutes a long-term memory. (The Phaeaco program does something similar.) In MusicatW, new Slipnet nodes can be created by codelets, and these nodes can subsequently be used by the program even when running on different melodies. Motif notes in the Slipnet form a small network of relationships: if two motifs share a rhythmic pattern or a pitch pattern, then activation can spread from one motif to the other. Motivic memory, while not explored deeply in Musicat, may prove to be useful for managing constraints of working memory and real-time listening.

Dynamic Group Scoring

In Copycat, strength values are computed for each group using a weighted sum of scores determined by various features of the group. In Musicat, there are factors contributing to the strength of a group that are created dynamically based on context, so there is no finite list of features whose scores can be included in such a sum. Even if there were such a list, I find the idea that *all* features of a group could be included to be cognitively implausible. Rather, only a *subset* comprised of features noticed by the program (called Group Reasons if they contribute a positive value, and Group Penalties for negative values) is included in the sum. A "squashing function" is used to ensure that the strength cannot exceed the maximum value of 100. Thus, the strength of a group is determined not by a preordained and fixed set of group features, but rather by aspects of the group that have become salient to the program.

In my experience, scoring of perceptual structures is one of the most critical components of designing a FARG model, and it definitely deserves more attention in the design of future models.

Instant Feedback during Development

The "add-in" that I developed for my programming environment, described in Chapter 8, is not part of the FARG architecture *per se*, but I found it to be a very helpful tool. Programming FARG models is surprisingly different from typical software engineering, so new tools such as this one are essential for handling the unusual programming challenges that arise.

Contributions to the Modeling of Music Cognition

Musicat is based on many things: a collection of ideas from many different people in the field of music cognition; the FARG architecture and projects by my FARG predecessors; my own ideas and intuitions about music; and countless conversations with Douglas Hofstadter about music, analogies, and the design and the behavior of the program. The long process of designing and implementing the program and figuring out how to model music-listening in a FARG framework has resulted in numerous personal insights. Because my project builds on so much prior knowledge, and because it is still quite exploratory in nature, it may seem conceited or premature for me to include a section entitled "contributions to the modeling of music cognition". My intent here is not to overstate my own contribution, but rather to pass along the most valuable ideas that have come up in the course of this work, in the hope that they will inspire further research. An analogy may be helpful: one of the activities I engaged in while designing Musicat was to create codelets that would constitute pressures for the system to hear music in a certain way that seemed reasonable to me;

similarly, the list of contributions in this section can be viewed as slight pressures for other researchers to focus on certain ways of modeling music cognition that seem important to me.

Listening as Performance

The most common question about Musicat I have heard from people who first hear about the program is simply "But what does it *do*? What is the output from the program?" The simplest answer — "It listens" — is typically unsatisfying to such people, but if I state that Musicat actively creates a "listening performance", they generally seem surprised but interested. The idea that listening is an active process is, of course, not new (the idea that music relies on the three elements of composer, performer, and listener is well known), but the particular perspective of "listening as performance" might be unique in the field of music-cognition modeling. I consider "listening performances" to involve creative work and the active creation of highly complex mental structures by the listener.

Importance of Rhythm

Naturally, rhythm is important, but I was surprised at how much of an improvement I saw in Musicat after we modified it to focus its attention much more on rhythm than on pitch. Rhythm seems to be more primal in human perception of melodies, and it makes sense for a listening model to reflect this. Perhaps my surprise just reflects my own bias: I am personally more interested in melody and harmony than in rhythm. I encourage others who share this bias to be sure to give adequate attention to rhythm.

A Computer Implementation of "Perspects"

Adam Ockelford's notion of musical perspects seems to me to be quite amenable to inclusion in a computer model, especially one that creates analogies between musical

structures. Ockelford himself has done some music research that makes use of perspects in a quantitative manner (Ockelford, 2008), but in that work in they are restricted to the domain of rhythm, even though the perspect concept applies to many musical parameters. In MusicatW, perspects were important components of the model: codelets could perceive various aspects of notes and record these perceptions (perspects) in the Workspace. In addition, I generalized the notion of perspects to apply not only to individual notes but also to groups of notes.

Elaboration of the Intricacies of Listening to Simple Melodies

In my experience, research that models music-listening tends to focus on one of two different points along a spectrum of musical complexity. There is a body of work at the less-complex end of the spectrum that examines listening on a short time-scale that involves just a few notes, exemplified by Krumhansl and Kessler's experiments on pitch stability in a tonal context (Carol L. Krumhansl, 1990). At the other end of the spectrum, there is much research on listening in real-world (and larger-scale) contexts, such as work that studies listeners' emotional responses as they listen to an entire musical piece, such as a pop song or a classical work than may be several minutes long. Of course there are numerous exceptions, such as David Temperley's computer models (Temperley, 2001), but my impression is that there is relatively little attention paid to melodies of the sort that I included earlier in this dissertation in the "Simple Melodies" chapter. Melodies such as "Twinkle, Twinkle, Little Star" and "Sur le pont d'Avignon" are deceptively simple: a full description of a human listening performance for melodies of this sort would be an extremely deep contribution to the understanding of both music and the human mind. Listening to simple melodies is far subtler and more complex than one might suspect, and I hope that Musicat's listening

performances, both in their accomplishments and in their failures, give at least an inkling of this complexity.

