Music as Language

Putting Probabilistic Temporal Graph Grammars to Good Use

Orestis Melkonian

August 23, 2019

Utrecht University, The Netherlands

Introduction

Motivation

- The aim is not to generate a complete music piece
  - Rather an aid in the compositional process
- Do this in a concise way, which is also expressive enough
  - To that end, we use formal grammars

Probabilistic Temporal Graph Grammars

Grammar-Based Automated Music Composition in Haskell

Donya Quick
Yale University
donya.quick@yale.edu

Paul Hudak
Yale University
paul.hudak@yale.edu

Abstract

Few algorithms for automated music composition are able to address the combination of harmonic structure, metrical structure, and repetition in a generalized way. Markov chains and neural nets struggle to address repetition of a musical phrase, and generative grammars generally do not handle temporal aspects of music in a way that retains a coherent metrical structure (nor do they handle repetition). To address these limitations, we present a new class of generative grammars called *Probabilistic Temporal Graph Grammars*, or PTGG's, that handle all of these features in music while allowing an elegant and concise implementation in Haskell. Being probabilistic allows one to express desired outcomes in a probabilistic manner; being temporal allows one to express metrical structure; and being a graph grammar allows one to express repetition of phrases through the sharing of nodes in the graph. A key aspect of our approach that enables handling of harmonic and metrical structure in addition to repetition is the use of rules that are parameterized by duration, and thus are actually *functions*. As part of our implementation, we also make use of a music-theoretic concept called *chord spaces*.

Rather than music analysis, however, we are interested in *automated music composition*. One way to approach this is to use grammars *generatively*—that is, to generate sentences from the start symbol. The problem is, with a conventional grammar (such as a context-free grammar, or CFG) the result is usually nonsensical—for example, "The dog wrote the house," or in the case of music, something that just doesn't sound right.

More specifically, conventional grammars intended for automated music composition have the following limitations:

1. They are unable to capture the *sharing* of identical phrases, such as in a song form AABA, where the A sections are intended to be identical (or nearly identical) to one other.
2. They do not take *probabilities* into account. Music analysis has shown that certain productions are more common than others—indeed specific genres of music (say, Bach chorales) have specific distributions of musical characteristics [23].
3. They do not capture *temporal* aspects of music. For example, a production rule stating that a I chord can be replaced with V-I does not capture the fact that the total duration of the

Probabilistic Temporal Graph Grammars

- **Probabilistic**: Rules are assigned prob. weights
- **Temporal**: Rules are time-parametric
- **Graph**: *Let* construct allows sharing/repetition

Euterpea

- Euterpea will be our music development vehicle
- Define some extra things that are not in the library
    1. intervals, chords, scales
    2. transposition on several music elements
    3. random actions in *MonadRandom*

$$
\begin{aligned}
&\textbf{data}\ \mathit{Interval}\\
&\quad = P1 \mid Mi2 \mid M2 \mid Mi3 \mid M3 \mid P4 \mid A4 \mid P5\\
&\quad \mid Mi6 \mid M6 \mid Mi7 \mid M7 \mid P8 \mid Mi9 \mid \ldots \mid P15\\
&\quad \textbf{deriving}\ (Eq, Enum)
\end{aligned}
$$

$$
\begin{aligned}
&\textbf{type}\ \mathit{ChordType} = [\mathit{Interval}] && \text{-- } \equiv \mathit{ScaleType}\\
&\textbf{type}\ \mathit{SemiChord} = [\mathit{PitchClass}] && \text{-- } \equiv \mathit{SemiScale}\\
&\textbf{type}\ \mathit{Chord} = [\mathit{Pitch}] && \text{-- } \equiv \mathit{Scale}
\end{aligned}
$$

$$
\begin{aligned}
&(\Vdash) :: \mathit{PitchClass} \rightarrow \mathit{ChordType} \rightarrow \mathit{SemiChord}\\
&(\Vdash) = \ldots
\end{aligned}
$$

