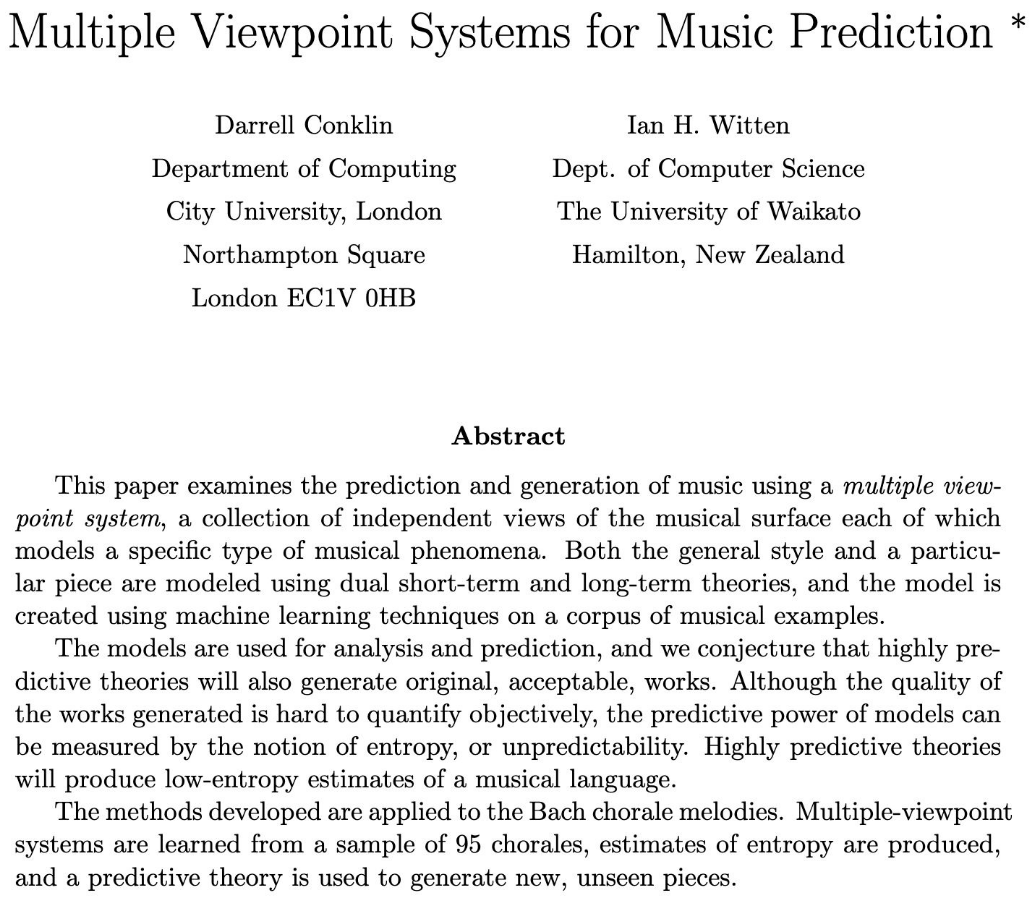
Constraint Based Systems

Prof. Michael Casey
CS89.29 / MUS14.05 / CS189.1

https://www.ehu.eus/cs-ikerbasque/conklin/papers/jnmr95.pdf



Multiple Viewpoint Systems for Music Prediction *

Darrell Conklin
Department of Computing
City University, London
Northampton Square
London EC1V 0HB

Ian H. Witten
Dept. of Computer Science
The University of Waikato
Hamilton, New Zealand

Abstract

This paper examines the prediction and generation of music using a *multiple viewpoint system*, a collection of independent views of the musical surface each of which models a specific type of musical phenomena. Both the general style and a particular piece are modeled using dual short-term and long-term theories, and the model is created using machine learning techniques on a corpus of musical examples.

The models are used for analysis and prediction, and we conjecture that highly predictive theories will also generate original, acceptable, works. Although the quality of the works generated is hard to quantify objectively, the predictive power of models can be measured by the notion of entropy, or unpredictability. Highly predictive theories will produce low-entropy estimates of a musical language.

The methods developed are applied to the Bach chorale melodies. Multiple-viewpoint systems are learned from a sample of 95 chorales, estimates of entropy are produced, and a predictive theory is used to generate new, unseen pieces.

Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence

https://ojs.aaai.org/index.php/AAAI/article/download/8661/8520

Enforcing Meter in Finite-Length Markov Sequences

Pierre Roy and **François Pachet**
Sony CSL
6, rue Amyot
75 005, Paris, France

Abstract

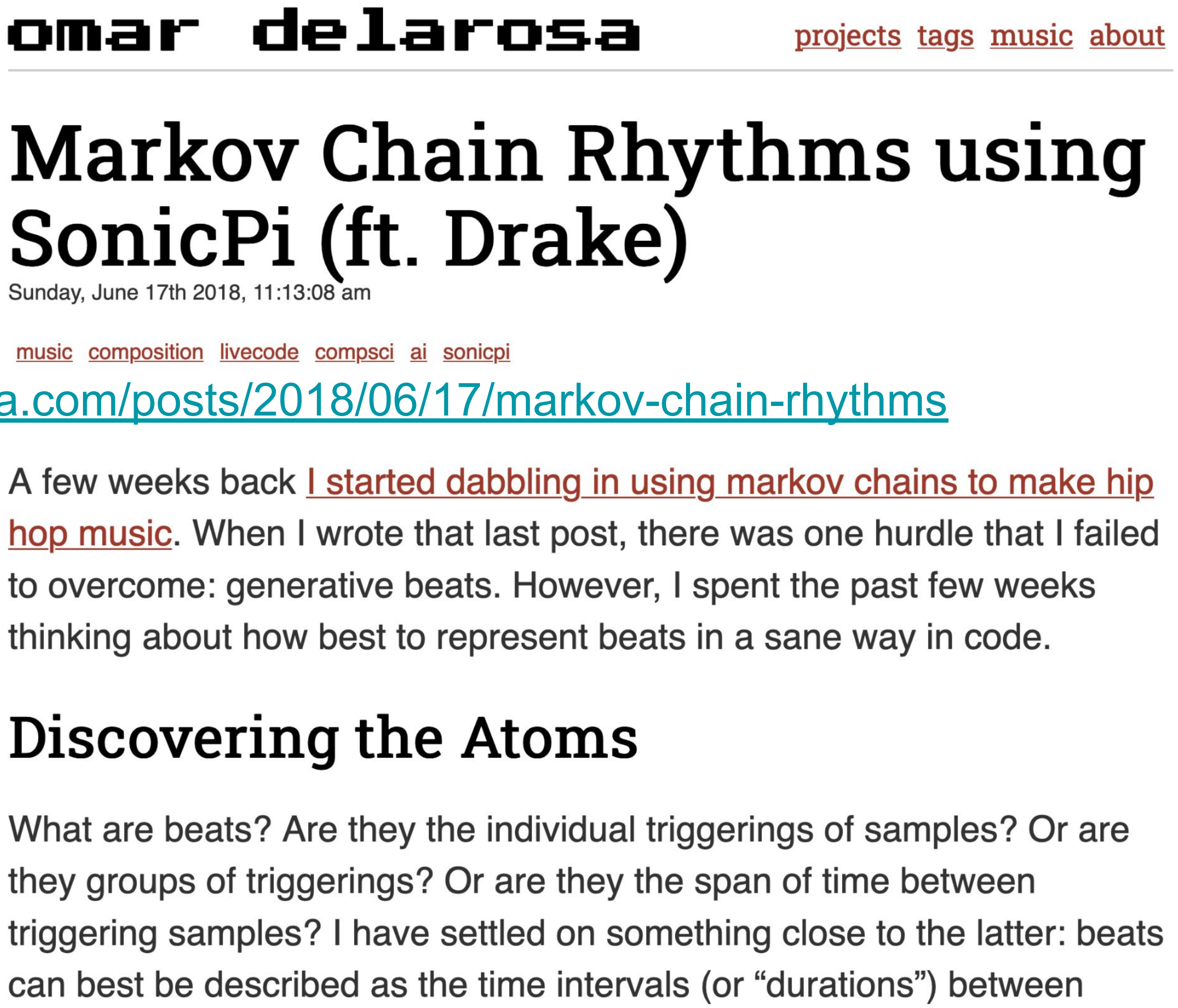
Markov processes are increasingly used to generate finite-length sequences that imitate a given style. However, Markov processes are notoriously difficult to control. Recently, Markov constraints have been introduced to give users some control on generated sequences. Markov constraints reformulate finite-length Markov sequence generation in the framework of constraint satisfaction (CSP). However, in practice, this approach is limited to local constraints and its performance is low for global constraints, such as cardinality or arithmetic constraints. This limitation prevents generated sequences to satisfy structural properties which are independent of the style, but inherent to the domain, such as meter. In this article, we introduce **`meter`**, a constraint that ensures a sequence is 1) Markovian with regards to a given corpus and 2) follows metrical rules expressed as cumulative cost functions. Additionally, **`meter`** can simultaneously enforce cardinality constraints. We propose a domain consistency algorithm whose complexity is pseudo-polynomial. This result is obtained thanks to a theorem on the growth of sumsets by Khovanskii. We illustrate our constraint on meter-constrained music generation problems that were so far not solvable by any

text, and toy text generators can easily be implemented to illustrate this point.

However, putting these ideas in practice raises difficult control problems: users generally want to enforce specific, domain-dependent properties on the sequences to generate. Unfortunately, Markov models, as most statistical models, do not offer natural handles to enforce such properties. The reason is that semantically interesting properties often establish long-range correlations between non contiguous items in the sequence, which is incompatible with the Markov hypothesis of limited dependency.

Markov Constraints

A general solution to this problem, called *Markov constraints*, was introduced in (Pachet and Roy 2011). It consists in reformulating Markov generation in the context of constraint satisfaction. The sequence to generate is viewed as a sequence of constrained variables, and Markovianity is represented as a constraint holding on consecutive pairs of contiguous variables (for order 1). Control constraints can then be expressed as arbitrary additional constraints, including global constraints. However, such a reformulation is costly as there is no boundary, in principle, on the com-



omar delarosa

projects tags music about

Markov Chain Rhythms using SonicPi (ft. Drake)

Sunday, June 17th 2018, 11:13:08 am

music composition livecode compsci ai sonicpi

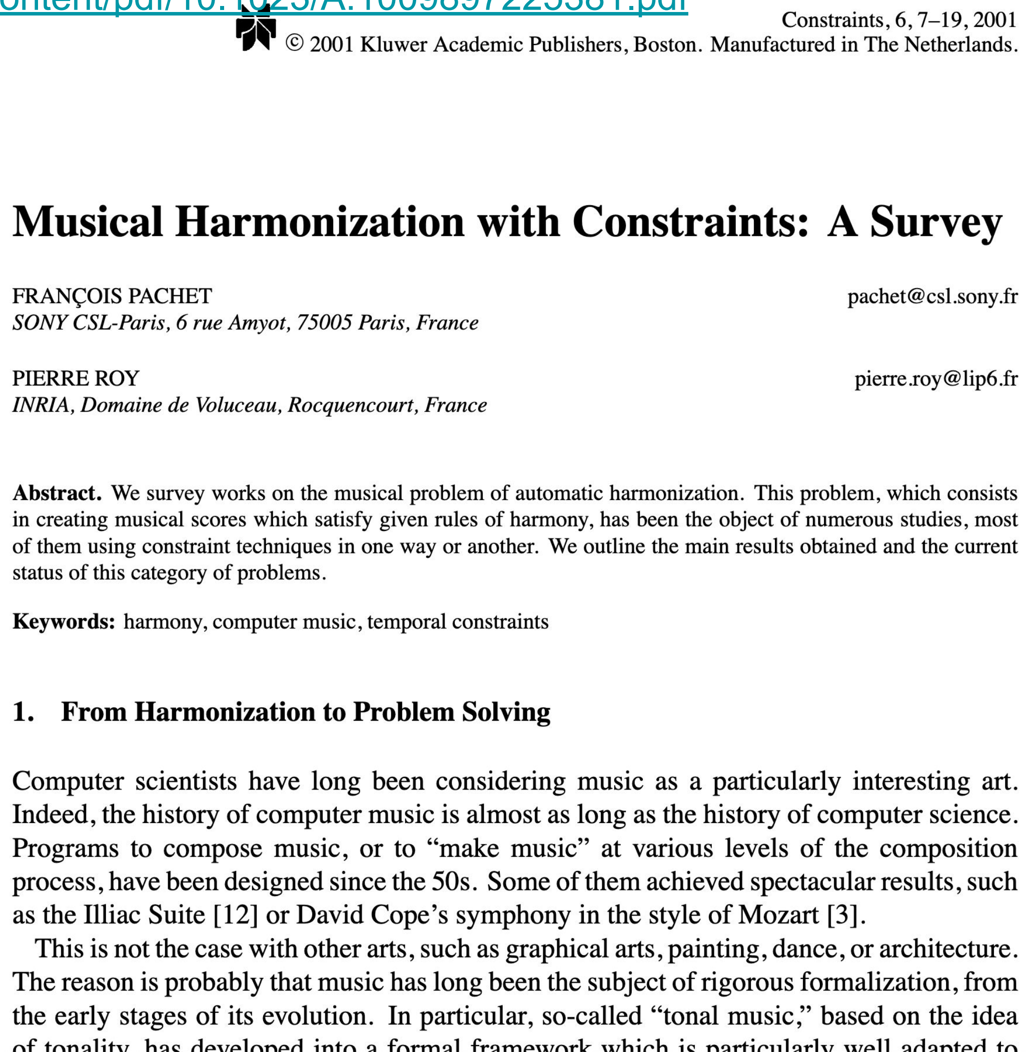
https://www.omardelarosa.com/posts/2018/06/17/markov-chain-rhythms

