REAL-TIME GRAMMAR-BASED PARSING AND RESTRUCTURING OF MUSICAL STREAMS

Dale E. Parson and Ryan R. Panuski
Kutztown University of Pennsylvania
Department of Computer Science

ABSTRACT

Processing spoken word and instrumental musical recordings via looping has been an important transformational approach since the early days of recorded sound. In terms of formal language theory, looping comprises a restricted form of finite automaton execution, typically specified by some form of right-regular expression. Limitations on early recording and playback technology coincide with the limitations of regular expressions. More powerful formal language constructs have been used in some domains for years. The current work is an investigation into the use of parsers specified by context-free grammars using off-the-shelf parser generation tools for performance-time transformation of spoken word and instrumental phrases. Context-free grammars surpass regular expressions by supporting nested organization and reorganization of phrases and deeper hierarchical structures. This approach does not require speech recognition or complex signal analysis. Instead, a performer uses a control device or transient pauses to align markers to positions in an audio stream. These lexical markers guide parsing and grammatical reorganization of phrased audio for playback in real time. This approach yields useful results in the macro-temporal domain of spoken and instrumental phrase restructuring, as well as in the micro-temporal domain of granular phrase restructuring.

1. INTRODUCTION[1]

"Looping" and "sequencing" are among the most fundamental and heavily used mechanisms in computer and electronic music generation and transformation. Both terms refer to repetition of sonic events at various temporal granularities. "Looping" normally refers to repetition of recorded sound samples, with temporal resolution ranging from the sub-note time scale used in sample-based tone generation to repetition of recorded phrases and verses [10]. "Sequencing" refers to the encoding, storage and subsequent playback of control or performance data needed to generate a series of musical events. Both of these mechanisms are concrete manifestations of finite automata. A *finite automaton* is a hardware or software machine comprised of a finite number of *states*, along with guarded *transitions* that connect the states [1]. The *guard* on a transition specifies constraints that an incoming datum must satisfy in order for the automaton to transit from the source state to the destination state of the transition. The simplest guard is a literal value, such as the string "in a Western rut" in the automaton of Figure 1. This automaton accepts an initial string of "You folks are stuck in a Western rut" in transiting from an initial *start state* to an intermediate state labelled *looping*, followed by zero or more repetitions of the prepositional phrase "in a Western rut" in transiting from the looping state to itself. Finally, the arrival of the input clause "You're either this or else you're that" triggers transition from the looping state to the final, *terminal state*.

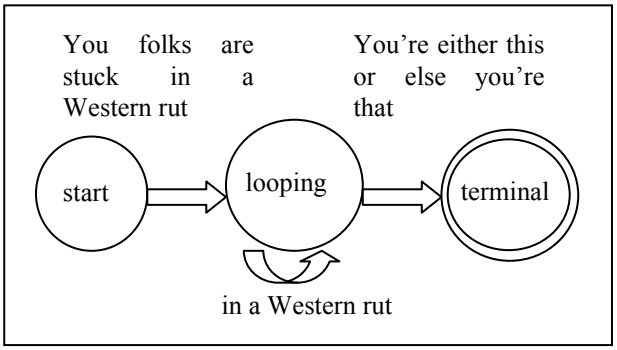

Figure 1. A rap as a finite automaton.

The automaton of Figure 1 is concisely described by the *regular expression*, "You folks are stuck (in a Western rut)+ You're either this or else you're that," where the "+" symbol signifies one or more repetitions of the parenthesized prepositional phrase. Regular expressions can include meta-symbols for specifying grouping, alternatives and repetition of sub-expressions.

A finite automaton can also specify an outgoing *action* on each transition, where an action is the production of output data occurring as part of the transition. An automaton representing a sequencer would produce control data as actions.

Finite automata provide a powerful abstraction for both compositional forms and performance mechanisms such as looping and sequencing, but they have a fundamental restriction. A finite automaton cannot generate or recognize hierarchical phrase structures that nest to arbitrary depth at composition or performance time. Fixed-depth nesting is possible with finite automata; the regular expression "You folks ((are stuck)+ in a Western rut)+" supports multiple repetition

[1] This project was made possible in part by an equipment grant from the Kutztown University Research Committee.

of the verb phrase "are stuck" as a sub-phrase within the larger repetitive clause. Such nested looping is common in rap, in part because it assists alignment of phrase boundaries with metric boundaries by making the former elastic at performance time. What is missing, though, is the ability to create recursive hierarchical structures found in language and many forms of musical composition.