A fairly extensive set of scales/chords

$$
\begin{aligned}
&\textit{-- Chord types.}\\
&maj \ \ = [P1, M3, P5]\\
&m7b5 = [P1, Mi3, A4, Mi7]\\
&\vdots\\
&allChords = [maj, \ldots] :: [ChordType]\\
\\
&\textit{-- Scale types.}\\
&ionian = [P1, M2, M3, P4, P5, M6, M7]\\
&major \ = ionian\\
&lydian = mode\ 4\ ionian\\
&\vdots\\
&allScales = [ionian, \ldots] :: [ScaleType]
\end{aligned}
$$

Transposition

$$
\begin{array}{l}
\textbf{class}\ \mathit{Transposable}\ a\ \textbf{where} \\
\quad (\uparrow), (\downarrow) :: a \rightarrow \mathit{Interval} \rightarrow a \\
\\
\textbf{instance}\ \mathit{Transposable}\ \mathit{PitchClass}\ \textbf{where}\ \ldots \\
\textbf{instance}\ \mathit{Transposable}\ \mathit{Chord}\ \textbf{where}\ \ldots
\end{array}
$$

$$
\begin{aligned}
&\mathit{equally} :: [a] \to [(\mathit{Double}, a)] \\
&\mathit{equally} = \mathit{zip}\ (\mathit{repeat}\ 1.0)
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{choose} :: \mathit{MonadRandom}\ m \Rightarrow [(\mathit{Double}, a)] \to m\ a \\
&\mathit{choose}\ \mathit{xs} = \mathbf{do} \\
&\quad i \leftarrow \mathit{getIndex} \langle\$\rangle\ \mathit{getRandomR}\ (0, \mathit{sum}\ (\mathit{fst} \langle\$\rangle\ \mathit{xs})) \\
&\quad \mathit{return}\ (\mathit{xs}\ !!\ i)
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{chooseBy} :: \mathit{MonadRandom}\ m \Rightarrow (a \to \mathit{Double}) \to [a] \to m\ a \\
&\mathit{chooseBy} = \mathit{choose} \circ \mathit{fmap}\ (\lambda a \to (f\ a, a))
\end{aligned}
$$

PTGG Extensions

- A grammars consists of an initial symbol and several rewrite rules:

$$\mathbf{data}\ \mathit{Grammar}\ \mathit{meta}\ a$$
$$= a\ \mid :\ [\mathit{Rule}\ \mathit{meta}\ a]$$

- A rule replaces an atomic symbol with a grammar term:

$$\mathbf{data}\ \mathit{Rule}\ \mathit{meta}\ a$$
$$= (a, \mathit{Double}, \mathit{Dur} \to \mathit{Bool}) \rightarrowtail (\mathit{Dur} \to \mathit{Term}\ \mathit{meta}\ a)$$

- Terms are (sequences of) atomic symbols, possibly wrapped with metadata or repeated with *Let*:

$$
\begin{aligned}
&\mathbf{data}\ \mathit{Term}\ \mathit{meta}\ a \\
&\quad = a : \mathit{Dur} \\
&\quad \mid \mathit{Term}\ \mathit{meta}\ a \otimes \mathit{Term}\ \mathit{meta}\ a \\
&\quad \mid \mathit{meta} \triangleright \mathit{Term}\ \mathit{meta}\ a \\
&\quad \mid \mathit{Let}\ (\mathit{Term}\ \mathit{meta}\ a)\ (\mathit{Term}\ \mathit{meta}\ a \to \mathit{Term}\ \mathit{meta}\ a)
\end{aligned}
$$

- The user has to provide a way to expand metadata:

$$\textbf{class}\ \mathit{Expand}\ \mathit{input}\ a\ \mathit{meta}\ b\ \textbf{where}$$
$$\quad \mathit{expand} :: \mathit{input} \rightarrow \mathit{Term}\ \mathit{meta}\ a \rightarrow \mathit{IO}\ (\mathit{Term}\ ()\ b)$$

- The user has to provide a way to interpret abstract musical structures:

```
class ToMusic1 c ⇒ Interpret input b c where
  interpret :: input → Music b → IO (Music c)
```

- All in all, the user makes sure the following constraints are satisfied:

```haskell
type Grammarly input meta a b c =
  (Eq a, Eq meta, ToMusic1 c
  , Expand input a meta b
  , Interpret input b c)
```

