to kill a monarch

Preferably played in a dark or dim setting (e.g. with the least light needed by the performers).

michael winter
(berlin, de; 2021)

---

instrumentation and dedication

This piece was originally written for quartertone bass flute and three strings. There are also two electronic accompaniment parts (synthesized by a custom computer program written in SuperCollider).

However, the instrumentation is flexible such that the part labeled in the score by an asterisks (the part originally intended for flute; referred to here as part 'star') has a distinct timbre from the other three parts which are labeled by Roman numerals and should be rather homogeneous in timbre within themselves (the parts originally intended for strings, referred to here as the 'candidate' parts). These three parts may also be synthesized and played back with the electronic accompaniments (individually referred to as 'electronic accompaniment' I and II). Conversely, the part star can be synthesized, while the candidate parts are played on acoustic instruments.

That is, the piece may be played as a solo, trio, or quartet. The reason for this variability stems from a sort of economy-of-means. After moving to Berlin, I started discussing with Rebecca Lane writing a piece for her to play on microtonal-bass flute. At the same time, three mutual friends of ours, all cellists, Deborah Walker, Judith Hammann, and Lucy Railton, were entertaining the idea of starting a cello trio as the former two had also recently moved to Berlin. This flexible instrumentation gives the option for each of them to play the piece as a solo or together in different configurations. There is a certain pragmatism, however, that makes the part star playable on a microtonal bass flute and the candidate parts specifically suited for strings. These idiosyncrasies are given in more detail in the part descriptions below.

I would like to extend a special thanks to Rebecca Lane who compelled the piece. For encouraging me to write it and more specifically, for her suggestion early on to be fastidious about notating all the interrelations among and within the parts. Doing so led to a comprehensive analysis and better understanding of the underlying process which ultimately made the piece possible.

process and structure (the first paragraph may be used as a short program note)

The first half of the piece is a sort of extended prelude. The second half is a series of modulations / interpolations where each modulation goes from one mode, always a gamut of 7 pitches built upon a given root / fundamental, to another. The interpolation is governed by an algorithm that models the phenomenon where the rich-get-richer such that the more pitches that have been selected from one of a set of potential 'candidate' modes, the more likely that candidate will eventually become the destination mode. Once a mode 'wins' (becomes overwhelming rich), its wealth is effectively stripped and it can no longer persist.

The current state of the interpolation is articulated by part star and electronic accompaniment I while the candidate parts articulate candidate modes that *may eventually become* the destination mode. That is, at any point in time, the pitch gamut of part star and electronic accompaniment I is comprised of subsets of pitches from the candidate modes articulated. As such, part star and electronic accompaniment I are typically multimodal (built upon 2 or more roots) while, individually, the candidate parts are always unimodal (each built upon one root). The exception being when a destination mode is reached and all parts play the same gamut of pitches built upon a single root.

The score is divided into sections and subsections. The first section is the extended prelude. Starting from section 2, each section is one full modulation of the above-described process where each subsection can be considered as a discrete point in the interpolation. The destination mode is always reached by the ultimate subsection of each section.

notation

At the beginning of section 1 and each subsection from section 2 onward, a key is provided above the staff that indicates the current root for each mode articulated by each of the candidate parts, respectively; the relationships among the roots; and the relationship of the previous root to the current root within each part. The relationships among the roots are given by frequency ratios written above lines that connect the part numbers (in Roman numerals). The relationship of the previous root to the current root within each part is given by a frequency ratio written below the note of the current root. The note indicating the pitch of the current root is preceded by a note (given in gray) indicating the pitch of the previous root unless the root has not changed.

For the candidate parts, each note indicates the closest pitch in twelve-tone equal temperament with a cent-deviation (100th of a tempered semitone) written above and a frequency ratio from the current root written below. Part star is the same except that the frequency ratio may be written as a superscript of a Roman numeral that indicates which root (of the mode from candidate part I, II, or III) the frequency ratio is referencing. If no Roman numeral is given, the last one is assumed.

All frequency ratios are given in a *collapsed* form as if the pitches were within one octave above the same arbitrary C and always in the form where the numerator is greater than the denominator.

candidate parts

These parts are designed such that within each subsection, each part only sounds tones with pitches from one mode. A mode is always a 7 pitch gamut in the following form (given by frequency ratios and cents from the 1/1):

		5/4 (386¢)	11/8 (551¢)		13/8 (841¢)	7/4 (969¢)
1/1	9/8 (204¢)			3/2 (702¢)		
		6/5 (316¢)	4/3 (498¢)		8/5 (813¢)	15/8 (1088¢)

Horizontally aligned frequency ratios indicate that one or the other pitch may be used in the mode (most likely the top one). Therefore, only a handful of modes are possible even though the root progresses / changes throughout. Since the mode structure is rather limited, the players need to be able to transpose the possible modes arbitrarily. This is why these parts are specifically suited for strings. Basically the intervals within the modes stay the same or similar, but the position on the instrument changes.

part star

The pitch gamut of this part often comprises notes derived from several candidate modes at once. As mentioned above, a Roman numeral indicates the part that is articulating the candidate mode from which the note is drawn from. When the gamut is multimodal, the part generally has a lower temporal density and the sequence of pitches is always rising: each pitch is followed by the next highest pitch in the gamut until an upper limit is reached. This should make playing the part more feasible for a wind instrument like bass flute despite the complexity. Also, if the final note in a rising sequence is too high or feels too rushed, the written penultimate note can be extended till the sequence drops back down instead of playing the written ultimate note. Throughout the first section and in each ultimate subsection from section 2 onward, the part comprises arbitrary sequences of notes like the other parts. Therefore, there is a distinct shift starting at section 2, where the rising sequences begin. This should be made as clear as possible. Similarly, the interruption of the rising scale in the ultimate subsection of each section should also be made as clear as possible. If necessary, this part can be transposed up or down an octave. The part is written assuming bass flute hence the octavation marking below the treble clef; i.e. sounding an octave lower.

electronic accompaniments I and II

Electronic Accompaniment I articulates the current state of the interpolation as mentioned previously. Electronic Accompaniment II articulates the overall section-per-section form by swelling throughout each section on a tone with a pitch that is a perfect 5th above or a perfect 4th below the root of the destination mode which then cadences to the root itself in the ultimate subsection of each section.

dynamics

Each section should generally have a dynamic crescendo that peaks in the ultimate subsection. This can be executed by following the dynamic envelope of electronic accompaniment I where the candidate parts sound within / equal to the electronics and part star sounds slightly above / in the foreground. Generally, the sound should be rather present; filling the room more and more throughout each crescendo. However, the beginning of the crescendo need not be extremely quiet nor the peak excessively loud. With that said, the cadential peak of electronic accompaniment II in each ultimate subsection should briefly overwhelm all the other parts.

Within each subsection, each part often has a small flourish of activities which should be articulated as sub-swells within the larger dynamic profile with peaks based on the temporal density.

tempo

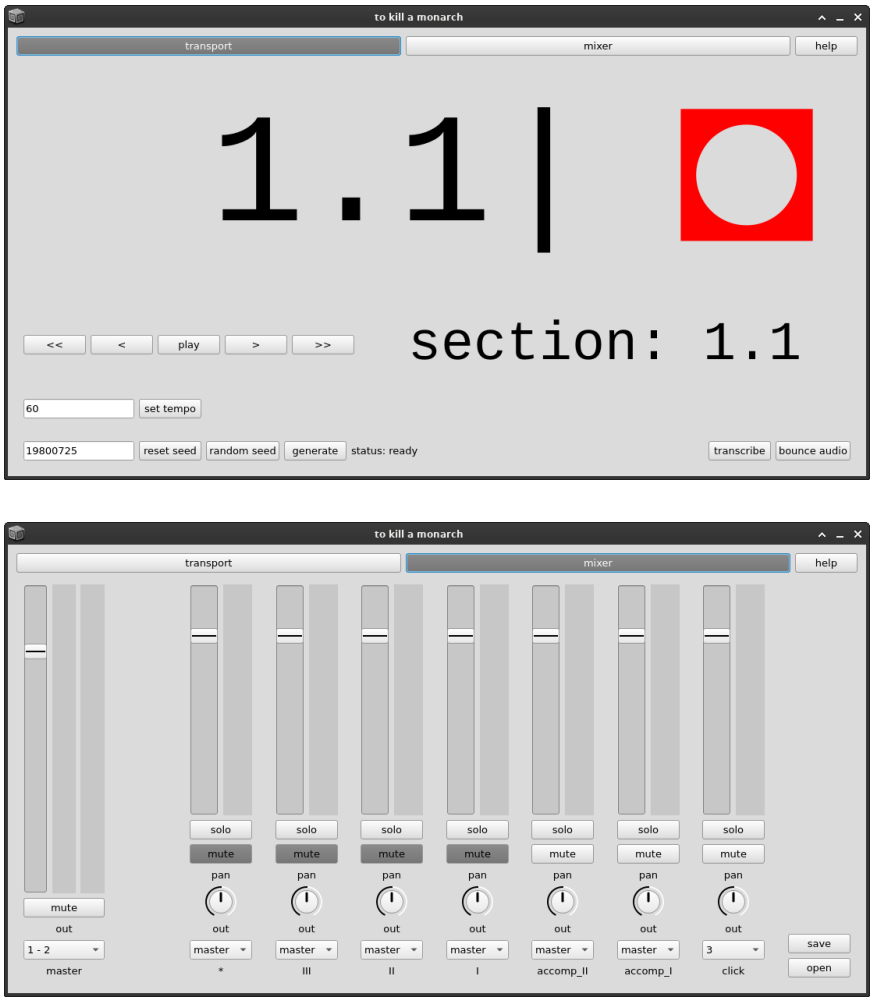
The score is written in a cut time with a tempo where the half note equals 60 beats per minute. The piece maybe be played at a slower tempo, but no less than the half note at 50 beats per minute.

SuperCollider program

A custom program written in the SuperCollider language can synthesize the parts that can be played by acoustic instruments and the strictly electronic accompaniments. A version of the application source code is appended at the end of this score. However, it is recommended to ensure that the most recent version of the code is being used which can be downloadable from a git repository at: `https://unboundedpress.org/code/mwinter/to_kill_a_monarch`

The application provides a transport window to control playback and set variables as well as a basic mixing console to control the levels of the various sonic elements of the piece. The program also allows new versions of the piece to be generated, transcribed, and rendered to separate audio files for use with other playback systems. Note that while most of the code facilitates usability, playback, and transcription, the music of the piece is completely generated by the algorithm in `tkam_musical_data_generator.scd`. A help / readme file is included with the application documenting its functionality and use. To launch the application, execute `tkam_main.scd` in SuperCollider (on Linux, this is achieved by pressing cmd+enter with the cursor anywhere within the code block).

The generation of this document (using LaTex) contains a version date at the bottom of this page in order to help track changes and the git repository will also detail commit changes. The piece was last generated using SuperCollider version 3.13 and Lilypond version 2.24.1.