In formal language theory the next most powerful abstraction after the regular expression is the *context-free grammar*, which supports hierarchically nested phrase structures [1]. The sentence diagram of Figure 2 shows one possible context-free parse of the sentence, "With the passing of the year comes another year to pass away," using a context-free sentence diagram. The non-arrowed lines delimit hierarchical composition of grammar *nonterminal symbols* including Sentence, Subject, Predicate, Noun Phrase (NP), Verb Phrase (VP), Prepositional Phrase (PP), Noun, Adjective (ADJ), Adverb (ADV), and Infinitive Phrase from lower-level nonterminals and *terminal symbols*. Each nonterminal comprises a sentence, clause, phrase or part of speech, acting as a subdivision of the more coarsely grained nonterminal division in which it occurs. Terminals are the bottommost, non-hierarchical structures. In this example terminals are words, but they could also be timed notes or other sonic events in a hierarchical musical structure. The dashed up-arrows show augmentation from modifiers to modified nonterminals. Context-free parsers typically use regular expressions to describe strings or other sequences of surface features that constitute terminal symbols. Context-free grammars and parsers are used extensively in translating programs into machine-executable forms.

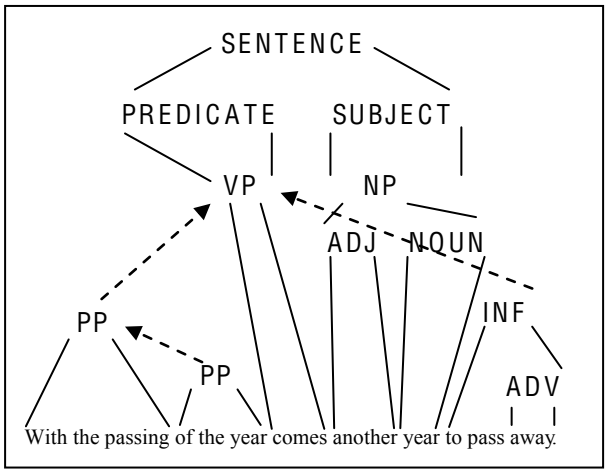

Figure 2. A context-free parse of a sentence.

A main goal of the current research project is support for performance-time restructuring of spoken word and instrumental phrase structures. For example, after recording the sentence of Figure 2, the performance system is capable of immediate restructuring and playing back an alternative sentence such as, "Another year comes to pass away with the passing of the year." Transformations include exchange, deletion and duplication of parse sub-trees anchored at nonterminals in the grammar. The intent is creation of real-time software instrument capabilities that extend and exceed standard looping-based capabilities. Preparation for performance includes construction and compilation of a grammar that describes sentences or musical passages to be performed, recorded, transformed and replayed, along with practice using the performance system.

2. RELATED WORK

Pierre Schaeffer was a pioneering investigator of the use of disks and tape for processing recorded sounds including looping in the early twentieth century [7,10]. Steve Reich made particularly effective early compositional use of tape loops [8]. Control sequencing has been used since the days of player pianos and similar mechanical and electro-mechanical instruments.

Application of context-free grammars and parsers to musical analysis and composition goes back to the 1970's. Roads and Wieneke summarize formal language theory and survey projects in progress in 1979 [9], stating, "A representation for music should at least have the power to handle nested phrases and motives, constructions which are technically excluded from type 3 grammars," where type 3 grammars include regular expressions and finite automata. The authors thoroughly discuss the strengths of context-free grammars in representing musical structure.

Lerdahl and Jackendoff present a thorough application of formal language theory to representation of musical structure that includes practical parsing concerns [5]. Swain relates syntax to musical tension and release [12]. Barbar et. al. discuss syntax-directed translation of musical structures [3]. A Markov Process constitutes a form of finite state grammar that has been applied extensively to musical structure analysis and generation [2,9].

The emphasis in formal language-derived musical work has been on the use of grammars and parsers in musical structure representation, analysis and composition. Roads and Wieneke predicted, "Grammars may lead beyond unified composing and analysis models and toward intelligent musical devices." [9] However, a survey of available software and electronic hardware instruments finds the state of practice limited to the application of finite automata / looping mechanisms. The intent of the present project is integration of context-free spoken word and instrumental musical parsers and generators into a recording-based software instrument. The one-to-many mappings of a grammar to sentences outlined by that grammar makes improvisation possible, given sufficient preparation of a grammar and practice with its performance-time application.