- Given a desired total duration, a well-formed grammar and the required input for expansion and interpretation, rewrite up to fixpoint:

$$
\begin{aligned}
&\mathit{runGrammar} :: \mathit{Grammarly}\ \mathit{input}\ \mathit{meta}\ a\ b\ c \\
&\qquad \Rightarrow \mathit{Grammar}\ \mathit{meta}\ a \rightarrow \mathit{Dur} \rightarrow \mathit{input} \\
&\qquad \rightarrow \mathit{IO}\ (\mathit{Music}\ b, \mathit{Music}1) \\
&\mathit{runGrammar}\ (\mathit{init}\ |{:}\ \mathit{rs})\ \mathit{t0}\ \mathit{input} = \mathbf{do} \\
&\quad \mathit{rewritten} \leftarrow \mathit{fixpointM}\ \mathit{rewrite}\ (\mathit{init} : \mathit{t0}) \\
&\quad \mathit{expanded} \leftarrow \mathit{expand}\ \mathit{input}\ (\mathit{unlet}\ \mathit{rewritten}) \\
&\quad \mathbf{let}\ \mathit{abstract} = \mathit{toMusic}\ \mathit{expanded} \\
&\quad \mathit{concrete} \leftarrow \mathit{toMusic1} \langle\$\rangle\ \mathit{interpret}\ \mathit{input}\ \mathit{abstract} \\
&\quad \mathit{return}\ (\mathit{abstract}, \mathit{concrete})
\end{aligned}
$$

Harmony

Musicology Source: [Rohrmeier, 2011]

Towards a generative syntax of tonal harmony

Martin Rohrmeier*

Centre for Music & Science, Faculty of Music, University of Cambridge, 11 West Road, Cambridge CB3 9DP, UK

(Received 3 November 2010; final version received 16 March 2011)

This paper aims to propose a hierarchical, generative account of diatonic harmonic progressions and suggest a set of phrase-structure grammar rules. It argues that the structure of harmonic progressions exceeds the simplicity of the Markovian transition tables and proposes a set of rules to account for harmonic progressions with respect to key structure, functional and scale degree features as well as modulations. Harmonic structure is argued to be at least one subsystem in which Western tonal music exhibits recursion and hierarchical organization that may provide a link to overarching linguistic generative grammar on a structural and potentially cognitive level.

Grammar Symbols: Harmonic Degrees

```haskell
data Degree
   = I | II  | III  | IV | V | VI | VII -- terminals
   | P | TR | DR | SR | T | D | S    -- non-terminals
  deriving (Eq, Enum)
```

```
harmony :: Grammar Interval Degree
harmony = P |:
  [ -- Phrase level
    (P, 1, always)    ↣ λt → fillBars (t, 4 ∗ wn) TR
    -- Functional level: Expansion
  , (TR, 1, (> wn)) ↣ λt → TR : t/2 ⊗ DR : t/2
  , (TR, 1, always) ↣ λt → DR : t/2 ⊗ T  : t/2
  , (DR, 1, always) ↣ λt → SR : t/2 ⊗ D  : t/2 ] ++
  [ (x, 1, (> wn)) ↣ λt → Let (x : t/2) (λy → y ⊗ y)
  | x ← [TR, SR, DR] ] ++
  [ (TR, 1, always) ↣ (T :)
  , (DR, 1, always) ↣ (D :)
  , (SR, 1, always) ↣ (S :)
```

```
  -- Functional level: Modulation
, (D, 1, (⩾ qn))   ↣ λt → P5 ▷ D : t
, (S, 1, (⩾ qn))   ↣ λt → P4 ▷ S : t ] ++
  -- Scale-degree level: Secondary dominants
[ (x, 1, (⩾ hn)) ↣ λt → Let  (x : t/2) (λy → (P5 ▷ y) ⊗  y)
| x ← [T, D, S] ] ++
[ -- Scale-degree level: Functional-Scale interface
  (T, 1, (⩾ wn))   ↣ λt → I : t/2 ⊗ IV : t/4 ⊗ I : t/4
, (T, 1, always)   ↣ (I   :)
, (S, 1, always)   ↣ (IV  :)
, (D, 1, always)   ↣ (V   :)
, (D, 1, always)   ↣ (VI  :) ]
```

```
data HarmonyConfig = HarmonyConfig
  { basePc    :: PitchClass
  , baseScale :: ScaleType
  , chords    :: [(Double, ChordType)] }

instance Expand HarmonyConfig Degree Interval SemiChord where
  expand :: HarmonyConfig → Term Interval Degree
           → IO (Term () SemiChord)
  expand cfg (i ▷ e)      = expand (cfg {basePc = basePc cfg ↑ i}) e
  ...
  expand cfg (degree : t) = (:t) ⟨$⟩ choose (filterChords cfg degree)
```