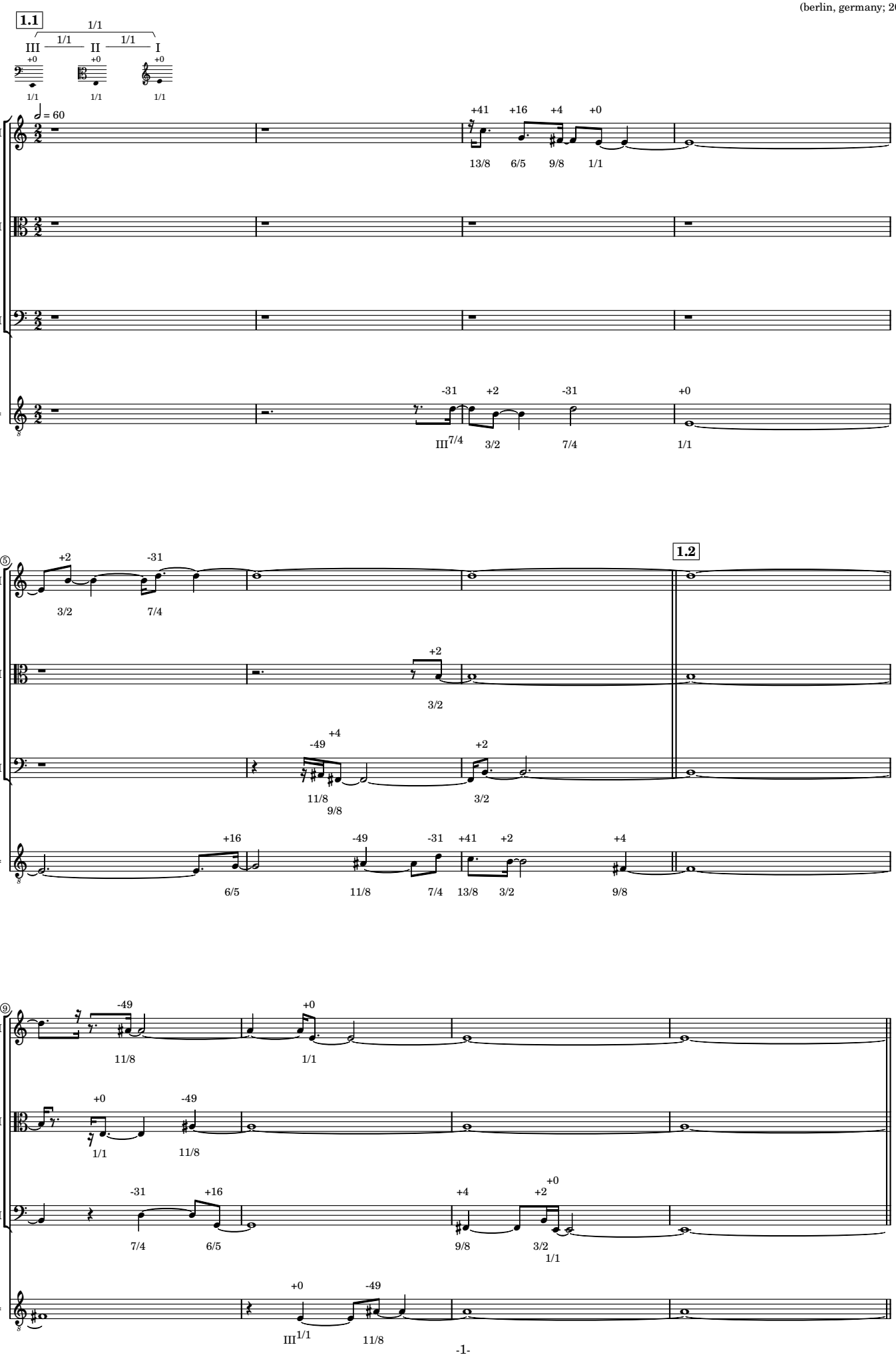
application user interface

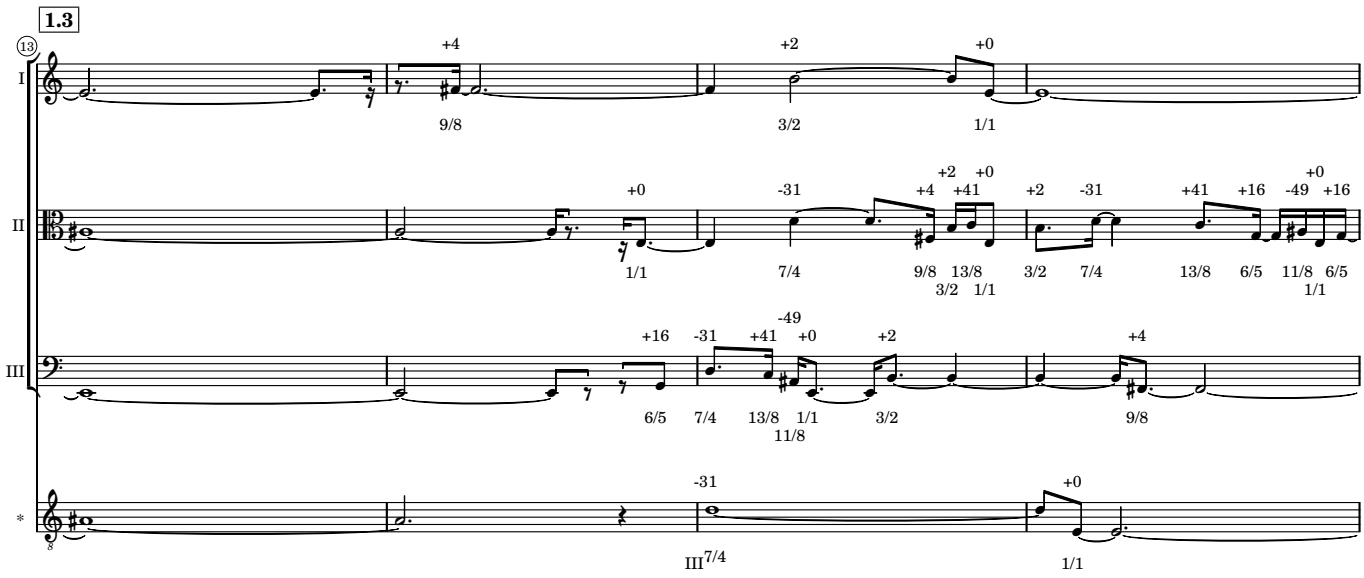
version generated: 2023.08.05

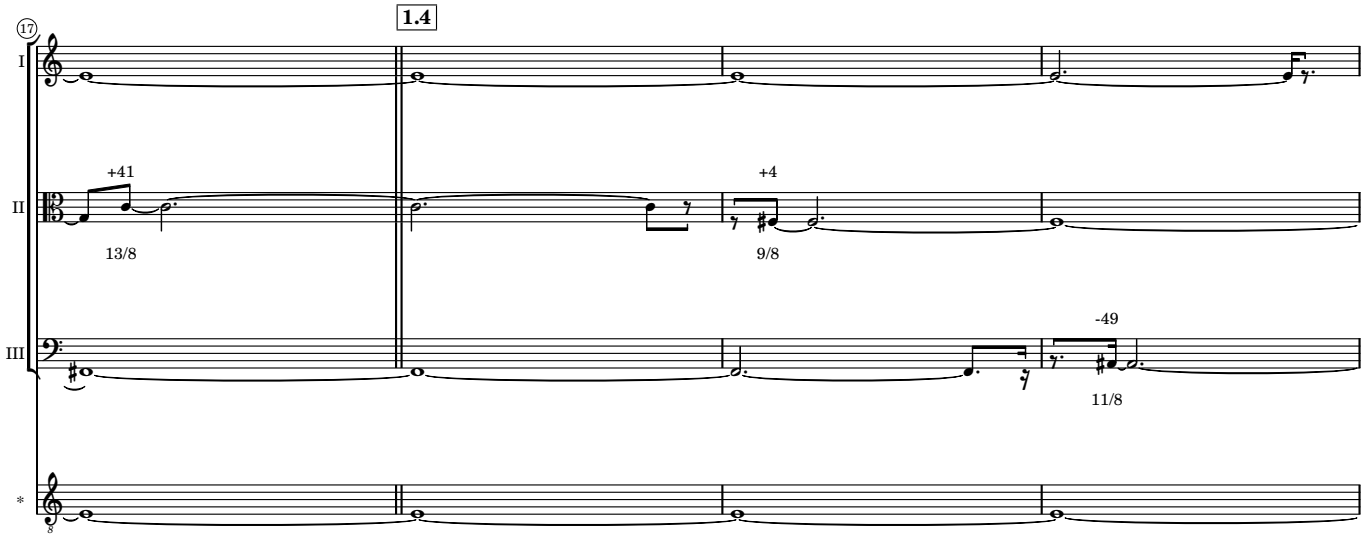
to kill a monarch

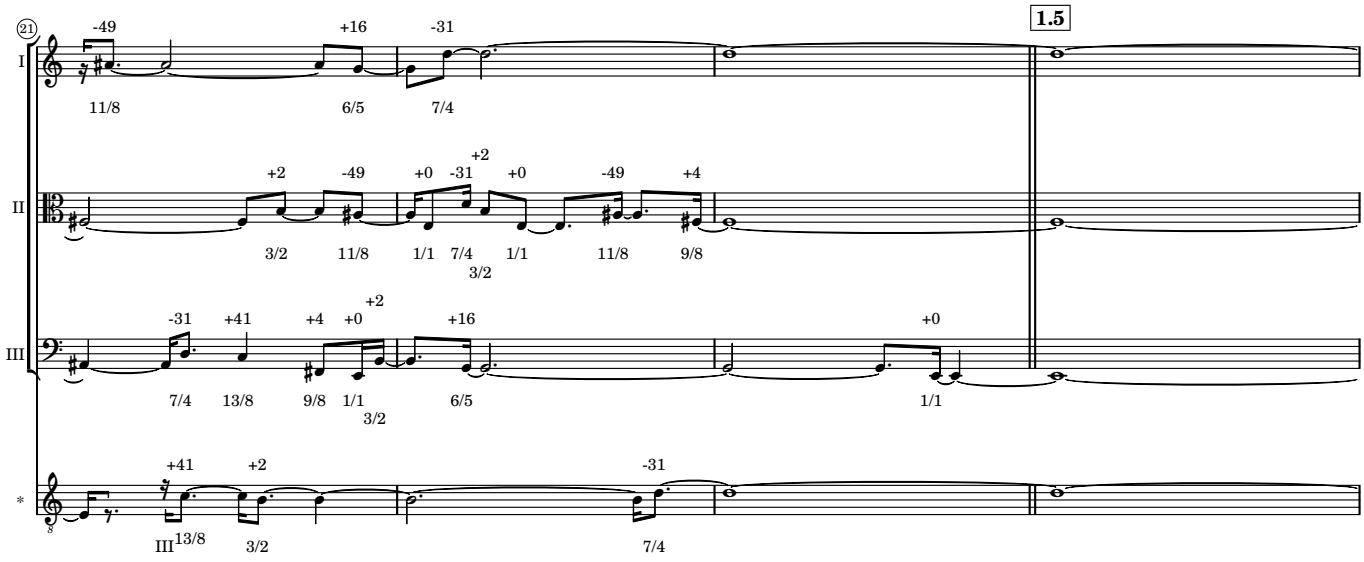
seed: 19800725

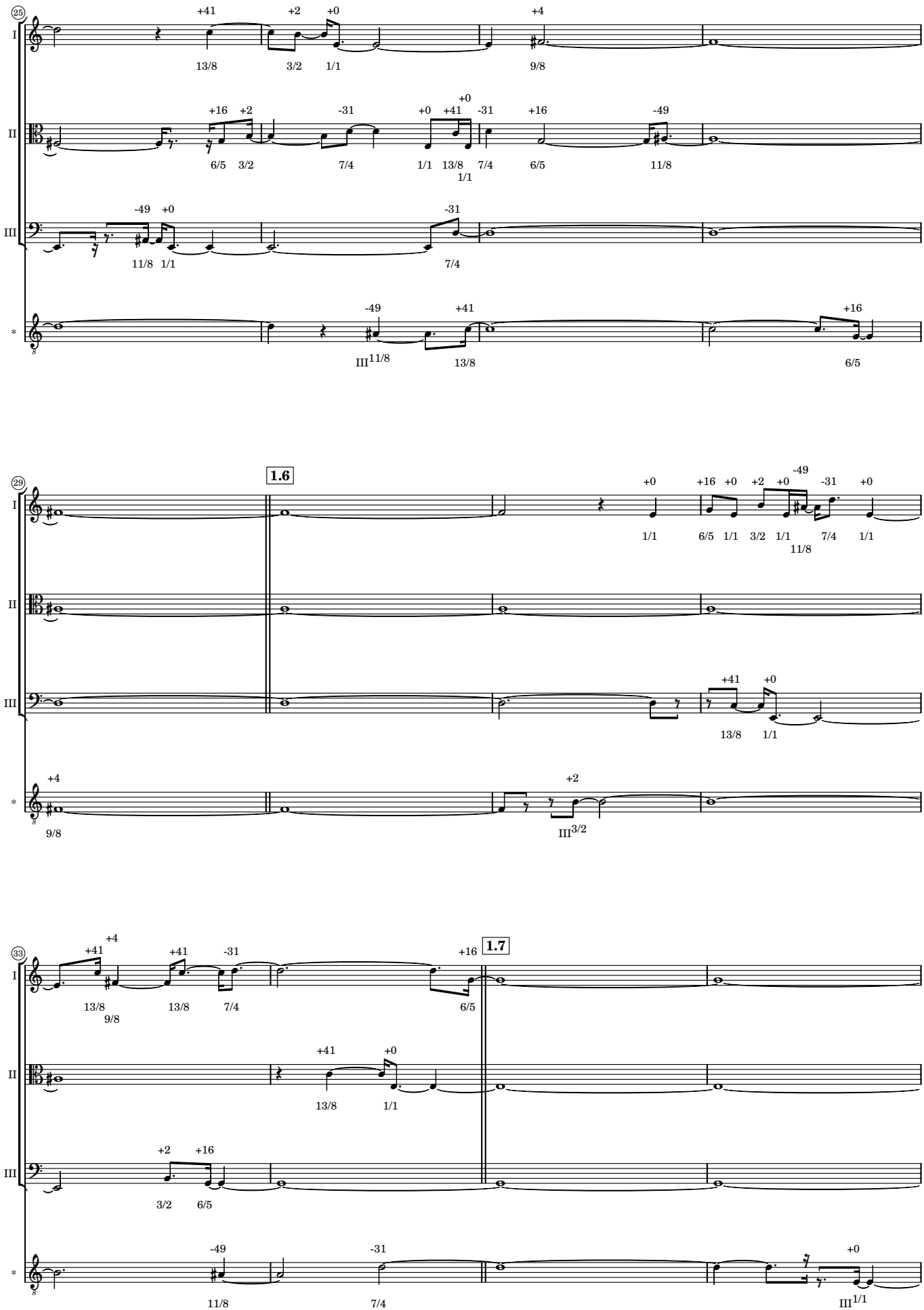
michael winter
(berlin, germany; 2021)