The Importance of Analogy

In Chapters 2 and 4 I mentioned several instances of theorists discussing the importance of musical parallelism — recall that parallelism was characterized in GTTM as an important avenue for future work, and Narmour also stressed its importance in his Implication–Realization theory. To the best of my knowledge, no prior computer models of musical listening make use of parallelism except in an extremely simple manner. In this work, however, parallelism is central: Musicat's analogies are a concrete implementation of the formerly only theoretical concept of parallelism. Musicat begins to fill the gap in theory explicitly mentioned by Lerdahl and Jackendoff, as I discussed in Chapter 4.

Moreover, I propose (admittedly without any justification aside from my own intuitions resulting from this work) that analogy is fundamental in music cognition, not just in that it helps to make sense of the structural organization of a piece of music, but also in that it contributes to musical semantics. To put it more clearly: grouping helps a listener understand music's formal structure, and analogy is important in generating a grouping structure, but perceiving analogy also contributes to a listener's sense of music having *meaning*.

Meta-grouping in Music-cognition Modeling

Computer models of music cognition often examine the problem of grouping. However, grouping is typically approached from the perspective of partitioning; in other words, group boundaries exist at only one level. Meta-groups or hierarchies are important to theorists, but appear less often than I would expect in computer models. For Musicat,

analogy is central, and all but the simplest analogies are formed between meta-groups, so meta-groups are essential. The set of Musicat's codelets that are involved with meta-group formation gives some ideas about how meta-groups might arise in listening, but much more work is needed. The field of music cognition would greatly benefit from an increased understanding of how listeners form meta-groups.

Musicat Introduces Structural Expectations

Melodic expectation has been a focus of much attention in the past decade or two. A great deal of work in the field involves expectation at the note level. That is, models typically generate expectations for the following note or several notes. In contrast, the latest version of Musicat generates expectations at the level of entire grouping structures and of inter-group analogies, rather than just at the level of individual notes. To be sure, both types of expectation are important, but these higher-level expectations translated more obviously into improvements of the program's listening performances.

Preliminary Steps towards a FARG-style Seek Well Program

In Chapter 2 I wrote about how the initial version of Musicat was based on Steve Larson's theory of musical forces and was intended as a successor to his Seek Well program. MusicatH attempted to automatically generate hierarchical embellishment structures, but the program didn't have much success. I believe that the main problem was the lack of analogy-making and of reliable group formation and meta-group formation in that program. Thus, although Musicat eventually diverged from the original goal of being an implementation of Larson's theory of musical forces using the FARG architecture, I now see some ideas in Musicat as necessary prerequisites to the automatic generation of embellishment structures. If

a future version of Musicat could successfully generate such structures, they could in turn be used to generate note-level expectations based on musical forces.

Future Work

Musicat is a work in progress, and there are many ways in which the program could be improved. The "Things Musicat Does Badly" section above suggests several obvious areas for additional work, as does careful observation of the program's listening performances, as given in Chapters 5–7. In this final section, I prioritize a few items that I think would be important next steps for Musicat. Following that, I speculate briefly about longer-term implications of this work.

Next Steps

My suggestions for future improvements are listed below in priority order; the first item has the highest priority, in my opinion.

1. Add more knowledge about pitch and tonality, especially focusing on types of tension associated with certain scale degrees, possibly in specific harmonic contexts.
2. Fix problems with the destruction and "resurrection" of large structures.
3. Remove the constraint that groups (and hence analogy components) must start and end at measure boundaries, while keeping a strong bias in favor of making group boundaries at these places.
4. Make the thicknesses of bar lines more responsive to strong groups that straddle strong bar lines. Grouping and bar-line thicknesses should influence each other more than they do at present.

5. Allow the program to "reset" its listening after an extremely thick bar line (*i.e.*, after reaching strong closure). Listening should be retrospective in some cases, but after a strong closure, listening seems to start "afresh". This should help the program listen better to long melodies.
6. Get the program to notice certain key types of superficial features more readily and to stick to simple perceptions based on such features unless and until they are strongly contradicted.
7. Restore the motivic memory component of the program, which was removed after MusicatW.
8. Generate note-level expectations to complement the structural expectations currently generated. Examine the program's real-time surprise and satisfaction based on how heard music denies or satisfies these expectations.