Interpretation: From Abstract to Semi Chords

```
instance Interpret HarmonyConfig SemiChord Chord where
  interpret :: HarmonyConfig → Music SemiChord → IO (Music Chord)
  interpret cfg = fold1 f
    where f m (sc, d) = do
      ch ← chooseBy (chordDistance m) (inversions sc)
      return (m ⊗ ch)
```

Rewrite Example

$$P : 8 * wn$$

Rewrite Example

$$P : 8 * wn$$

$$\downarrow$$

$$TR : 4 * wn \ \otimes \ TR : 4 * wn$$

Rewrite Example

$$P : 8 * wn$$

$$\downarrow$$

$$TR : 4 * wn \;\otimes\; TR : 4 * wn$$

$$\downarrow$$

$$TR : 2 * wn \;\otimes\; DR : 2 * wn \;\otimes\; TR : 2 * wn$$

Rewrite Example

$$P : 8 * wn$$

$$\downarrow$$

$$TR : 4 * wn \;\otimes\; TR : 4 * wn$$

$$\downarrow$$

$$TR : 2 * wn \;\otimes\; DR : 2 * wn \;\otimes\; TR : 2 * wn$$

$$\vdots$$

$$\downarrow$$

$$I : wn \;\otimes\; IV : hn \; I : hn \;\otimes\; (P5 \triangleright VI : wn) \;\otimes\; V : wn \;\otimes\; I : 2 * wn$$

Melody

Musicology Source: [Keller, 2007]

A Grammatical Approach to Automatic Improvisation

Robert M. Keller, David R. Morrison
Harvey Mudd College
Claremont, California, USA
keller@hmc.edu, dmorrison@hmc.edu

***Abstract*— We describe an approach to the automatic generation of convincing jazz melodies using probabilistic grammars. Uses of this approach include a software tool for assisting a soloist in the creation of a jazz solo over chord progressions. The method also shows promise as a means of automatically improvising complete solos in real-time. Our approach has been implemented and demonstrated in a free software tool.**

***Keywords*— jazz, improvisation, educational software, probabilistic context-free grammar, melody generator.**

context-free grammar, as it occurs in computer science and linguistics. Then we show how a grammar can be used to generate plausible rhythmic and melodic sequences for jazz. Playable examples will be presented that show the effectiveness of the approach. Our approach has been implemented and has been tested in the past year as part of a broader educational software tool, which proved beneficial in a jazz improvisation course taught by the first author and has also been used by various third parties.

```
data M
    = HT | CT | L | AT | ST | R -- terminals
    |  P  | Q  | V | N          -- non-terminals
  deriving Eq
```

$(\multimap) :: (a, Double, Dur \to Bool) \to Term\ meta\ a \to Rule\ meta\ a$

$a \multimap b = a \rightarrowtail const\ b$

$$
\begin{array}{l}
melody :: Grammar\ ()\ M \\
melody = P\ |\!: \\
\quad [\ \text{-- Rhythmic Structure: Expand } P \text{ to } Q \\
\quad\ \ (P,\ 1,\ (\equiv qn))\ \rightarrowtail (Q\ :) \\
\quad ,(P,\ 1,\ (\equiv hn))\ \rightarrowtail (Q\ :) \\
\quad ,(P,\ 1,\ (\equiv hn^{\cdot}))\ \multimap Q : hn \otimes Q : qn \\
\quad ,(P, 25, (> hn^{\cdot}))\ \rightarrowtail \lambda t \to Q : hn \otimes P : (t - hn) \\
\quad ,(P, 75, (> wn))\ \rightarrowtail \lambda t \to Q : wn \otimes P : (t - wn)
\end{array}
$$