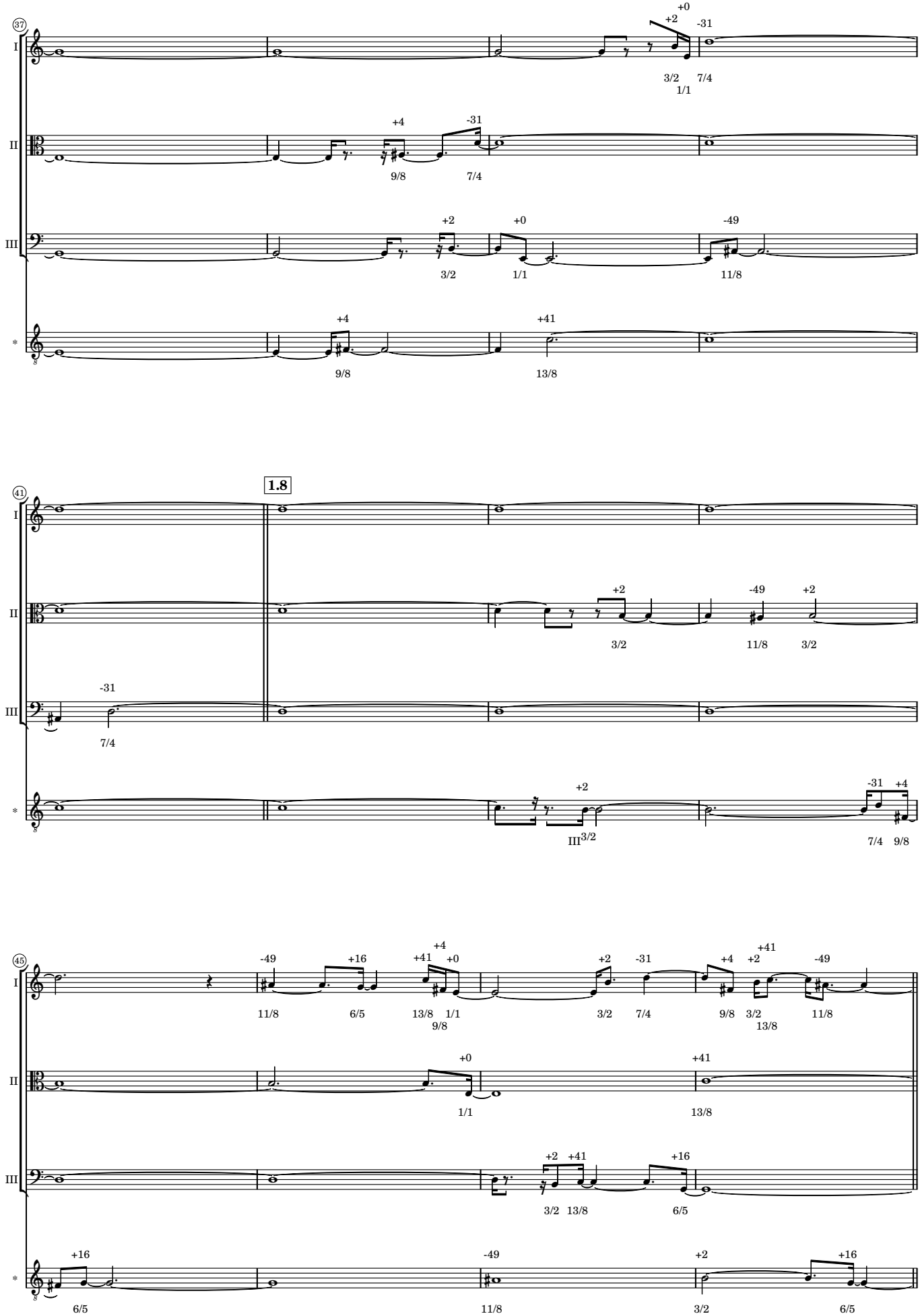


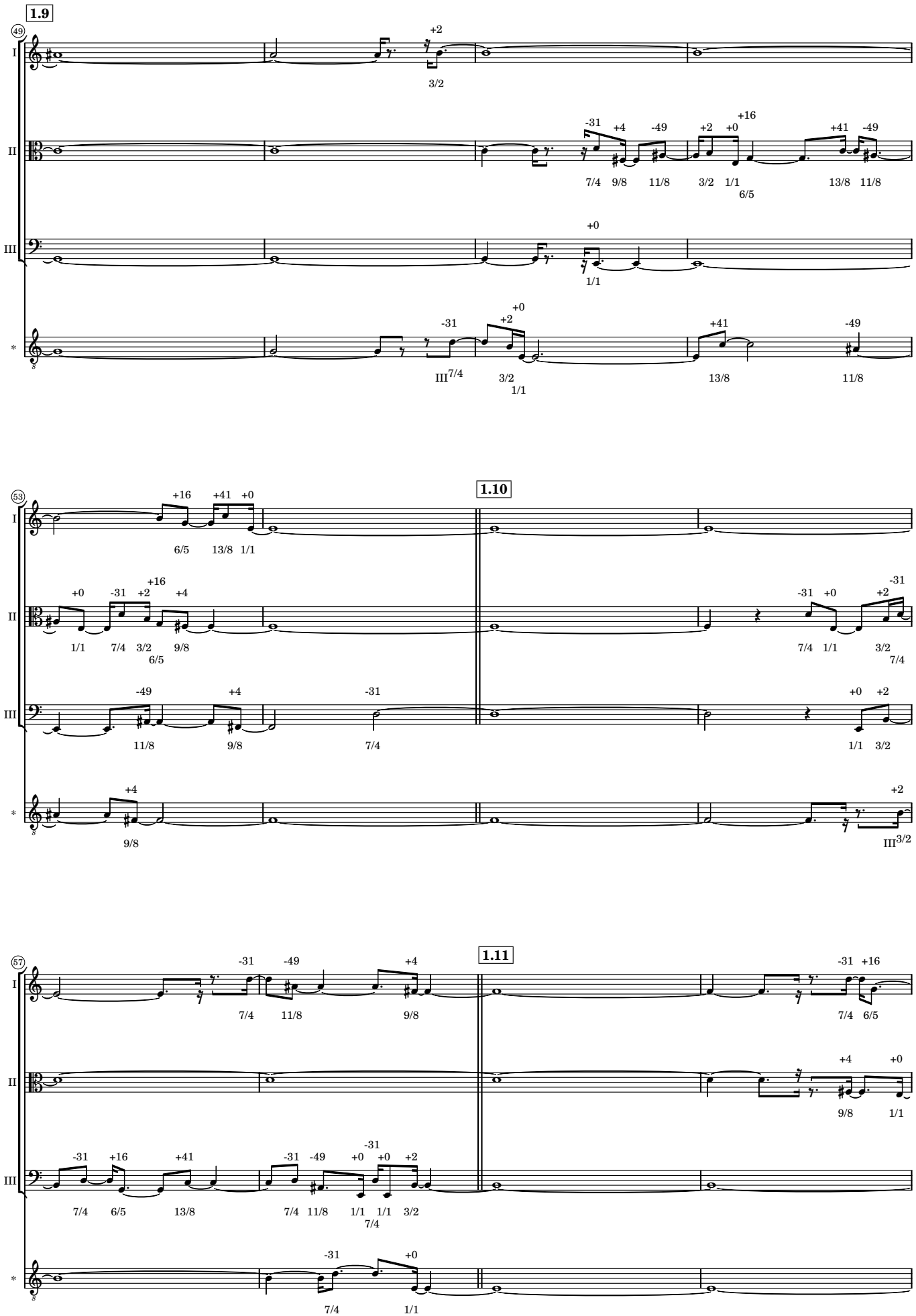

1.3
13
I
II
III
*
+4
9/8
+2
3/2
+0
1/1
+0
1/1
-31
7/4
+2 +0
+4 +41
9/8 13/8
3/2 1/1
+2
3/2
-31
7/4
+41
13/8
+16
6/5
-49 +16
11/8 6/5
1/1
-49
+16
6/5
-31
7/4
+41 +0
13/8 1/1
11/8
+2
3/2
+4
9/8
-31
III7/4
+0
1/1