A few weeks back I started dabbling in using markov chains to make hip hop music. When I wrote that last post, there was one hurdle that I failed to overcome: generative beats. However, I spent the past few weeks thinking about how best to represent beats in a sane way in code.

Discovering the Atoms

What are beats? Are they the individual triggerings of samples? Or are they groups of triggerings? Or are they the span of time between triggering samples? I have settled on something close to the latter: beats can best be described as the time intervals (or “durations”) between https://link.springer.com/content/pdf/10.1023/A:1009897225381.pdf

Constraints



Constraints, 6, 7–19, 2001

Musical Harmonization with Constraints: A Survey

FRANÇOIS PACHET pachet@csl.sony.fr
SONY CSL-Paris, 6 rue Amyot, 75005 Paris, France

PIERRE ROY pierre.roy@lip6.fr
INRIA, Domaine de Voluceau, Rocquencourt, France

Abstract. We survey works on the musical problem of automatic harmonization. This problem, which consists in creating musical scores which satisfy given rules of harmony, has been the object of numerous studies, most of them using constraint techniques in one way or another. We outline the main results obtained and the current status of this category of problems.

Keywords: harmony, computer music, temporal constraints

1. From Harmonization to Problem Solving

Computer scientists have long been considering music as a particularly interesting art. Indeed, the history of computer music is almost as long as the history of computer science. Programs to compose music, or to "make music" at various levels of the composition process, have been designed since the 50s. Some of them achieved spectacular results, such as the Illiac Suite [12] or David Cope's symphony in the style of Mozart [3].

This is not the case with other arts, such as graphical arts, painting, dance, or architecture. The reason is probably that music has long been the subject of rigorous formalization, from the early stages of its evolution. In particular, so-called "tonal music," based on the idea of tonality, has developed into a formal framework which is particularly well adapted to

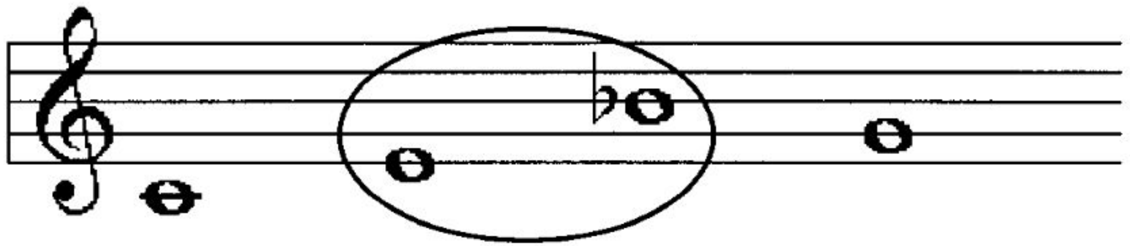


Figure 1. The two notes in the oval violate the rule about forbidden horizontal intervals (here, an ascending tritone interval between *E* and *B flat*). The two other intervals are legal (ascending major third and descending minor third).