-- Melodic Structure: Expand Q to V, V to N

$$
\begin{array}{llll}
, (Q, 52, & (\equiv wn)) & \rightarrow & Q : hn \otimes V : qn \otimes V : qn \\
, (Q, 47, & (\equiv wn)) & \rightarrow & V : qn \otimes Q : hn \otimes V : qn \\
, (Q, 1, & (\equiv wn)) & \rightarrow & V : en \otimes N : qn \otimes N : qn \otimes N : qn \otimes V : en \\
, (Q, 60, & (\equiv hn)) & \rightarrow & Let\ (V : qn)\ (\lambda x \rightarrow x \otimes x) \\
, (Q, 16, & (\equiv hn)) & \rightarrow & HT : qn^{\cdot} \otimes N : en \\
, (Q, 12, & (\equiv hn)) & \rightarrow & V : en \otimes N : qn \otimes V : en \\
, (Q, 6, & (\equiv hn)) & \rightarrow & N : hn \\
, (Q, 6, & (\equiv hn)) & \rightarrow & HT : qn^{3} \otimes HT : qn^{3} \otimes HT : qn^{3} \\
, (Q, 1, & (\equiv qn)) & \rightarrow & CT : qn \\
, (V, 1, & (\equiv wn)) & \rightarrow & Let\ (V : qn)\ (\lambda x \rightarrow x \otimes x \otimes x \otimes x) \\
, (V, 72, & (\equiv qn)) & \rightarrow & Let\ (V : en)\ (\lambda x \rightarrow x \otimes x) \\
, (V, 22, & (\equiv qn)) & \rightarrow & N : qn \\
, (V, 5, & (\equiv qn)) & \rightarrow & Let\ (HT : en^{3})\ (\lambda x \rightarrow x \otimes x \otimes x) \\
, (V, 1, & (\equiv qn)) & \rightarrow & Let\ (HT : en^{3})\ (\lambda x \rightarrow x \otimes x \otimes AT : en^{3})
\end{array}
$$

-- Melodic Structure: Expand N to terminals

$$
\begin{aligned}
&, (N, 1, (\equiv hn)) \rightarrow CT : hn \\
&, (N, 50, (\equiv qn)) \rightarrow CT : qn \\
&, (N, 50, (\equiv qn)) \rightarrow ST : qn \\
&, (N, 45, (\equiv qn)) \rightarrow R : qn \\
&, (N, 20, (\equiv qn)) \rightarrow L : qn \\
&, (N, 1, (\equiv qn)) \rightarrow AT : qn \\
&, (N, 40, (\equiv en)) \rightarrow CT : en \\
&, (N, 40, (\equiv en)) \rightarrow ST : en \\
&, (N, 20, (\equiv en)) \rightarrow L : en \\
&, (N, 20, (\equiv en)) \rightarrow R : en \\
&, (N, 1, (\equiv en)) \rightarrow AT : en \;]
\end{aligned}
$$

$$
\begin{array}{l}
\textbf{data}\ \mathit{MelodyConfig} = \mathit{MelodyConfig} \\
\quad \{\, \mathit{scales} \;\; :: [(\mathit{Double}, \mathit{ScaleType})] \\
\quad ,\, \mathit{octaves} :: [(\mathit{Double}, \mathit{Octave})]\ \}
\end{array}
$$