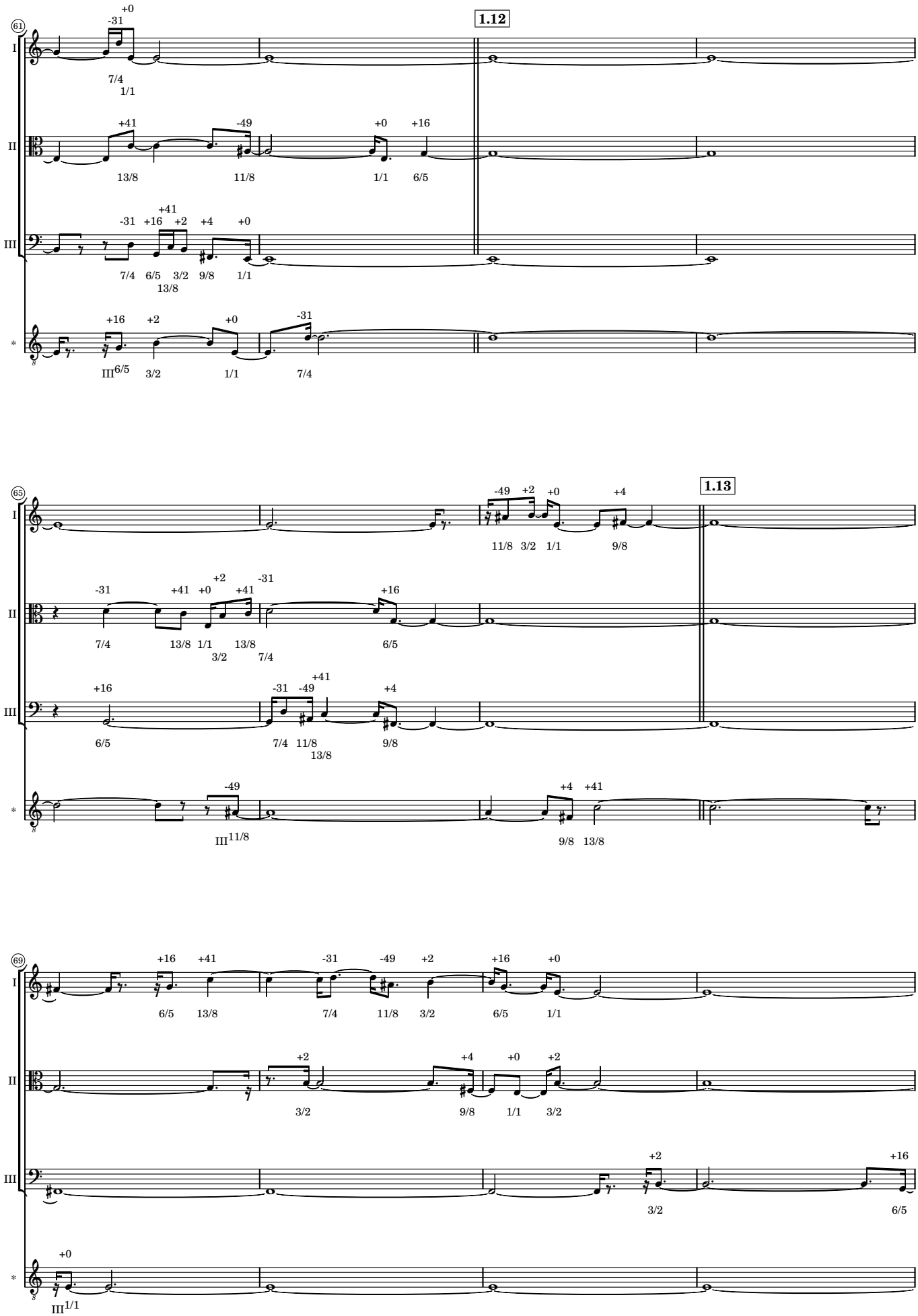

1.4
17
I
II
III
*
+41
13/8
+4
9/8
-49
11/8

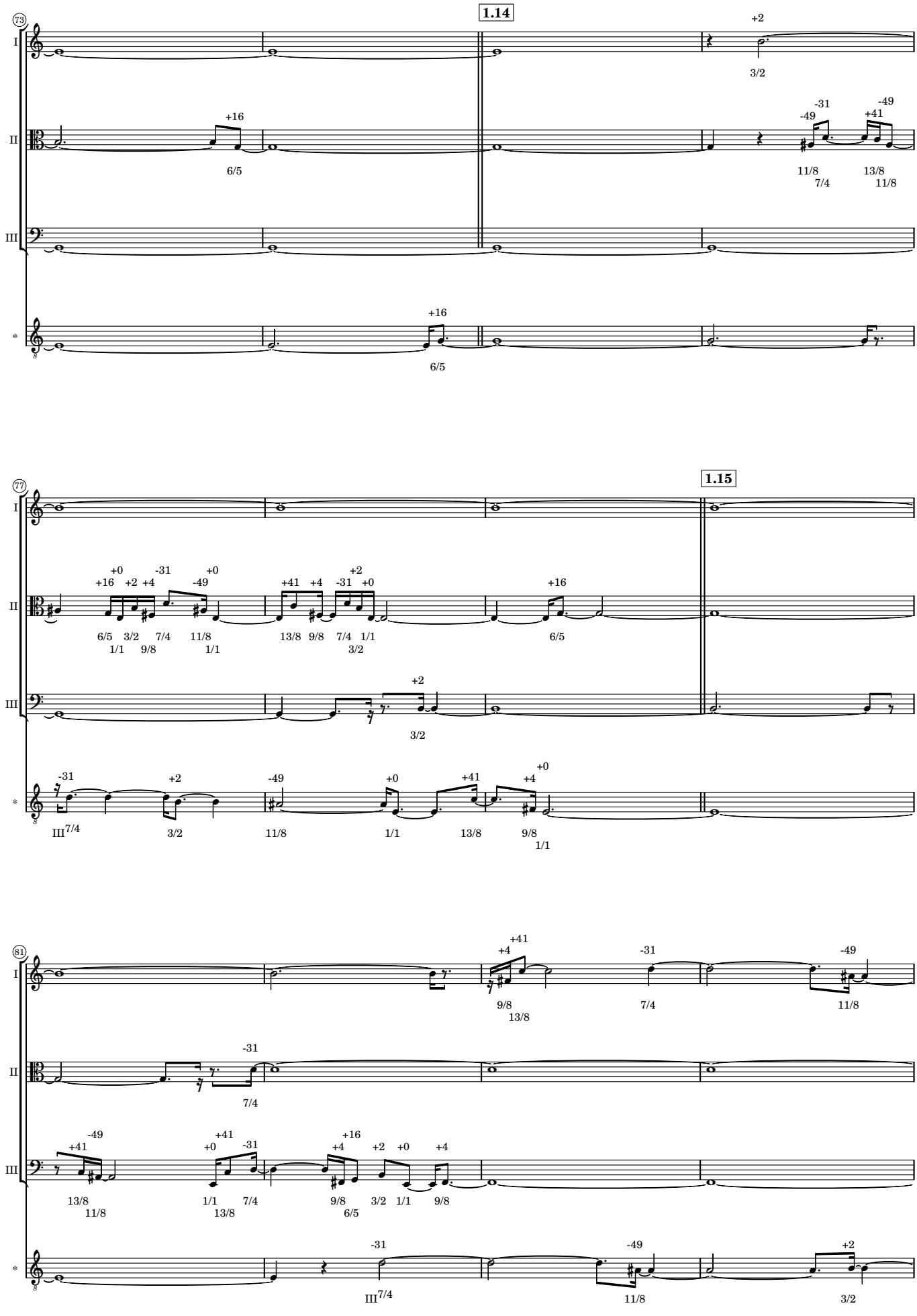

1.5
21
I
II
III
*
-49
11/8
+16
6/5
-31
7/4
+2
3/2
-49
11/8
+0
1/1
-31
7/4
3/2
+2
1/1
-49
11/8
+4
9/8
-31
+41
7/4
13/8
+4
9/8
+0
1/1
3/2
+2
+16
6/5
+0
1/1
+41
III13/8
+2
3/2
-31
7/4

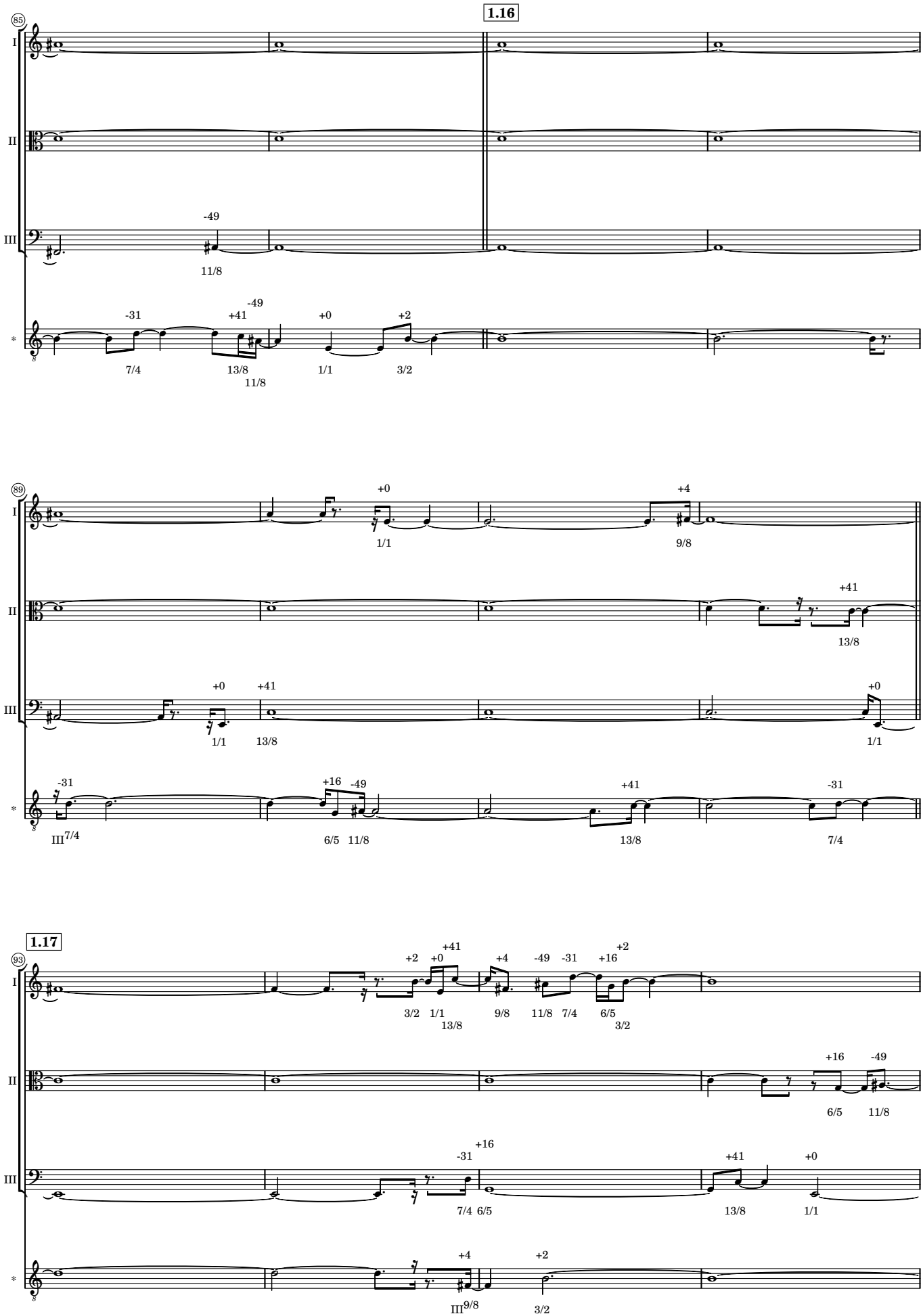

25
I
II
III
*
+41 +2 +0 +4
13/8 3/2 1/1 9/8
+16 +2 -31 +0 +41 -31 +16 -49
6/5 3/2 7/4 1/1 13/8 7/4 6/5 11/8
1/1
-49 +0 -31
11/8 1/1 7/4
-49 +41 +16
III11/8 13/8 6/5
29
1.6
+0 +16 +0 +2 +0 -49 -31 +0
1/1 6/5 1/1 3/2 1/1 11/8 7/4 1/1
+41 +0
13/8 1/1
+4 +2
9/8 III3/2
33
1.7
+4 +41 +41 -31 +16
13/8 9/8 13/8 7/4 6/5
+41 +0
13/8 1/1
+2 +16
3/2 6/5
-49 -31 +0
11/8 7/4 III1/1

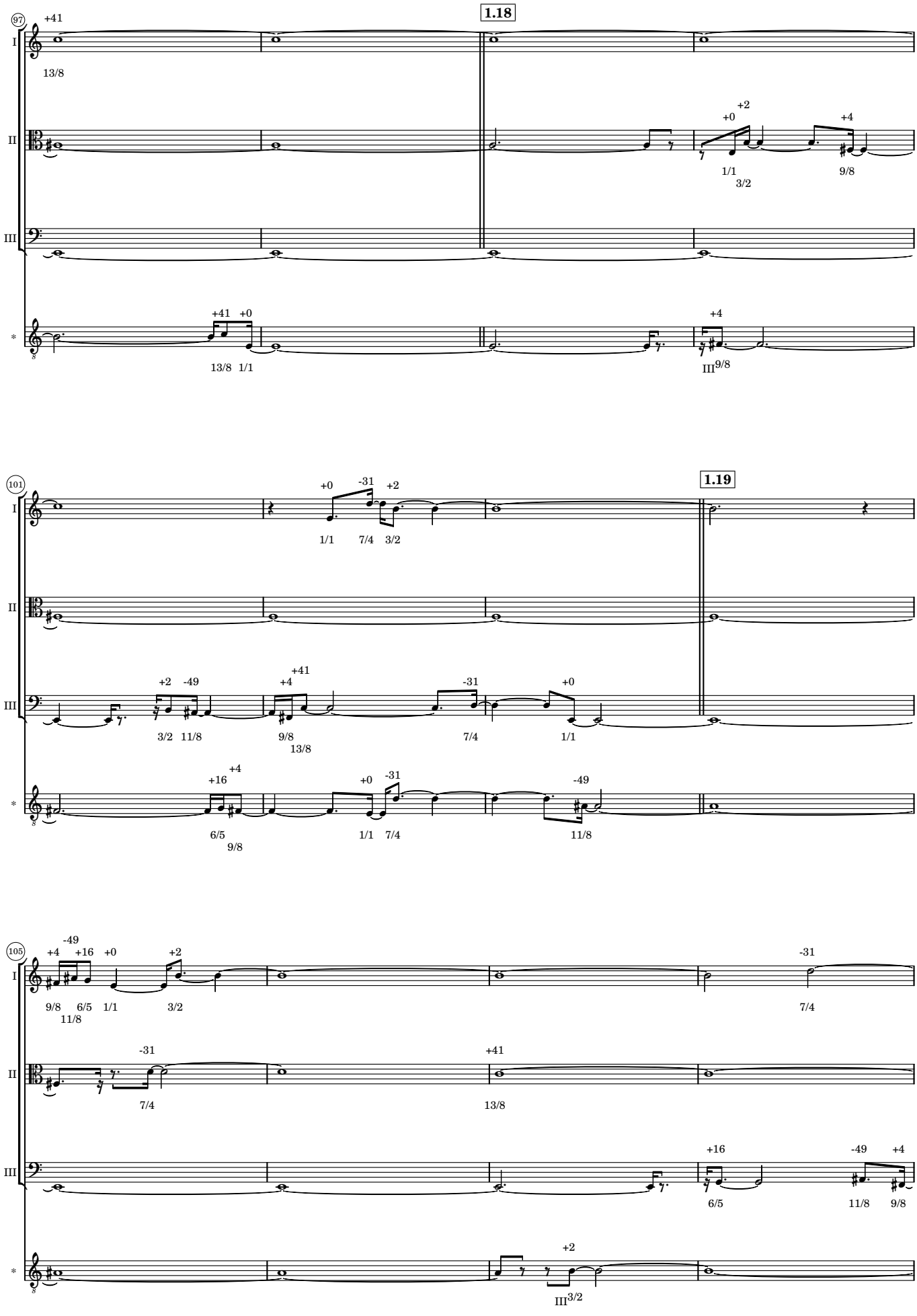

37
I
II
III
*
+0
+2
-31
3/2
1/1
7/4
+4
-31
9/8
7/4
+2
+0
-49
3/2
1/1
11/8
+4
+41
9/8
13/8
41
1.8
+2
-49
+2
3/2
11/8
3/2
-31
7/4
+2
III3/2
-31
+4
7/4
9/8
45
-49
+16
+41
+4
+0
11/8
6/5
13/8
9/8
1/1
+2
-31
3/2
7/4
+4
+2
+41
-49
9/8
3/2
13/8
11/8
+0
+41
1/1
13/8
+2
+41
+16
3/2
13/8
6/5
+16
-49
+2
+16
6/5
11/8
3/2
6/5