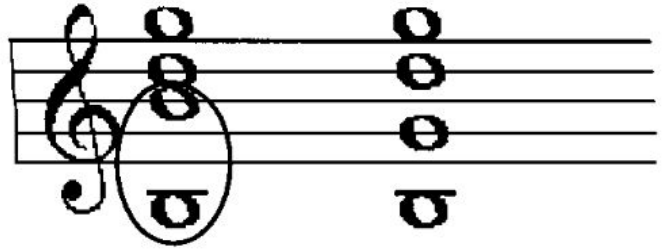
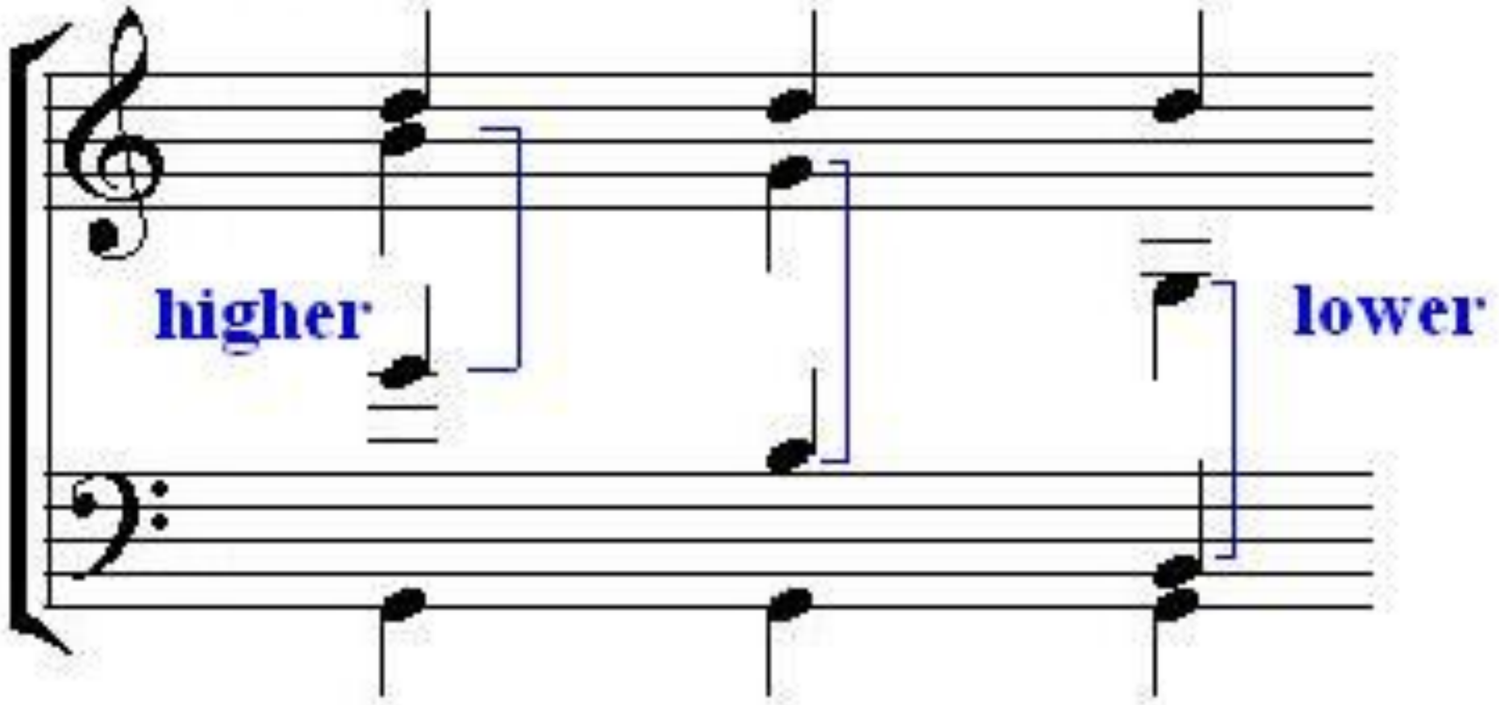
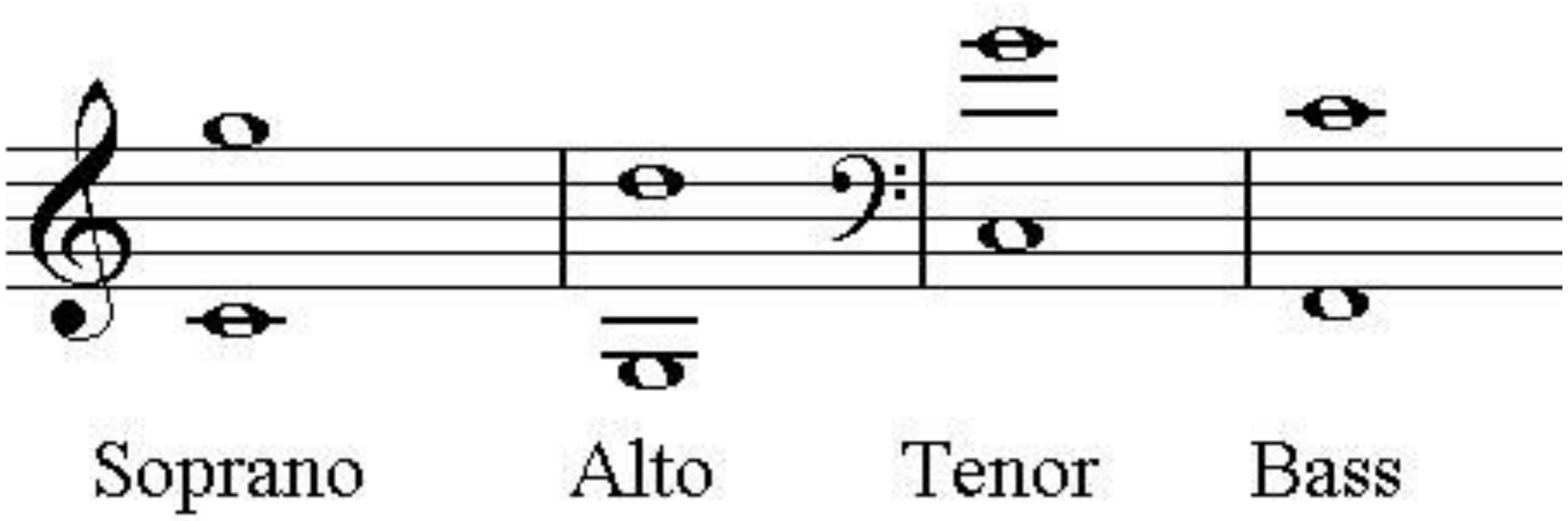


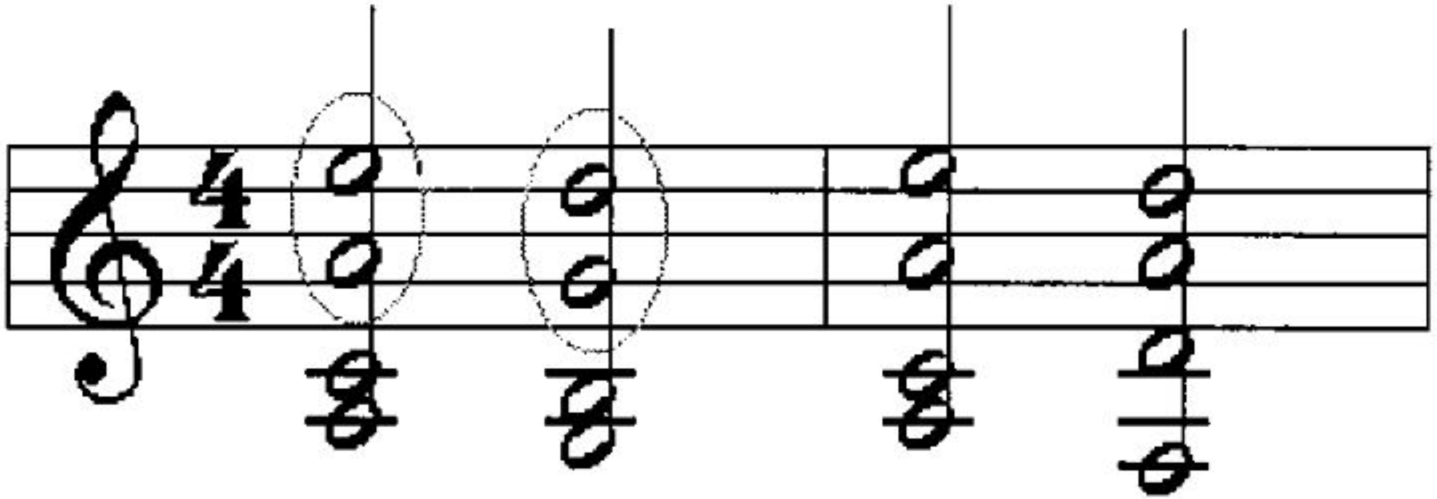
Figure 2. The two notes of the first chord in the oval violate the rule about forbidden repetition of the bass of a “6” chord. The next chord (also a “6” chord) is legal.