3. PARSING AND RESTRUCTURING AUDIO STREAMS IN REAL TIME

3.1. Manual placement of lexical markers

Figure 3 is a recasting of Figure 2 showing the architecture of the grammar-based software instrument. The initial implementation uses a ChucK [4] process in the bottom of Figure 3 to capture and play back an audio stream, and to store markers into that stream. Figure 3's top half represents a Python process that accepts performer input from a graphical user interface (GUI) and sends control messages to the ChucK recorder. Python has several excellent parser generation tools [6] that make it useful as a front end.

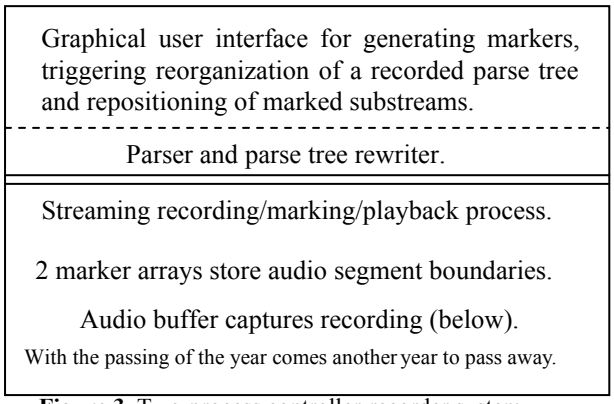

Figure 3. Two-process controller-recorder system.

The system begins recording when a user clicks a Start button in the Python GUI, sending a Start message to the recorder. During the recording phase, a user can click grammatical Mark buttons, sending a Mark message. The recorder maintains two arrays of markers that are indices into the audio buffer being recorded. There are two marker arrays so that later reorganization of the markers can support playback using one array without interference from the other array being reorganized. Recording stops with the arrival of a Stop message, optionally entering a playback loop.

In addition to sending Mark messages, the Python parser constructs a parse tree similar to Figure 2 whenever the user clicks a Mark button. There are Mark buttons for low-level nonterminal types of Figure 2 such as PP, ADJ, NOUN, etc. Instead of parsing the audio stream, the parser parses the *lexical type* (noun, etc.) of the GUI Mark button being pressed at the same time that it sends markers to the recorder process. This temporal association between arrival of player-triggered grammar events in the GUI and marker positions in the audio stream supports construction of a parse tree as in Figure 2 and its correlation to phrases in the audio stream. After completing the recording / marking phase, the performer uses additional GUI buttons to restructure the parse tree. Transformations include exchange, deletion and duplication of parse sub-trees anchored at nonterminals as previously mentioned. For example, triggering the phrase rearrangement of the parse tree of Figure 2 leading to the sentence, "Another year comes to pass away with the passing of the year," causes the Python process to send a new arrangement of existing markers to the ChucK process. The ChucK playback loop uses that new sequence of markers as soon as it is received and stored. The recorder never actually modifies the audio buffer, and it has no knowledge of grammar processing. Its playback loop simply plays audio sub-sequences whose order is determined by the markers. Some sub-sequences may be skipped or repeated. The timing and order of the markers is fully under the performer's manual control.

3.2. Linear-time detection of phrase-boundary lulls

The system as described and implemented to this point has been used in some performances, but it has a serious pitfall. There is a tendency for a spoken word performer to click grammatical Mark buttons "on the beat," i.e., in the region of audio attack transitions. Instrumental performances with Mark buttons triggered by MIDI foot switches suffer from the same problem. Subsequent rearrangement and playback exhibit clicks and pops caused by initiation or termination of marker-delimited sub-sequences in the middle of audio transients. Performers can practice leaving intentional lulls at phrase boundaries and learning to click Mark buttons at those boundaries. This approach coupled with sub-sequence cross-fading is an improvement, but performance becomes stilted, performer timing becomes irregular, and the problems of markers at transient attack boundaries are never fully eliminated.

Our solution has been to introduce a minimum of signal processing into the recording system. Spoken word performers do tend to use transient lulls "before the beat," i.e., between phrases, as seen in Figure 4. We now have the recording system perform ongoing lull detection by sampling the incoming audio amplitude and using a rolling average to maintain its own markers for periods of relative quiet. Threshold parameters for maximum amplitude and minimum duration of lull periods are under performer control. Each lull period gets two markers, one for its start and another for its end. The temporal center point of such a period is a candidate for a phrase boundary. The recorder correlates the arrival of marker events from the Python control process back to the most recent center of a lull period and uses that time point as the actual playback marker. Maintaining a rolling average of absolute amplitude is a linear-time, low overhead effort. This modification of the original approach has mostly eliminated clicks and pops in reorganized playback.