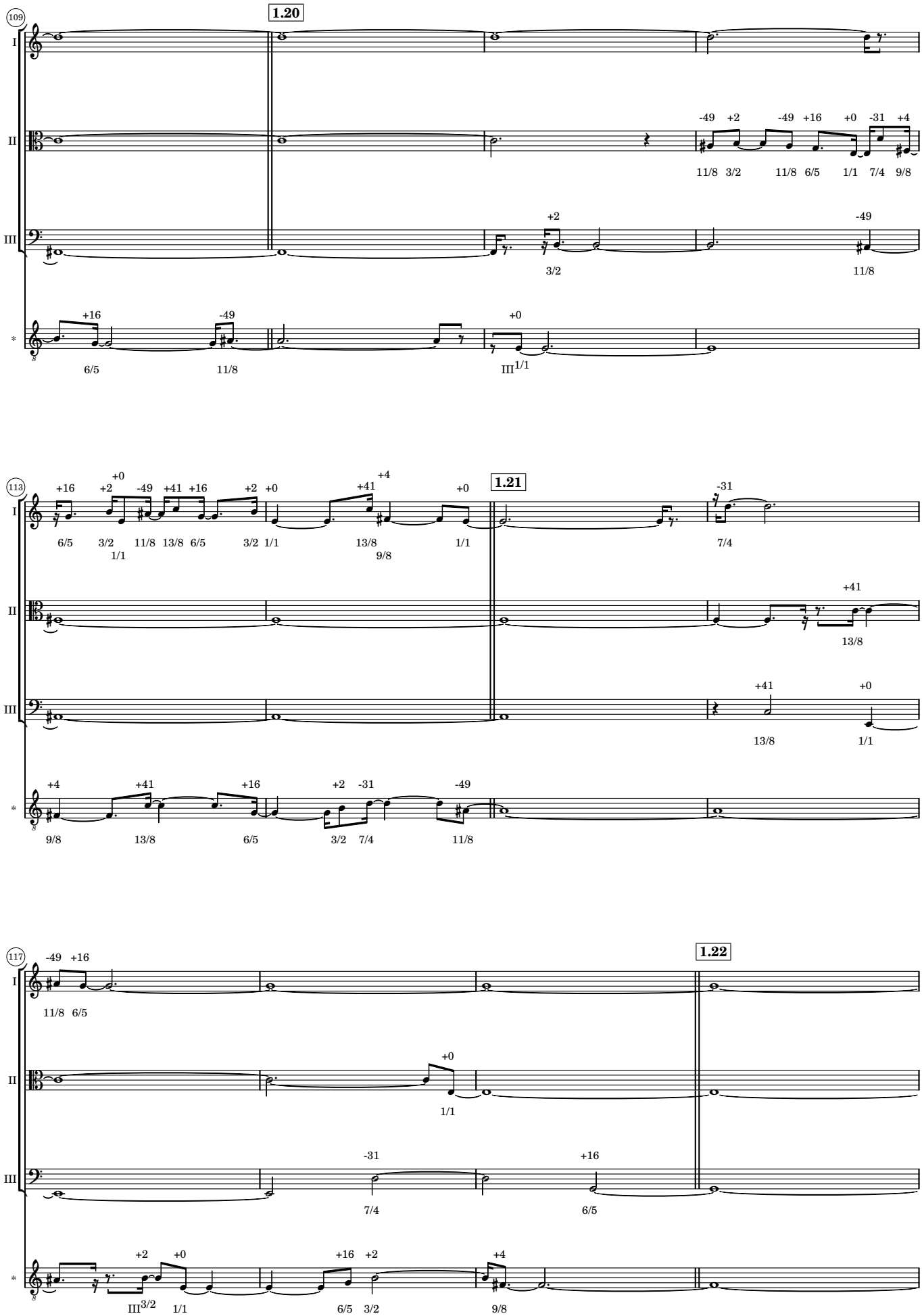

1.9
1.10
1.11
I
II
III
*

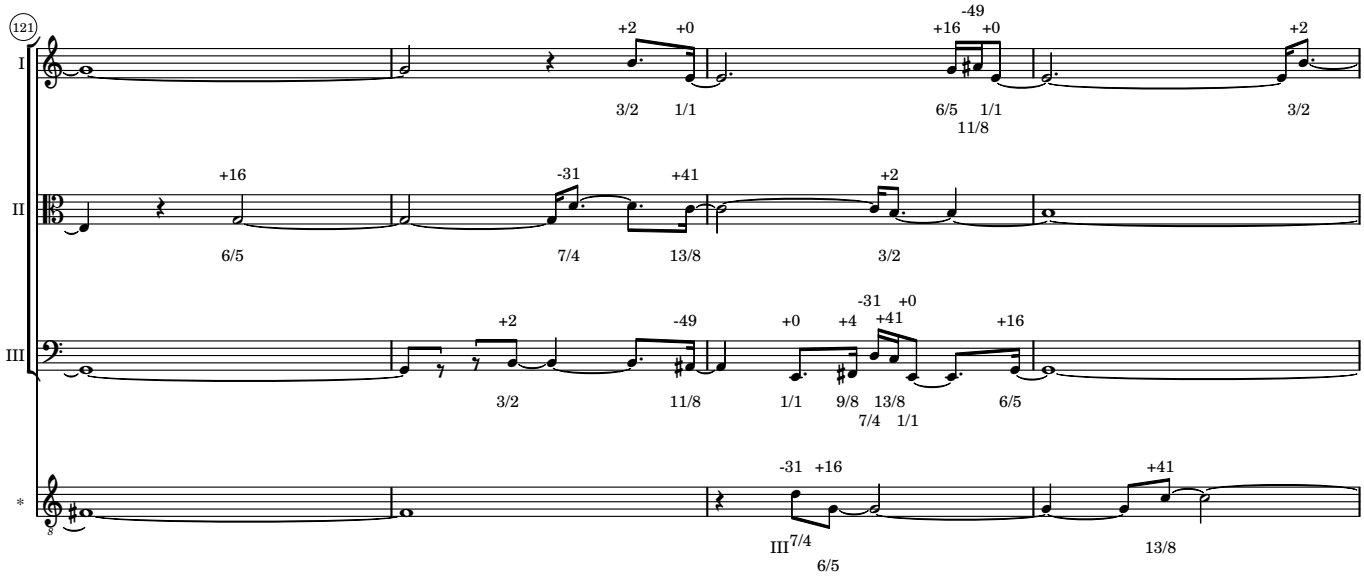

1.12
1.13
I
II
III
*

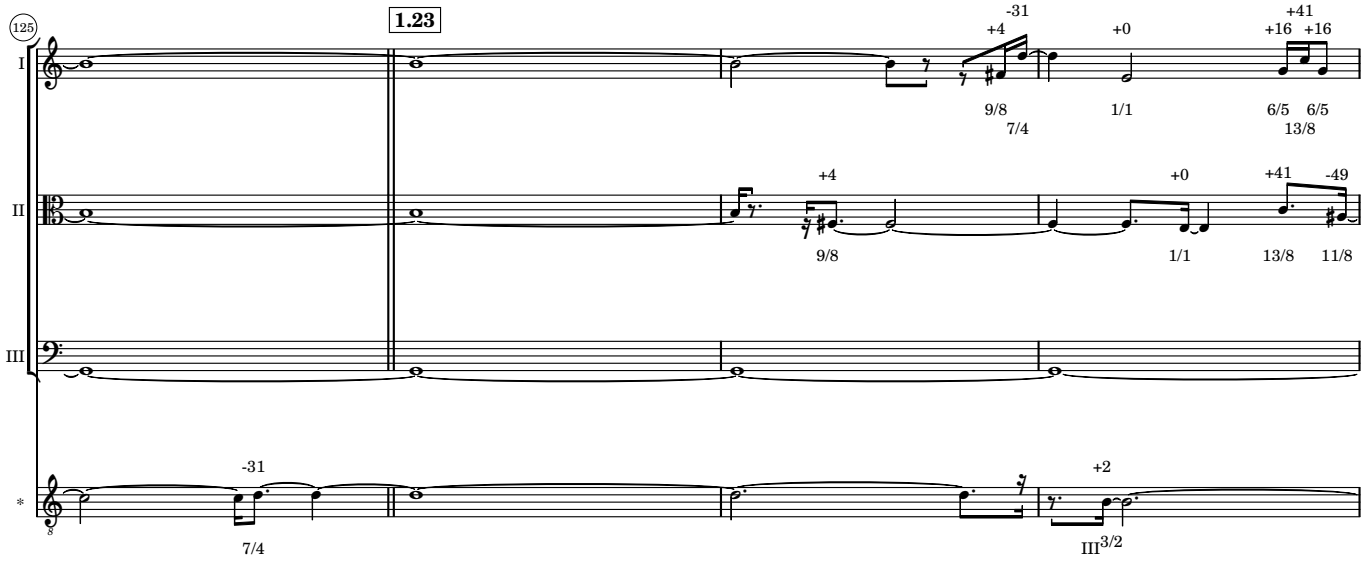

1.14
1.15
I
II
III
*

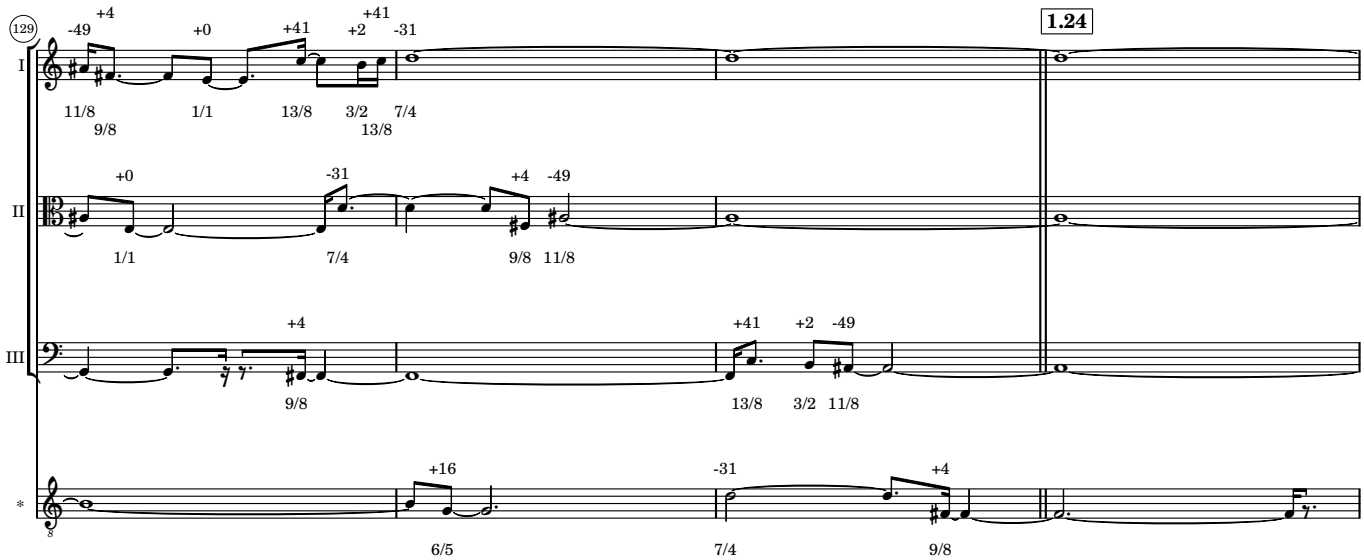

1.16
1.17
I
II
III
*

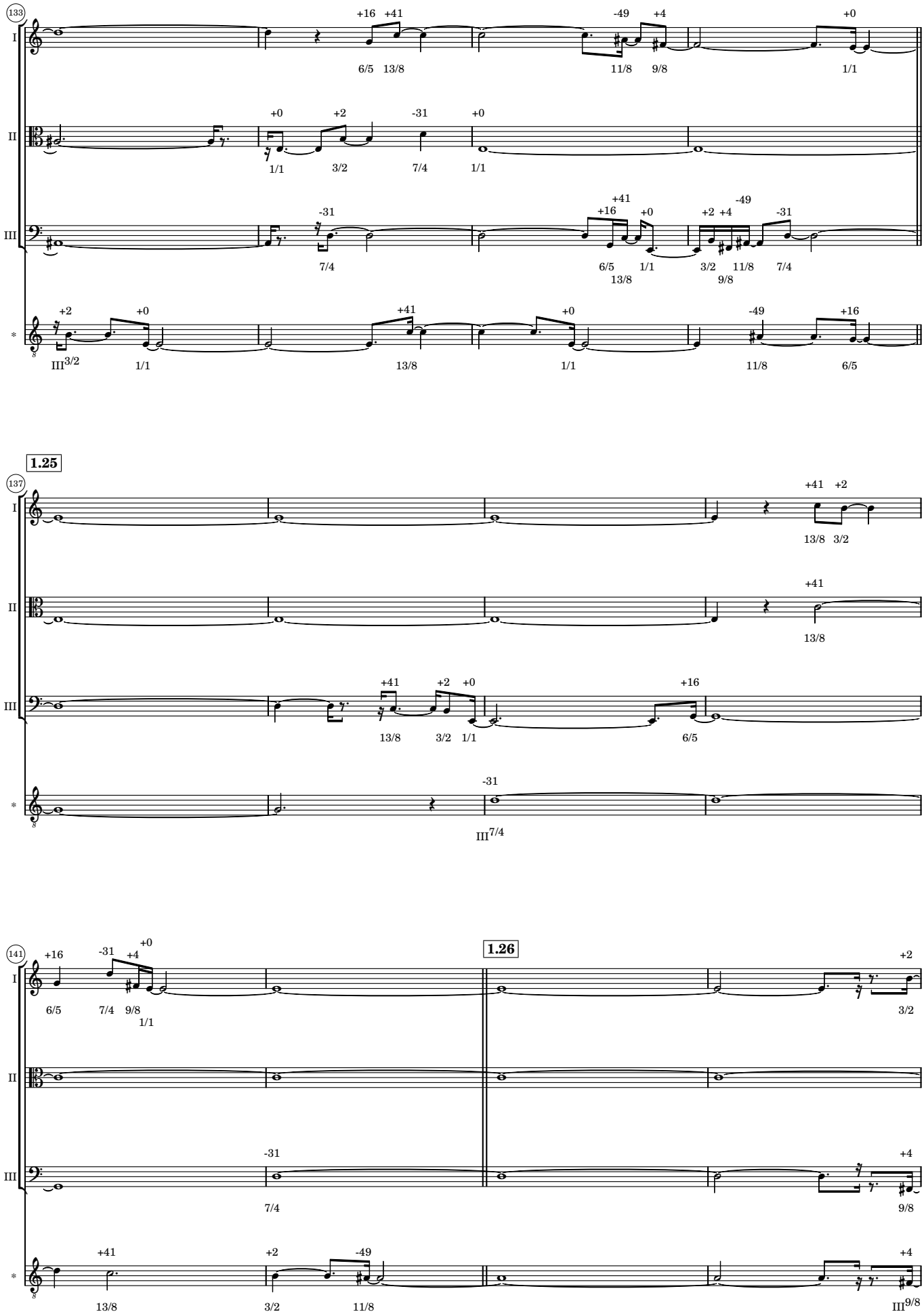

97
1.18
I
II
III
*
+41
13/8
+0
+2
+4
1/1
3/2
9/8
+41
+0
13/8
1/1
+4
III9/8
101
1.19
-31
1/1
7/4
3/2
+2
-49
3/2
11/8
+41
9/8
13/8
7/4
1/1
+16
6/5
9/8
11/8
105
-49
9/8
11/8
7/4
13/8
6/5
III3/2