higher
lower

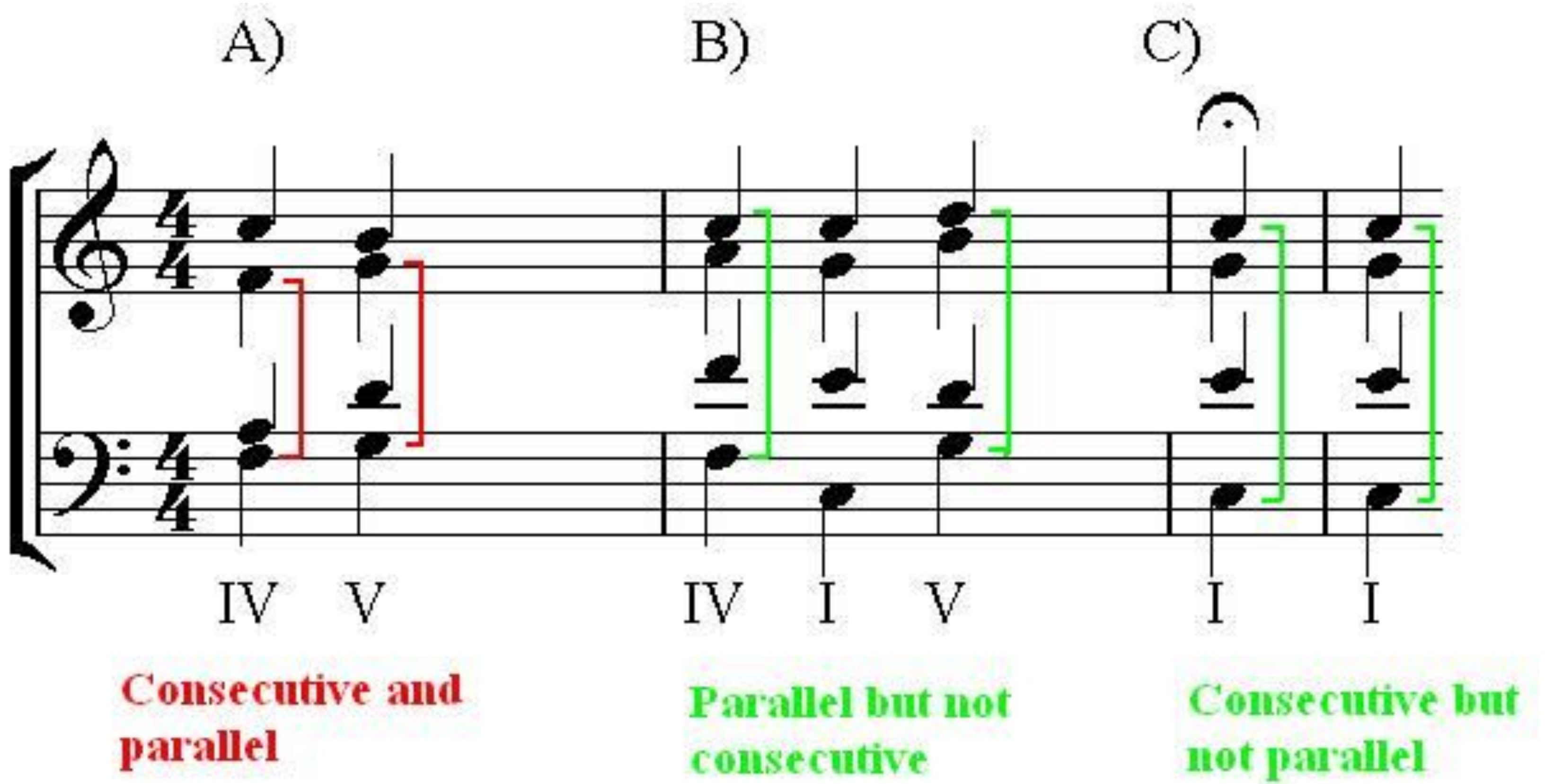
Pitch range (Tessitura)