We have also experimented with eliminating GUI Mark buttons and using the center points of lull periods alone as phrase boundary markers. This approach loses type of speech information for constructing deep parse trees. It is possible to use the relative length and depth of lull periods as "parts of speech." Long, deep lulls mark major phrase boundaries, while short, shallow lulls mark minor phrases. The limited categories of phrase types reduce the potential complexity of grammars and parse trees. This approach does show promise for instrumental

music, as well as stochastic phrase rearrangement, where grammar-based rearrangement is replaced by probability-tagged restructuring rules for the limited phrase types. The restructuring rules are in fact grammar productions, but they apply probabilistically rather than deterministically to the audio sub-sequences. A reduction in available phrase types is compensated by an increase in available transformations applied stochastically, at the cost of predictability as with any stochastic process.

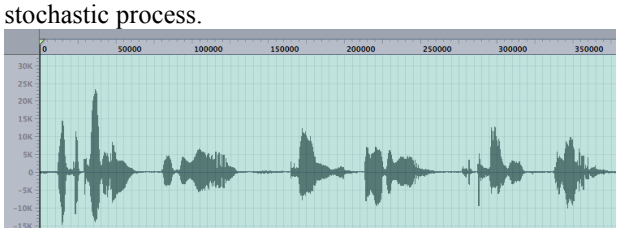

Figure 4. Sample of "With the passing – of the year – comes – another year – to pass – away" showing intentional lulls in the performance.

The software instrument is to the point where the major effort required is that of constructing a good spoken word or instrumental piece as a template for a grammar, then constructing the grammar, and then learning to use that grammar in improvisation. The work load is shifting from instrument design to composition and perfomance effort, as it should. Using parser generators is a task that some spoken word performers may not mind; we do not anticipate that most instrumental performers will be willing to program parser generators. Providing graphical construction of grammars and performance-time display of graphical parse trees similar to Figure 2, with performer controls for interactively restructuring those graphical trees, might make grammars more acceptable to composers and performers. We have not yet started this GUI work.

4. FUTURE WORK

One surprise in investigating this approach is the discovery that, with sufficiently small thresholds for lull period marking, the non-manual marking approach (stochastic or deterministic) allows restructuring using phrases of sound grains. The field of *microsound* [11] examines temporal transitions from rhythm into pitch and from pitch into timbre. We have used primitive granular restructuring in one public web-based spoken word performance, and we look to refine this approach.

The advent of multicore laptop computers makes possible the use of more complex performance-time phrase delineation based on sonic lexical properties, such as spectral properties within a temporal region of sound. The first key software engineering observation is the fact that marker extraction, copying and rearrangement do not alter the recorded audio stream. Processor cores can manipulate marker arrays concurrently without costly synchronization. A second key observation is the fact that, in a live performance, the audio buffer is captured as the audio is performed. Low-overhead, linear-time marker extraction such as lull extraction can occur during performance recording. Higher overhead linear-time marker extraction can use separate cores to extract markers during recording. Non-linear-time spectral and granular approaches can run concurrently on separate cores, locating markers for later application of parsing and parse tree restructuring based on sonic properties of audio sub-sequences. Phrase marker extraction based on sonic properties, and graphical interfaces for grammar capture and parse tree reorganization, are two frontiers for future exploration.

5. REFERENCES

[1] Aho, A, Lam, Sethi and Ullman, *Compilers – Principles, Techniques & Tools*, Second Edition, Addison-Wesley, 2007.

[2] Ames, C., "The Markov Process as a Compositional Model: A Survey and Tutorial," *Leonardo* 22(2) (1989), pp. 175-187.

[3] Barbar, K., Desainte-Catherine and Miniussi, "The Semantics of Musical Hierarchies," *Computer Music Journal* 17(4) (Winter, 1993), pp. 30-37.

[4] ChucK audio programming language, January, 2011, http://chuck.cs.princeton.edu/.

[5] Lerdahl, F. And R. Jackendoff, *A Generative Theory of Tonal Music*, MIT Press, 1982.

[6] PLY (Python Lex-Yacc), January, 2011, http://www.dabeaz.com/ply/.

[7] Prendergast, Mark, *The Ambient Century*, Bloomsbury Publishing, New York, 2003.

[8] Reich, Steve, *Writings on Music 1965 – 2000*, Oxford University Press, 2002.

[9] Roads, C. And P. Wieneke, "Grammars as Representations for Music," *Computer Music Journal* 3(1) (March, 1979), pp 48-55.

[10] Roads, C., *The Computer Music Tutorial*, MIT Press, 1996.

[11] Roads, C., *Microsound*, MIT Press, 2001.

[12] Swain, J., "The Concept of Musical Syntax," *The Musical Quarterly* 79(2) (Summer, 1995), pp. 281-308.