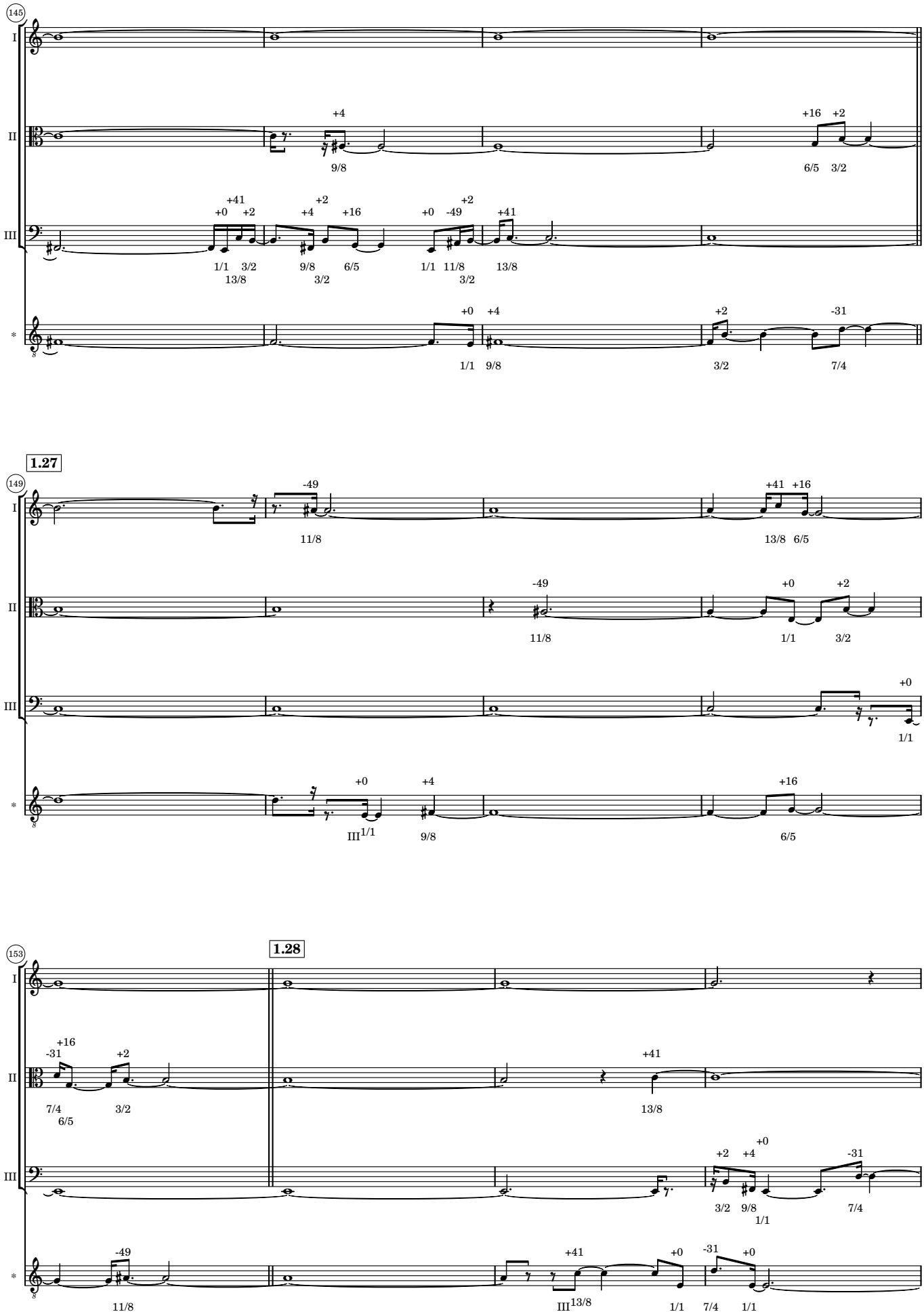

109
1.20
I
II
III
*
-49 +2 -49 +16 +0 -31 +4
11/8 3/2 11/8 6/5 1/1 7/4 9/8
+2 -49
3/2 11/8
+16 -49 +0
6/5 11/8 III1/1
113
1.21
+16 +2 +0 -49 +41 +16 +2 +0 +41 +4 +0 -31
6/5 3/2 1/1 11/8 13/8 6/5 3/2 1/1 13/8 9/8 1/1 7/4
+41
13/8
+41 +0
13/8 1/1
+4 +41 +16 +2 -31 -49
9/8 13/8 6/5 3/2 7/4 11/8
117
1.22
-49 +16
11/8 6/5
+0
1/1
-31 +16
7/4 6/5
+2 +0 +16 +2 +4
III3/2 1/1 6/5 3/2 9/8

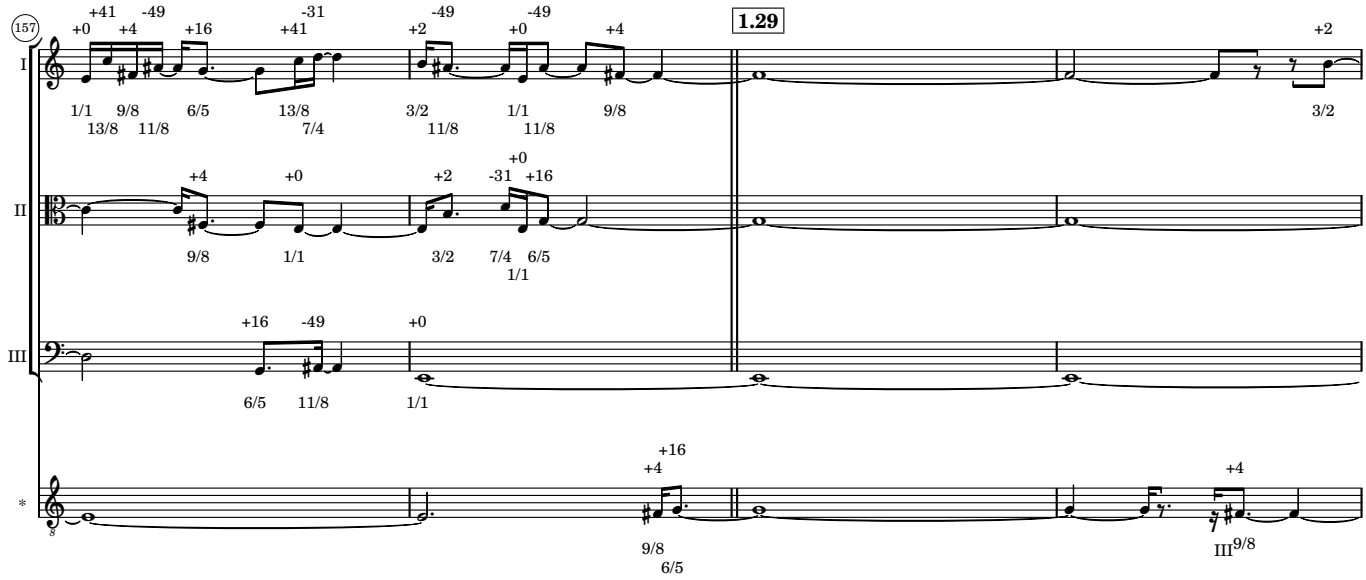

121
I
II
III
*
+2 +0 -49 +16 +0 +2
3/2 1/1 6/5 1/1 3/2
11/8
+16 -31 +41 +2
6/5 7/4 13/8 3/2
+2 -49 +0 +4 -31 +0 +41 +16
3/2 11/8 1/1 9/8 13/8 6/5
7/4 1/1
-31 +16 +41
III7/4 13/8
6/5

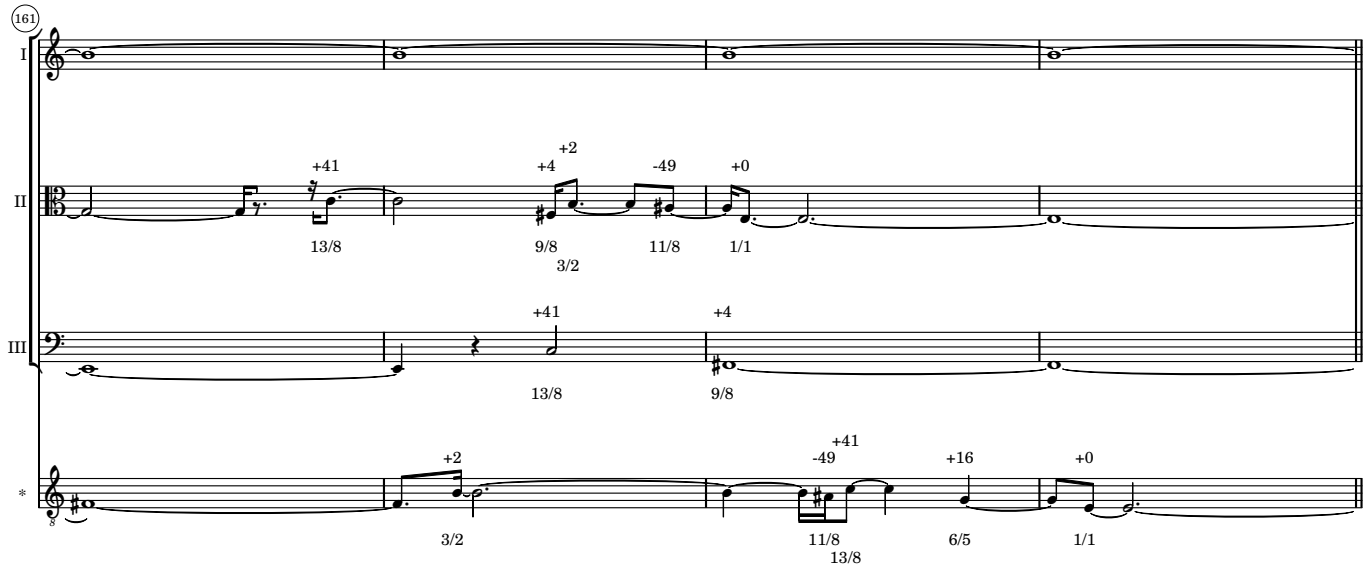

125
1.23
I
II
III
*
-31 +4 +0 +41 +16 +16
9/8 1/1 6/5 6/5
7/4 13/8
+4 +0 +41 -49
9/8 1/1 13/8 11/8
-31 +2
7/4 III3/2

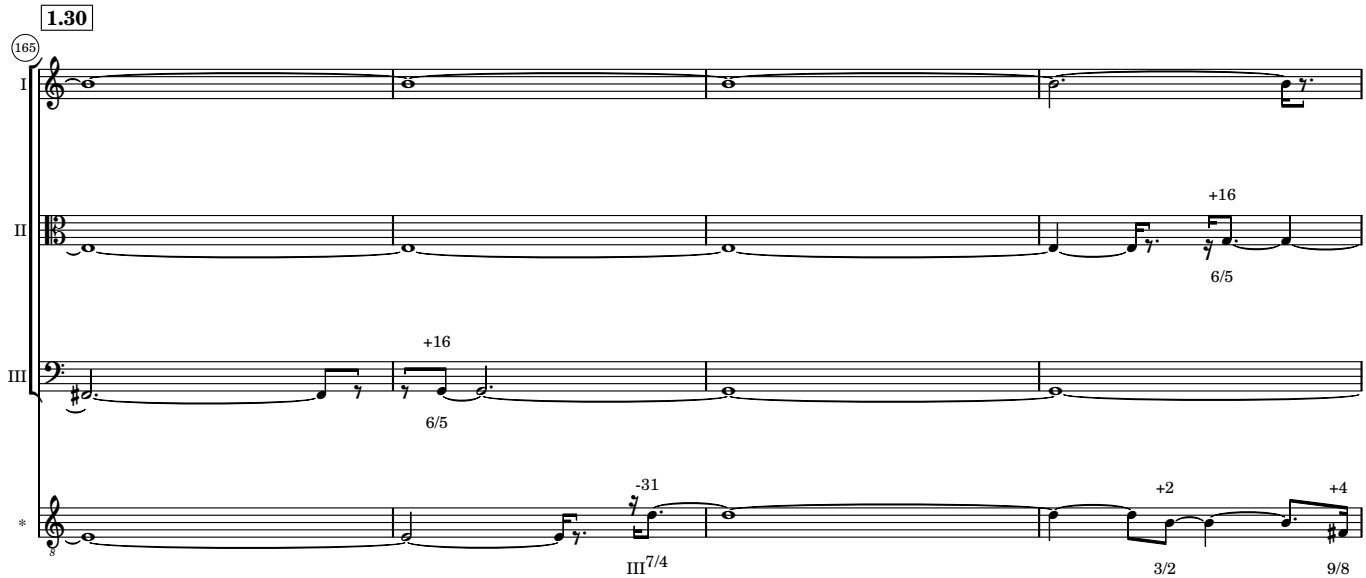

129
1.24
I
II
III
*
-49 +4 +0 +41 +2 +41 -31
11/8 9/8 1/1 13/8 3/2 13/8 7/4
+0 -31 +4 -49
1/1 7/4 9/8 11/8
+4 +41 +2 -49
9/8 13/8 3/2 11/8
+16 -31 +4
6/5 7/4 9/8