$$
\begin{array}{l}
\textbf{instance}\ \mathit{Interpret}\ (\mathit{MelodyConfig}, \mathit{Music}\ \mathit{SemiChord})\ \mathit{M}\ \mathit{Pitch}\ \textbf{where} \\
\quad \mathit{interpret}\ (\mathit{cfg}, \mathit{chs})\ \mathit{symbols} = \mathit{mapM}\ \mathit{interpretSymbol} \circ \mathit{synchronize}\ \mathit{chs} \\
\qquad \textbf{where} \\
\qquad\quad \mathit{synchronize} :: \mathit{Music}\ a \to \mathit{Music}\ b \to \mathit{Music}\ (a, b) \\
\qquad\quad \mathit{interpretSymbol} :: (\mathit{SemiChord}, \mathit{M}) \to \mathit{IO}\ \mathit{Pitch} \\
\qquad\quad \mathit{interpretSymbol}\ (\mathit{ch}, s) = \textbf{case}\ s\ \textbf{of} \\
\qquad\qquad \mathit{CT} \to \mathit{choose}\ \mathit{ch} \\
\qquad\qquad \mathit{AT} \to \mathit{choose}\ \$\ (\mathit{ch} \downarrow \mathit{Mi2}) +\!\!+ (\mathit{ch} \uparrow \mathit{Mi2}) \\
\qquad\qquad \mathit{ST} \to \mathit{choose}\ \$\ \mathit{filter}\ (\mathit{match}\ \mathit{chord})\ (\mathit{scales}\ \mathit{cfg}) \\
\qquad\qquad \ldots
\end{array}
$$

Rhythm

Musicology Source: [Bel, 1992]

Modelling improvisatory and compositional processes

Abstract

An application of formal languages to the representation of musical processes is introduced. Initial interest was the structure of improvisation in North Indian *tabla* drum music, for which experiments have been conducted in the field as far back as 1983 with an expert system called the Bol Processor, BP1. The computer was used to generate and analyze drumming patterns represented as strings of onomatopeic syllables, bols, by manipulating formal grammars. Material was then submitted to musicians who assessed its accuracy and increasingly more elaborate and sophisticated rule bases emerged to represent the musical idiom.

Since several methodological pitfalls were encountered in transferring knowledge from musician to machine, a new device, named QAVAID, was designed with the capability of learning from a sample set of improvised variations supplied by a musician. A new version of Bol Processor, BP2, has been implemented in a MIDI studio environment to serve as a aid to rule-based composition in contemporary music. Extensions of the syntactic model, such as substitutions, metavariables, and remote contexts, are briefly introduced.

Keywords

Formal grammars, pattern languages, knowledge acquisition, cognitive anthropology, ethnomusicology.

Grammar Symbols: Rhythmic Syllables

```
data Syllable
    = -- terminals
      Tr | Kt  | Dhee | Tee | Dha | Ta
    | Ti | Ge | Ke    | Na  | Ra   | Noop
      -- non-terminals
    | S   | XI  | XD  | XJ  | XA  | XB  | XG  | XH | XC
    | XE  | XF  | TA7 | TC2 | TE1 | TF1 | TF4 | TD1
    | TB2 | TE4 | TC1 | TB3 | TA8 | TA3 | TB1 | TA1
  deriving Eq

(↠) :: a → [a] → Rule meta a
x ↠ xs = (x, 1, always) ⇾ fold1 ⊗ ((: en) ⟨$⟩ xs)
```

Grammar for Tabla Improvisation

$$
\begin{aligned}
&\mathit{rhythm} :: \mathit{Grammar}\ ()\ \mathit{Syllable}\\
&\mathit{rhythm} = S\ |{:}\\
&\quad [S \twoheadrightarrow [TE1, XI]\\
&\quad , XI \twoheadrightarrow [TA7, XD], XD \twoheadrightarrow [TA8]\\
&\quad , XI \twoheadrightarrow [TF1, XJ]\ \ , XJ \twoheadrightarrow [TC2, XA]\\
&\quad , XA \twoheadrightarrow [TA1, XB], XB \twoheadrightarrow [TB3, XD]\\
&\quad , XI \twoheadrightarrow [TF1, XG], XG \twoheadrightarrow [TB2, XA]\\
&\quad , S \twoheadrightarrow [TA1, XH]\\
&\quad , XH \twoheadrightarrow [TF4, XB], XH \twoheadrightarrow [TA3, XC]\\
&\quad , XC \twoheadrightarrow [TE4, XD], XC \twoheadrightarrow [TA3, XE]\\
&\quad , XE \twoheadrightarrow [TA1, XD], XE \twoheadrightarrow [TC1, XD]\\
&\quad , XC \twoheadrightarrow [TB1, XB]\\
&\quad , S \twoheadrightarrow [TB1, XF]\\
&\quad , XF \twoheadrightarrow [TA1, XJ]\ \ , XF \twoheadrightarrow [TD1, XG]
\end{aligned}
$$