A Hypothesis for Future Testing

After I have improved Musicat in the ways specified in the previous section, and in particular after removing the strict constraint that groups must start and end at measure boundaries (item 3 above), I hope to compare Musicat's grouping performance with that of Temperley's CBMS model. More importantly, I intend this comparison to substantiate the claim made in Chapter 4 that analogy-making is essential for modeling in music cognition. With the present version of the program, however, Musicat cannot be tested on the same corpus of melodies used by Temperley in his experiments (Temperley, 2001) because of the constraint on where group boundaries occur: in the corpus, many group boundaries occur at places other than measure boundaries. (See Appendix C, however, for some preliminary results obtained from Musicat if we consider a modified version of the problem.)

The CBMS model of grouping is intended to work on simple monophonic folk melodies, just as Musicat is, but uses just a few simple rules. An important difference is that grouping in CBMS is not hierarchical; instead it simply predicts phrase boundaries without regard to the hierarchical structure considered by GTTM or Musicat. In addition, CBMS has a notion of parallelism, but this should not be confused with the parallelism discussed in GTTM. In CBMS, "parallelism" is the very specific and simple idea that successive musical phrases tend to start at the same metric position, which is quite similar to Musicat's strict restriction about group boundaries (when considered in conjunction with how Musicat internally shifts measures sideways to account for upbeats). CMBS prefers groupings which yield to this pressure of beginning successive phrases at the same metric position. However, the notion of parallelism invoked in GTTM, which operates naturally on hierarchical grouping structures instead of phrase boundaries, is the type of parallelism I am interested in with Musicat.

While much of the work of Musicat and other FARG models is that of modeling a cognitive system and examining whether the model has similar behaviors as humans, the difference in approach between Musicat and CBMS results in a straightforward, testable hypothesis involving analogy-making. CBMS is quite well-tuned to detecting phrase boundaries, and I hypothesize that if I disable Musicat's analogy-making mechanisms, it will perform worse than CBMS with respect to detecting phrase boundaries for the corpus of melodies used in Temperley's experiments. However, with analogy-making enabled, the predicted boundaries should be more accurate than those of CBMS.

Whither Musicat?
(Questions and Speculations about the Future)

The ideas in this section look very far ahead and are thus very speculative, but nonetheless I hope they are thought-provoking. What can we look forward to from Musicat in the future?

Could Musicat be Extended to Listen to Polyphonic Music?

Polyphonic music is much more complex than monophonic music, yet I believe that the most significant problems occur in listening to a melody. Fundamental structures such as groups and analogies are also central to polyphonic listening, although grouping will need to be more flexible to allow for multiple melody lines and their interactions. Fortunately, Musicat is already based on the parallel actions of a large number of codelets, and this parallelism will be easily adaptable to the task of listening to multiple lines of music in parallel. As the degree of polyphony increases, a human listener's perceptual resources available for each individual note are limited; this sort of limitation will arise naturally in the model as well, because a fixed number of codelets is allocated to be run for each unit of musical time. Musicat will need better mechanisms for focusing its attention when there are many choices about what to listen to, but the current architecture seems adaptable to polyphonic listening.

Will Musicat Ever Listen to Audio Recordings?

Musicat was designed to handle note-level input, not raw audio. However, since Musicat runs in real time, this offers hope for real-time audio listening. I can imagine a hybrid system in which Musicat "listens" to notes that are detected by a lower-level audio processing system based on standard engineering techniques such as the fast Fourier

transform, filter banks, Cepstral coefficients, and the like. However, cutting-edge techniques available at the time of this writing still make many errors in detecting notes. I believe that a listening program such as Musicat could provide real-time feedback to the audio-processing component of such a system, potentially improving the accuracy of note detection. Just as Musicat's own top-down pressures can guide its low-level perception, so Musicat as a whole might act as a source of top-down pressures that could guide the extremely low-level perception provided by an audio-processing algorithm. (After all, the human auditory system has neural feedback loops from higher-level processing back down to lower-level neural circuits, suggesting that higher-level cognition has a very direct way to focus low-level auditory perception.)

Will Musicat Ever Hear Music in the Same Way as a Human Does?

As much as I would like to imagine Musicat developing into a full-fledged listener, the honest answer is "no"! I have high hopes for Musicat, but one critical thing is missing: Musicat doesn't have emotions. I haven't discussed emotion in this dissertation because it is simply beyond the scope of this work, but emotion is critical to the human listening experience. Human listeners can be moved to tears or experience "chills" (or "*frisson*") as a result of listening to music. Until someone can model emotion, Musicat will remain unmoved by music, no matter how well it understands its structure and makes analogies.

Will Musicat Ever Become a Composer?

I am slightly more hopeful here (or perhaps I'm just a cockeyed optimist). As long as it doesn't listen in the same emotional way as a human does, Musicat will always lack something important. However, there are some interesting results in computer music composition, especially David Cope's program EMI (Cope & Hofstadter, 2001), which,

despite its shortcomings, succeeds in making stuff that sounds to many people like meaningful music. Programs such as EMI, however, lack the ability to listen to what they compose! If Musicat is one day able to create passable listening performances, it would have a huge advantage over any "deaf" computer composers, and it might well try its "hand" (and "ears") at writing music!