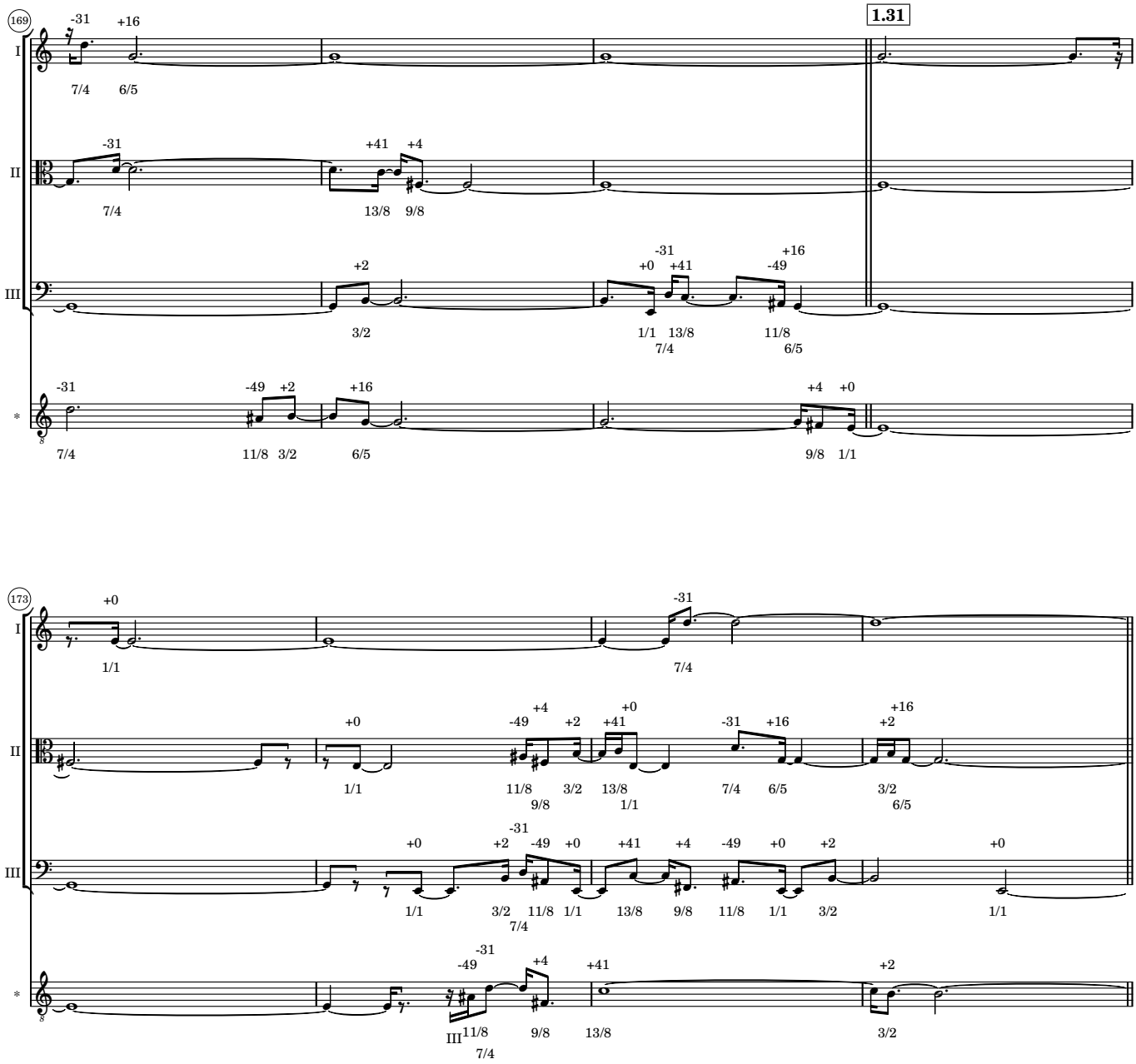

133
I
II
III
*
+16 +41
6/5 13/8
-49 +4
11/8 9/8
+0
1/1
+0 +2 -31 +0
1/1 3/2 7/4 1/1
-31
7/4
+41
+16 +0
6/5 1/1
13/8
+2 +4 -49 -31
3/2 11/8 7/4
9/8
+2 +0
III3/2 1/1
+41
13/8
+0
1/1
-49 +16
11/8 6/5
1.25
137
+41 +2
13/8 3/2
+41
13/8
+41 +2 +0
13/8 3/2 1/1
+16
6/5
-31
III7/4
141
+16 -31 +4 +0
6/5 7/4 9/8 1/1
1.26
+2
3/2
-31
7/4
+4
9/8
+41 +2 -49
13/8 3/2 11/8
+4
III9/8


145
I
II
III
*
+4
9/8
+16 +2
6/5 3/2
+41
+0 +2
+2
+4 +16
+2
+0 -49
+41
1/1 3/2
13/8
9/8 6/5
3/2
1/1 11/8
3/2
13/8
+0 +4
+2
-31
1/1 9/8
3/2
7/4
1.27
149
-49
11/8
+41 +16
13/8 6/5
-49
+0 +2
11/8
1/1 3/2
+0
1/1
+0 +4
III 1/1
9/8
+16
6/5
1.28
153
+16
-31 +2
7/4 3/2
6/5
+41
13/8
+2 +4 +0
-31
3/2 9/8 7/4
1/1
-49
11/8
+41 +0
III 13/8
1/1
-31 +0
7/4 1/1

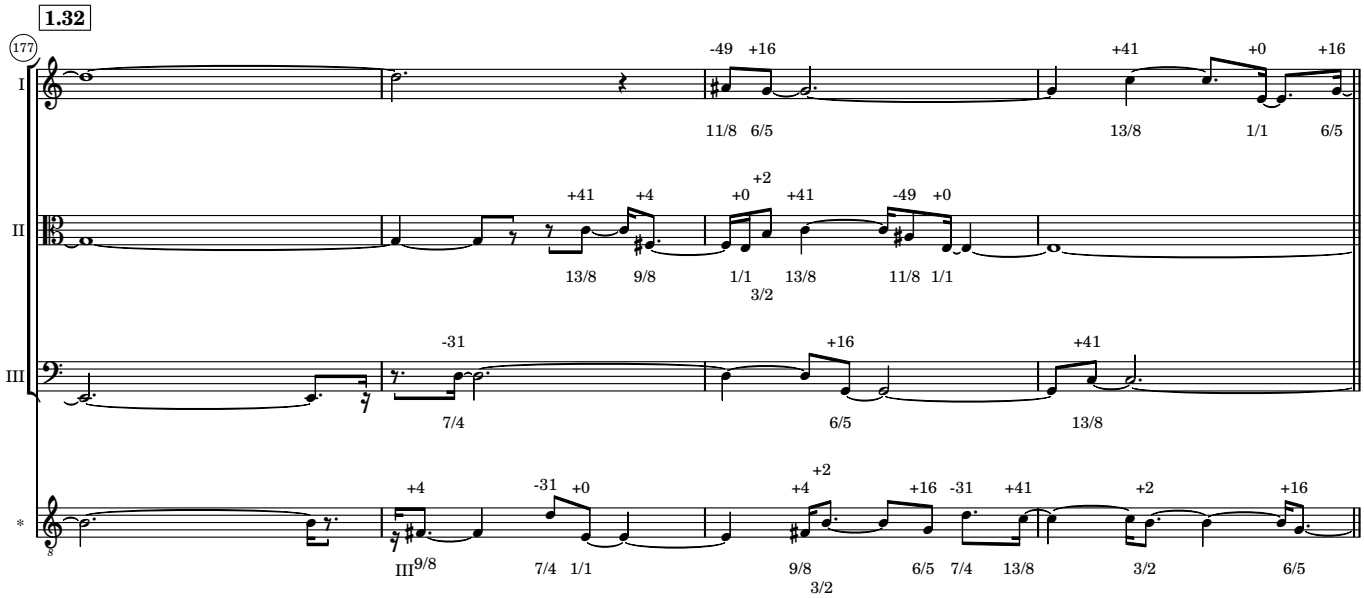

157
I
II
III
*
1.29
+41 -49 +0 +4 +16 -31 +41 -49 +2 +0 +4 +2
1/1 9/8 6/5 13/8 3/2 1/1 9/8 3/2
13/8 11/8 7/4 11/8 11/8
+4 +0 +2 +0 -31 +16
9/8 1/1 3/2 7/4 6/5
1/1
+16 -49 +0
6/5 11/8 1/1
+4 +16 +4
9/8 III9/8
6/5