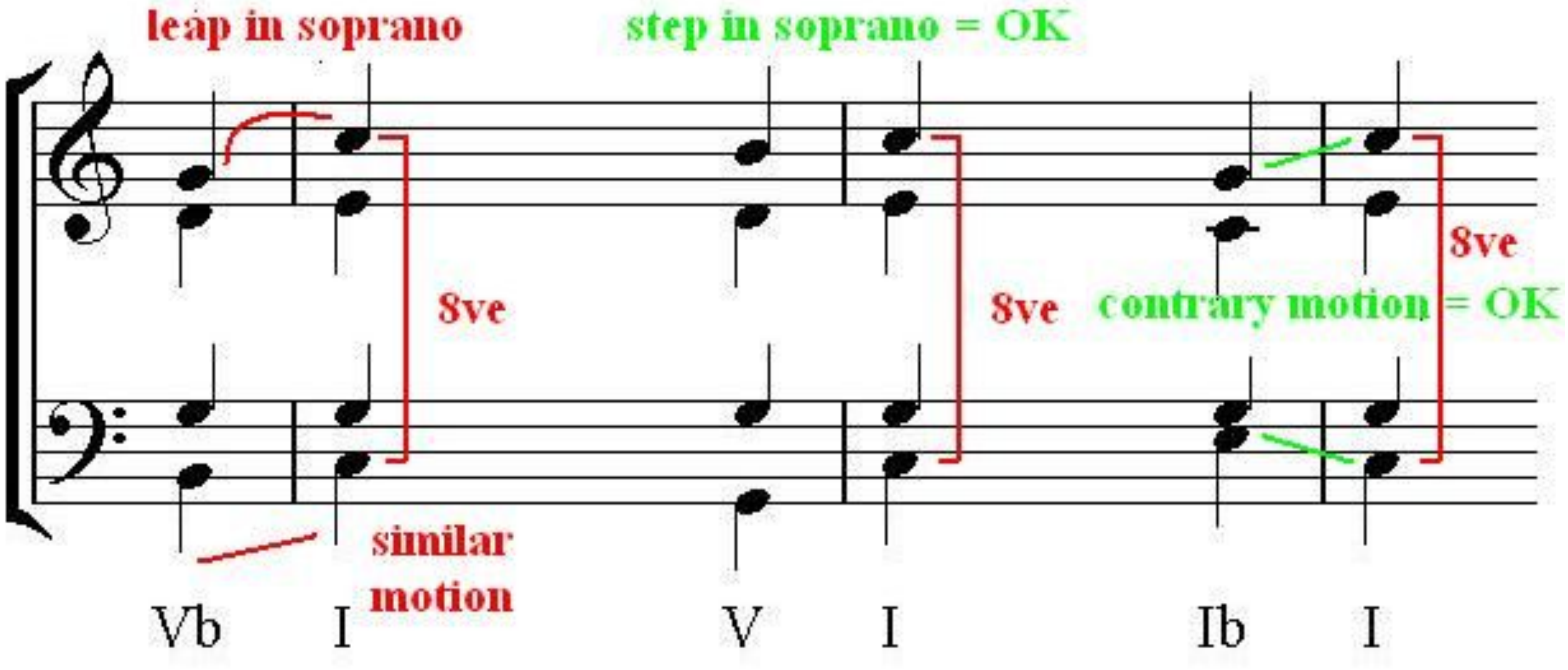
Parallels



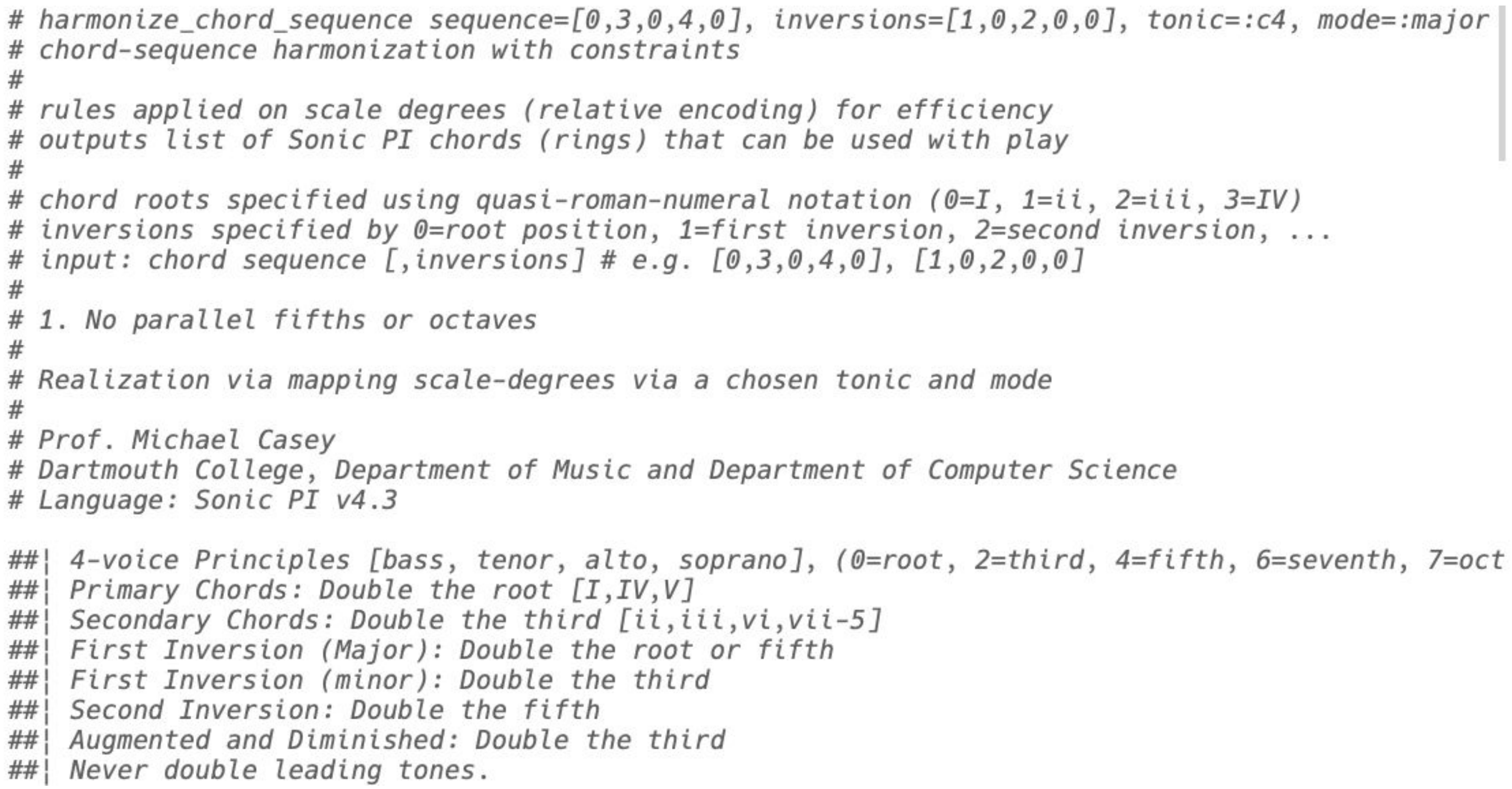
Rules of (Western Tonal) Harmony



Rules of (Western Tonal) Harmony



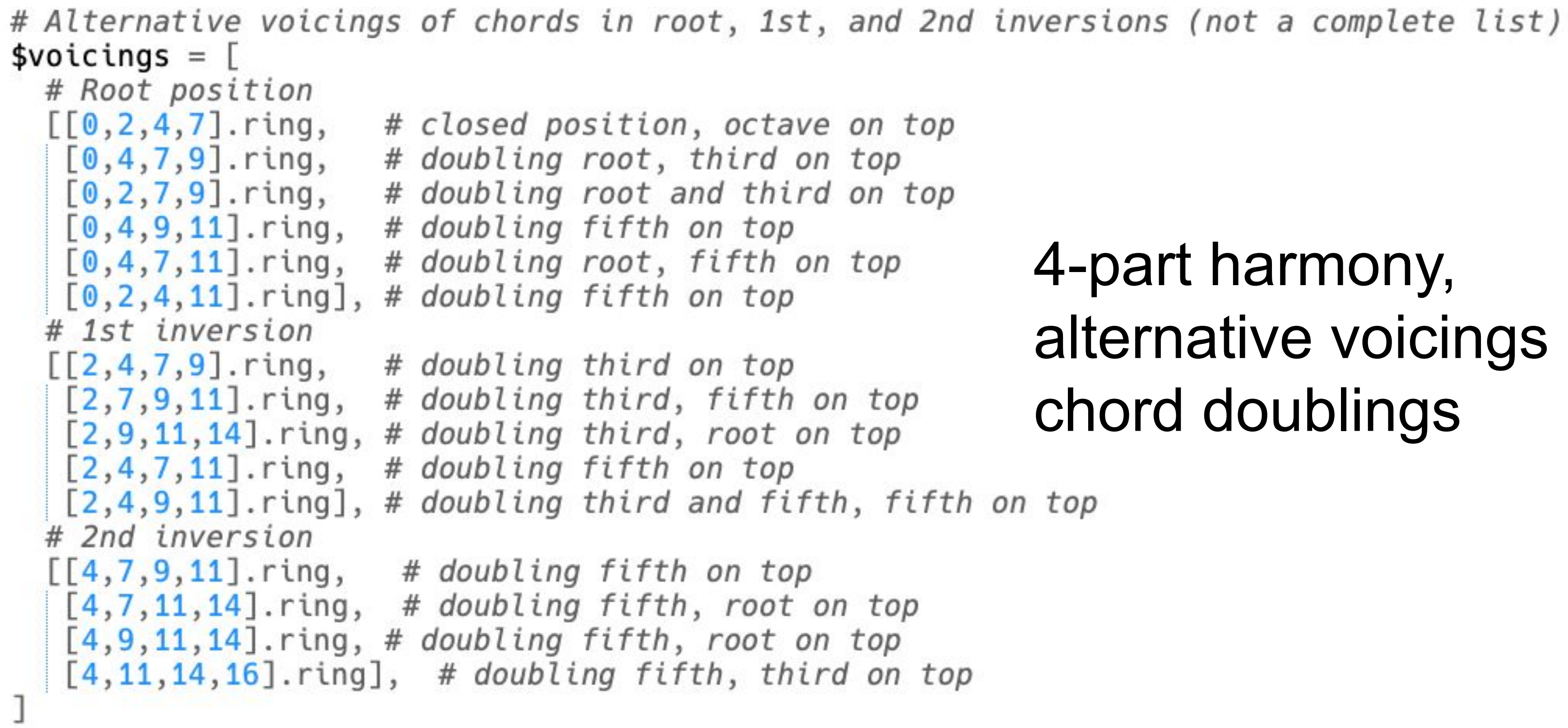
Constraints applied to chord-sequence realization



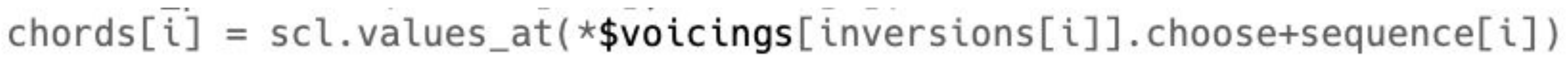
```
# harmonize_chord_sequence sequence=[0,3,0,4,0], inversions=[1,0,2,0,0], tonic=:c4, mode=:major
# chord-sequence harmonization with constraints
#
# rules applied on scale degrees (relative encoding) for efficiency
# outputs list of Sonic PI chords (rings) that can be used with play
#
# chord roots specified using quasi-roman-numeral notation (0=I, 1=ii, 2=iii, 3=IV)
# inversions specified by 0=root position, 1=first inversion, 2=second inversion, ...
# input: chord sequence [,inversions] # e.g. [0,3,0,4,0], [1,0,2,0,0]
#
# 1. No parallel fifths or octaves
#
# Realization via mapping scale-degrees via a chosen tonic and mode
#
# Prof. Michael Casey
# Dartmouth College, Department of Music and Department of Computer Science
# Language: Sonic PI v4.3

##¦ 4-voice Principles [bass, tenor, alto, soprano], (0=root, 2=third, 4=fifth, 6=seventh, 7=oct
##¦ Primary Chords: Double the root [I,IV,V]
##¦ Secondary Chords: Double the third [ii,iii,vi,vii-5]
##¦ First Inversion (Major): Double the root or fifth
##¦ First Inversion (minor): Double the third
##¦ Second Inversion: Double the fifth
##¦ Augmented and Diminished: Double the third
##¦ Never double leading tones.
```