Grammar for Tabla Improvisation

$$
\begin{aligned}
&, TA7 \twoheadrightarrow [\,Kt, Dha, Tr, Kt, Dha, Ge, Na\,] \\
&, TC2 \twoheadrightarrow [\,Tr, Kt\,],\;\; TE1 \twoheadrightarrow [\,Tr\,],\;\; TF1 \twoheadrightarrow [\,Kt\,] \\
&, TF4 \;\twoheadrightarrow [\,Ti, Dha, Tr, Kt\,] \\
&, TE4 \;\twoheadrightarrow [\,Ti, Noop, Dha, Ti\,] \\
&, TD1 \twoheadrightarrow [\,Noop\,],\;\; TB2 \twoheadrightarrow [\,Dha, Ti\,],\;\; TC1 \twoheadrightarrow [\,Ge\,] \\
&, TB3 \twoheadrightarrow [\,Dha, Tr, Kt\,] \\
&, TA8 \twoheadrightarrow [\,Dha, Ti, Dha, Ge, Dhee, Na, Ge, Na\,] \\
&, TA3 \twoheadrightarrow [\,Tr, Kt, Dha\,], TB1 \twoheadrightarrow [\,Ti\,], TA1 \twoheadrightarrow [\,Dha\,]\;\,]
\end{aligned}
$$

Interpretation: From Syllables to MIDI

```
instance ToMusic1 Syllable where
  toMusic1 = toMusic ∘ pitch ∘ percussionMap
    where
      percussionMap :: Syllable → Int
      percussionMap s = case s of Tr → 38
                                  Kt → 45
                                  ...
```

Generated Results

- Given a total duration and the required configurations, generate a ”music piece”:

$$
\begin{aligned}
&\mathit{generate} :: \mathit{FilePath} \rightarrow \mathit{Dur} \\
&\qquad\qquad \rightarrow \mathit{HarmonyConfig} \rightarrow \mathit{MelodyConfig} \\
&\qquad\qquad \rightarrow \mathit{IO}\ () \\
&\mathit{generate}\ f\ t\ \mathit{hCfg}\ \mathit{mCfg} = \textbf{do} \\
&\quad (\mathit{absHarm}, \mathit{harm}) \leftarrow \mathit{runGrammar}\ \mathit{harmony}\ t\ \mathit{hCfg} \\
&\quad (\_, \mathit{mel}) \leftarrow \mathit{runGrammar}\ \mathit{melody}\ t\ (\mathit{mCfg}, \mathit{absHarm}) \\
&\quad (\_, \mathit{rhy}) \leftarrow \mathit{runGrammar}\ \mathit{rhythm}\ t\ () \\
&\quad \mathit{writeToMidiFile}\ f\ (\mathit{harm} :=: \mathit{mel} :=: \mathit{rhy})
\end{aligned}
$$

```
sonata
  = generate "sonata" (12 * wn)
    HarmonyConfig
      { basePc    = E
      , baseOct   = 4
      , baseScale = minor
      , chords    = equally [mi, maj, dim]}
    MelodyConfig
      { scales  = equally [ionian, harmonicMinor]
      , octaves = [(5, 4), (20, 5), (10, 6)]}
```

```
orientalAlgebras
 = generate "oriental" (12 * wn)
    HarmonyConfig
       { basePc    = A
       , baseOct   = 3
       , baseScale = arabian -- ≡ [P1, M2, Mi3, P4, A4, Mi6, M7]
       , chords    = equally allChords }
    MelodyConfig
       { scales  = equally allScales
       , octaves = [(20, 4), (15, 5), (5, 6)] }
```

Conclusion

Future Work

- Jazz harmony (extend with context-sensitive features)
  - *[Steedman, 1984]* A generative grammar for jazz chord sequences
  - *[Steedman, 1996]* The blues and the abstract truth
- Better & more principled interpretation
- Intrinsically-typed grammars
- Non-musical Domains

Conclusion

It's gotta be simple, so people can dig it!

Thelonious Monk

Questions?