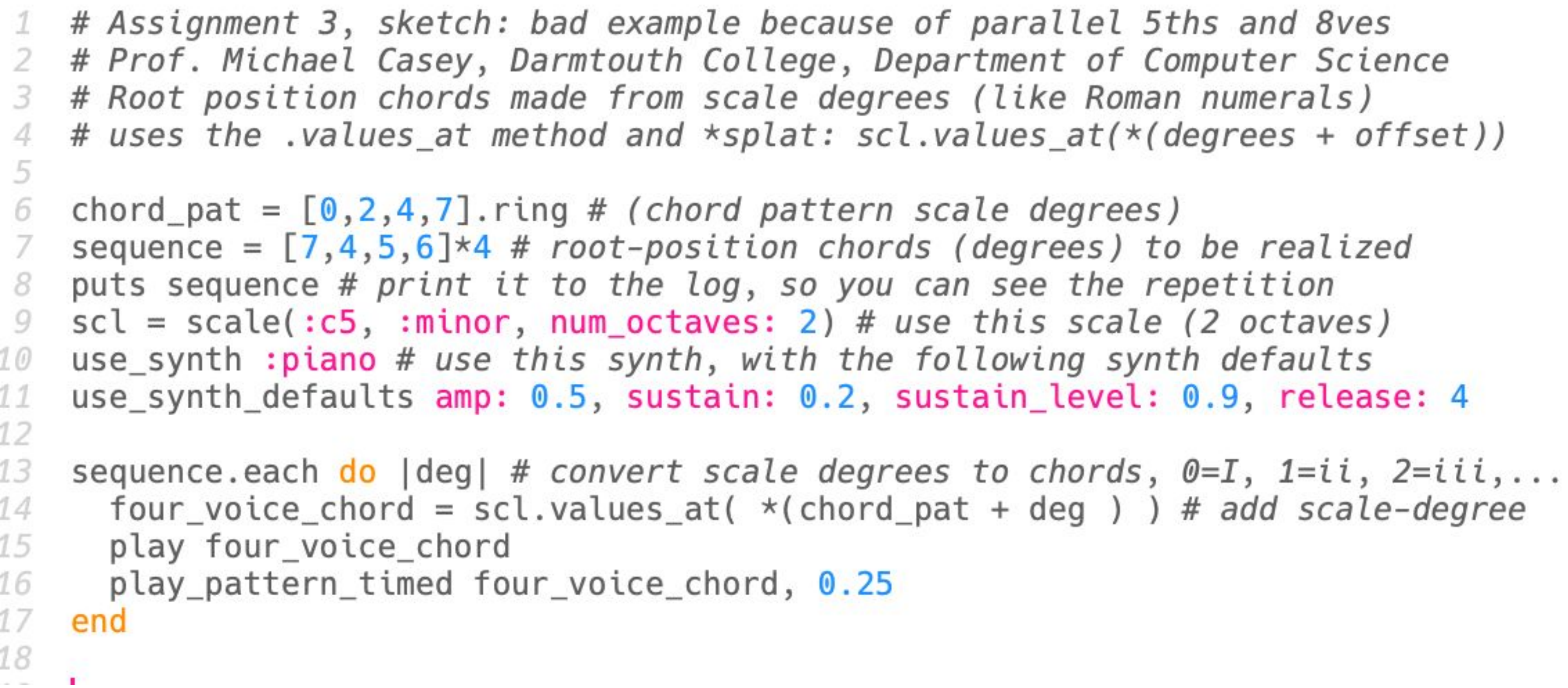
4-part harmony, alternative voicings and chord doublings



```
# Alternative voicings of chords in root, 1st, and 2nd inversions (not a complete list)
$voicings = [
  # Root position
  [[0,2,4,7].ring,   # closed position, octave on top
   [0,4,7,9].ring,   # doubling root, third on top
   [0,2,7,9].ring,   # doubling root and third on top
   [0,4,9,11].ring,  # doubling fifth on top
   [0,4,7,11].ring,  # doubling root, fifth on top
   [0,2,4,11].ring], # doubling fifth on top
  # 1st inversion
  [[2,4,7,9].ring,   # doubling third on top
   [2,7,9,11].ring,  # doubling third, fifth on top
   [2,9,11,14].ring, # doubling third, root on top
   [2,4,7,11].ring,  # doubling fifth on top
   [2,4,9,11].ring], # doubling third and fifth, fifth on top
  # 2nd inversion
  [[4,7,9,11].ring,   # doubling fifth on top
   [4,7,11,14].ring,  # doubling fifth, root on top
   [4,9,11,14].ring, # doubling fifth, root on top
   [4,11,14,16].ring],  # doubling fifth, third on top
]
```



```
chords[i] = scl.values_at(*$voicings[inversions[i]].choose+sequence[i])
```



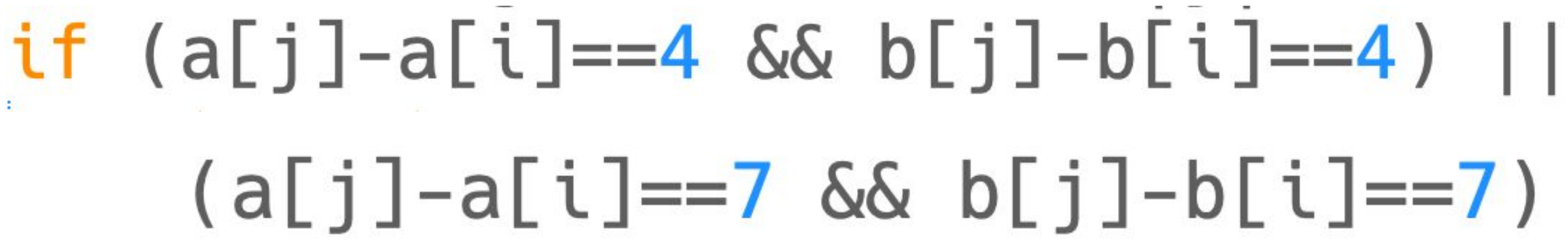
```
# Assignment 3, sketch: bad example because of parallel 5ths and 8ves
# Prof. Michael Casey, Darmtouth College, Department of Computer Science
# Root position chords made from scale degrees (like Roman numerals)
# uses the .values_at method and *splat: scl.values_at(*(degrees + offset))

chord_pat = [0,2,4,7].ring # (chord pattern scale degrees)
sequence = [7,4,5,6]*4 # root-position chords (degrees) to be realized
puts sequence # print it to the log, so you can see the repetition
scl = scale(:c5, :minor, num_octaves: 2) # use this scale (2 octaves)
use_synth :piano # use this synth, with the following synth defaults
use_synth_defaults amp: 0.5, sustain: 0.2, sustain_level: 0.9, release: 4

sequence.each do |deg| # convert scale degrees to chords, 0=I, 1=ii, 2=iii,...
  four_voice_chord = scl.values_at( *(chord_pat + deg ) ) # add scale-degree
  play four_voice_chord
  play_pattern_timed four_voice_chord, 0.25
end
```

Checking for Parallel 5ths and Octaves (scale degrees)

test chords a and b for each pair of voices i and j, e.g. chord **a** bass a[0] & alto a[2], chord **b** bass b[0] and alto b[2] :



```
if (a[j]-a[i]==4 && b[j]-b[i]==4) ||
     (a[j]-a[i]==7 && b[j]-b[i]==7)
```