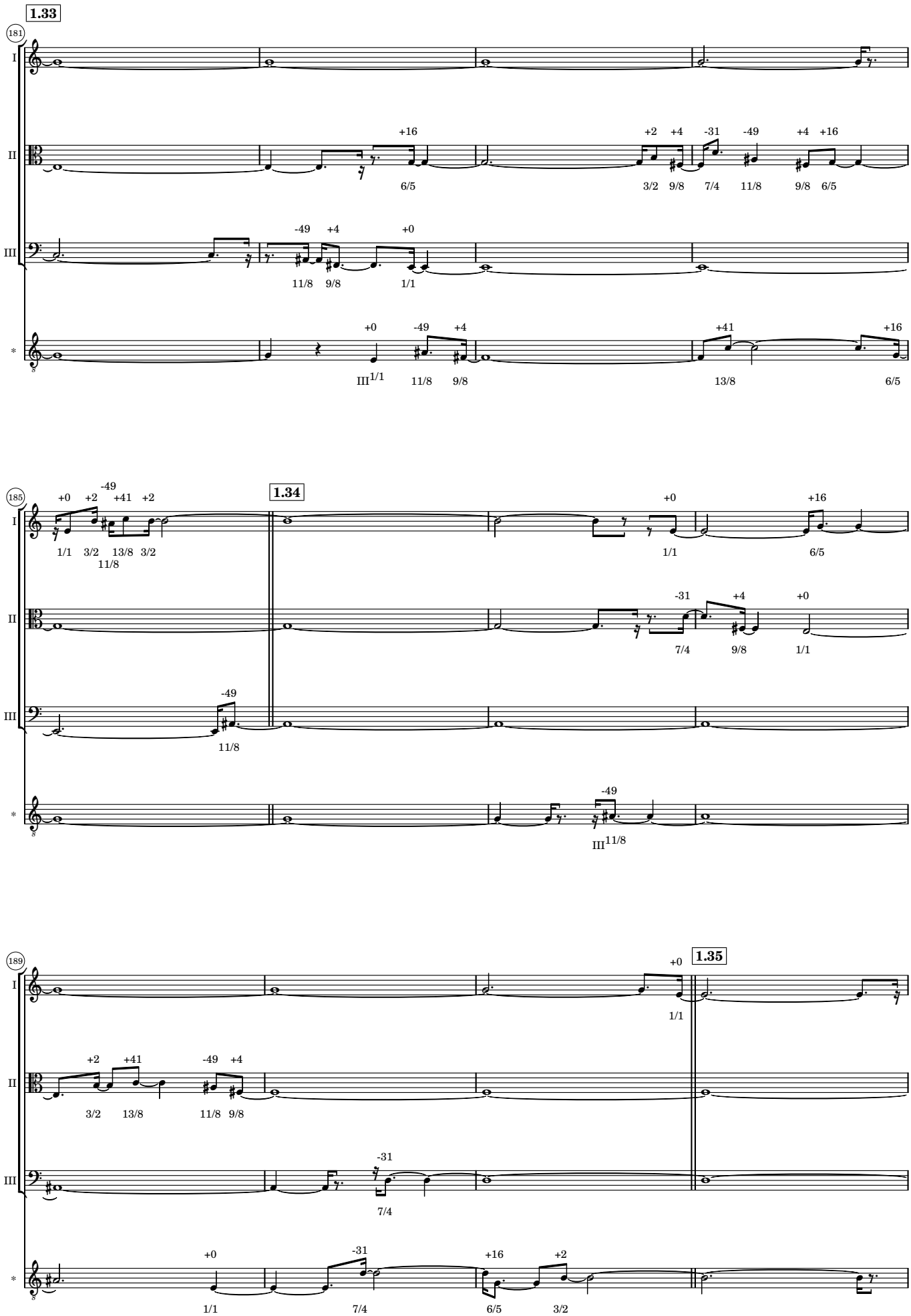

161
I
II
III
*
+41 +2 +4 -49 +0
13/8 9/8 11/8 1/1
3/2
+41 +4
13/8 9/8
+2 +41 -49 +16 +0
3/2 11/8 6/5 1/1
13/8

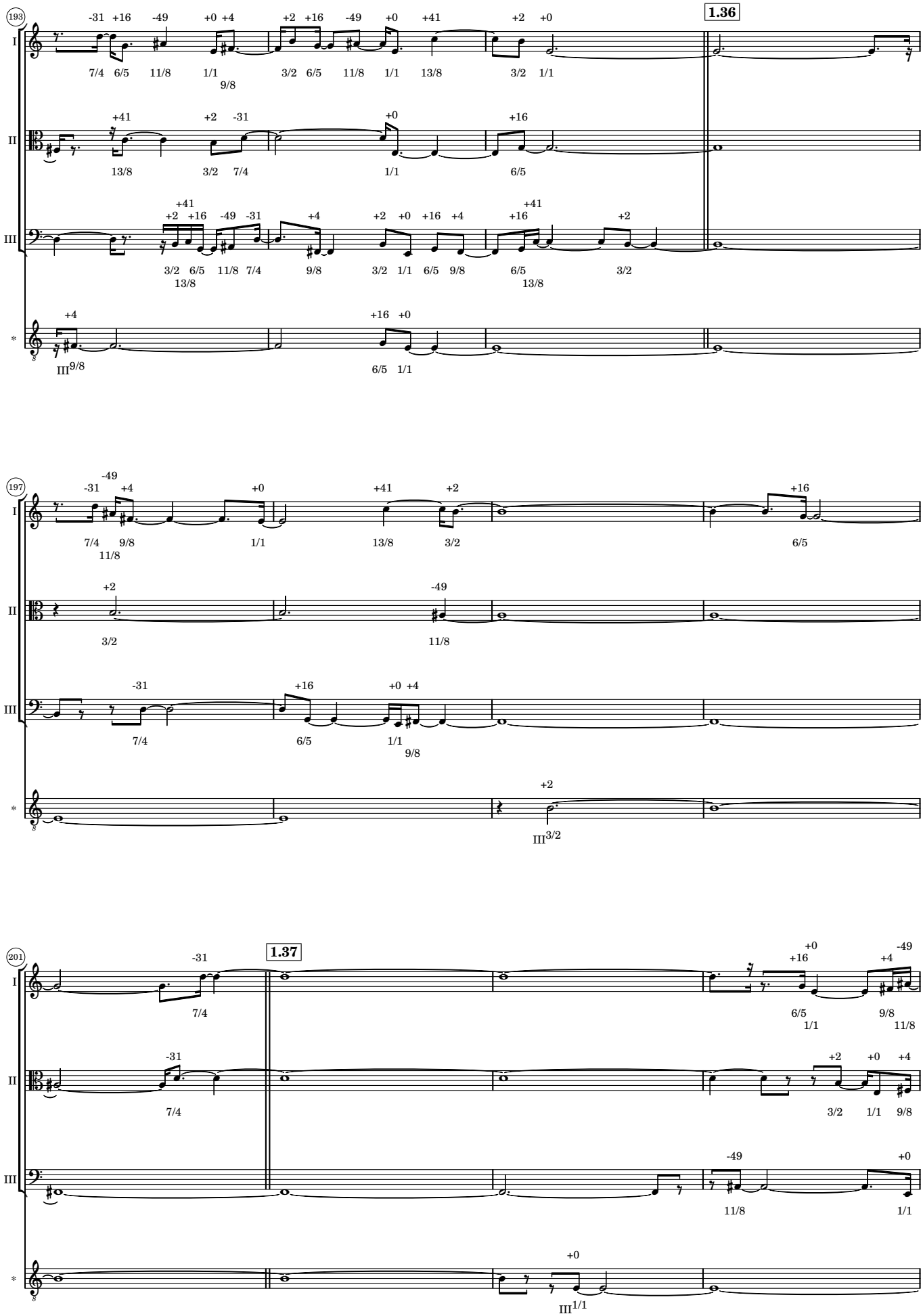

1.30
165
I
II
III
*
+16
6/5
+16
6/5
-31 +2 +4
III7/4 3/2 9/8

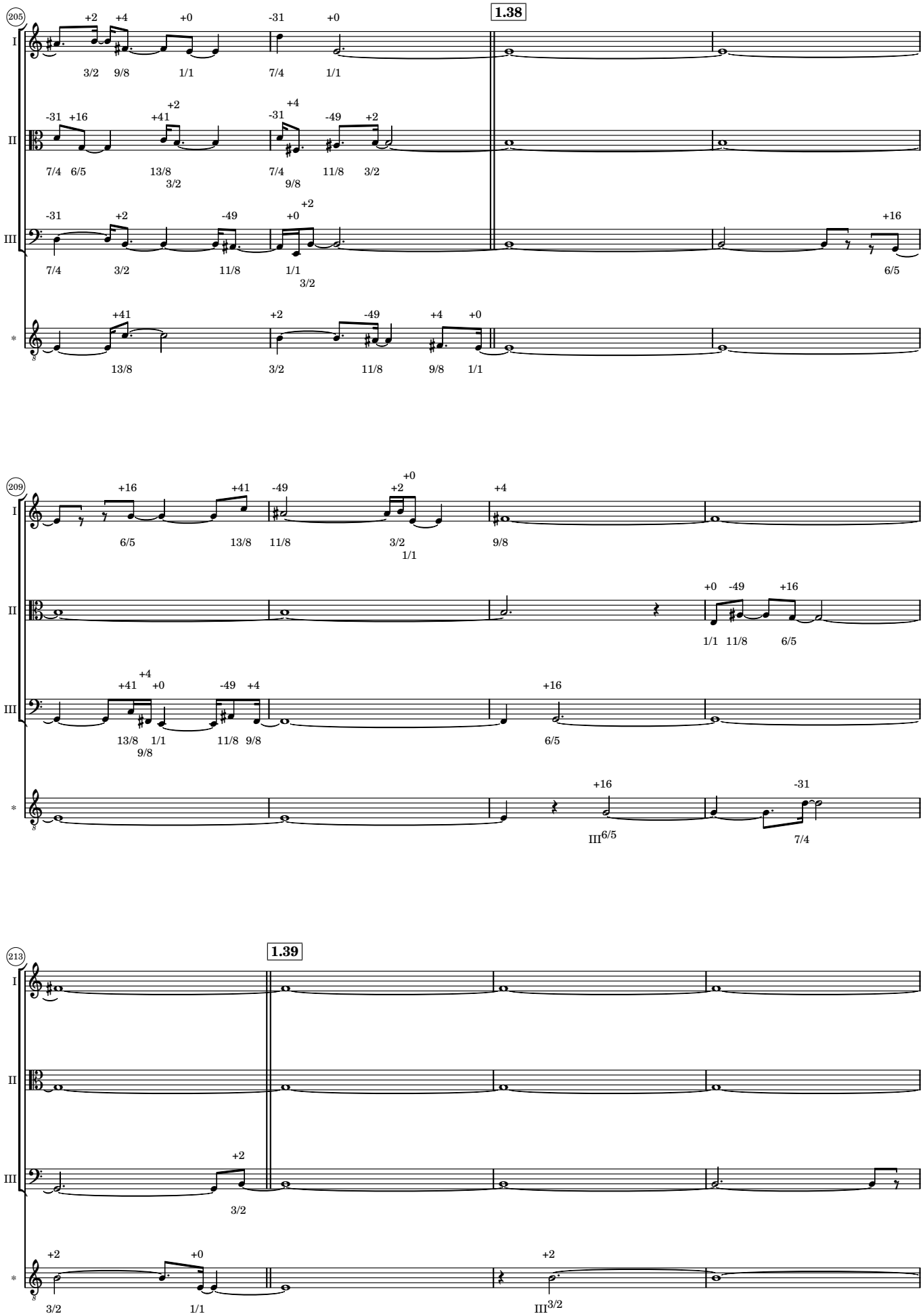

169
1.31
I
II
III
*
-31 +16
7/4 6/5
-31 +41 +4
7/4 13/8 9/8
+2 -31 +0 +41 +16 -49
3/2 1/1 13/8 11/8
7/4 6/5
-31 -49 +2 +16 +4 +0
7/4 11/8 3/2 6/5 9/8 1/1

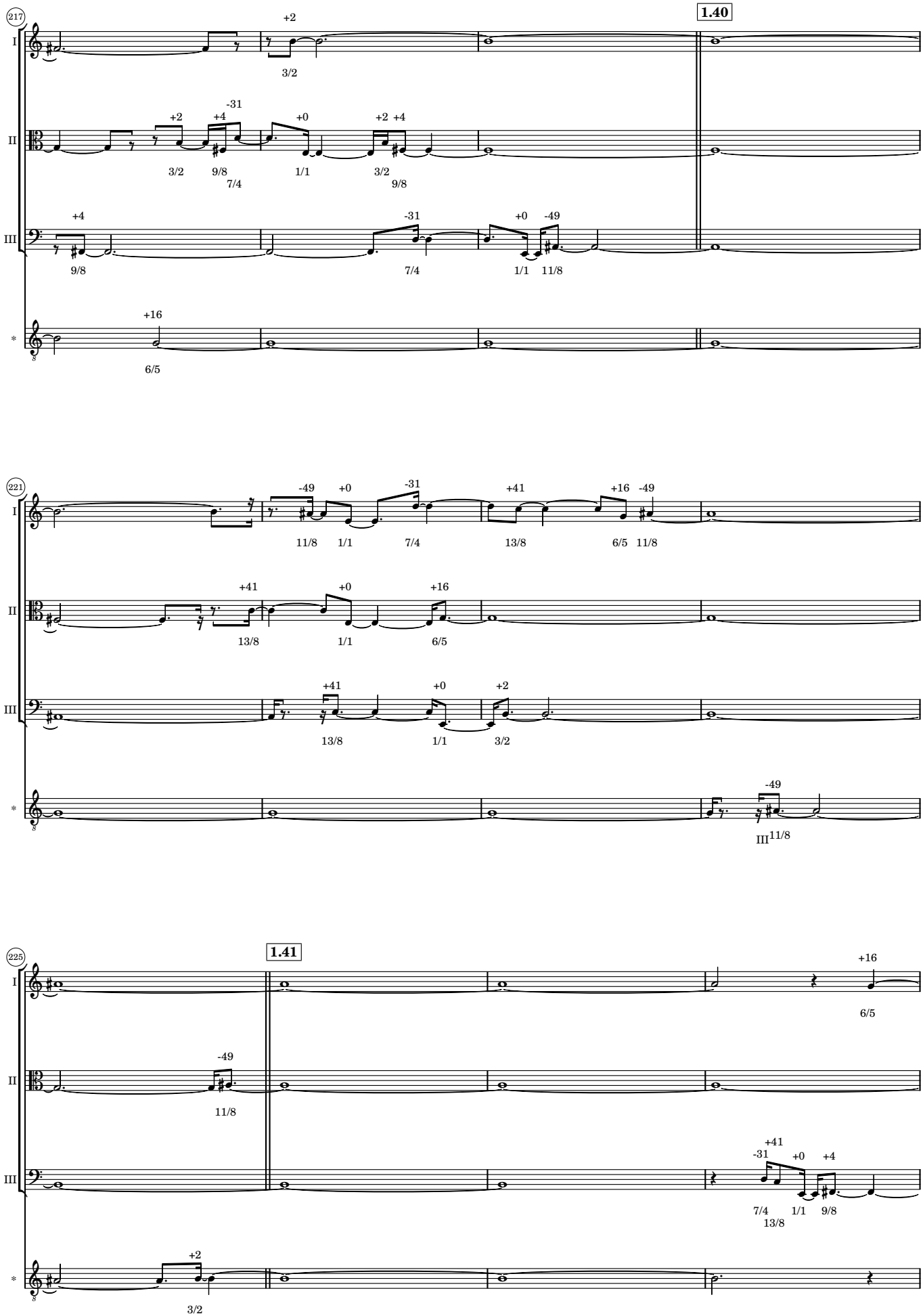
173
I
II
III
*
+0 -31
1/1 7/4
+0 -49 +4 +2 +41 +0 -31 +16 +2 +16
1/1 11/8 9/8 3/2 13/8 1/1 7/4 6/5 3/2 6/5
+0 +2 -31 -49 +0 +41 +4 -49 +0 +2 +0
1/1 3/2 11/8 1/1 7/4 13/8 9/8 11/8 1/1 3/2 1/1
-31 -49 +4 +41 +2
III 11/8 7/4 9/8 13/8 3/2

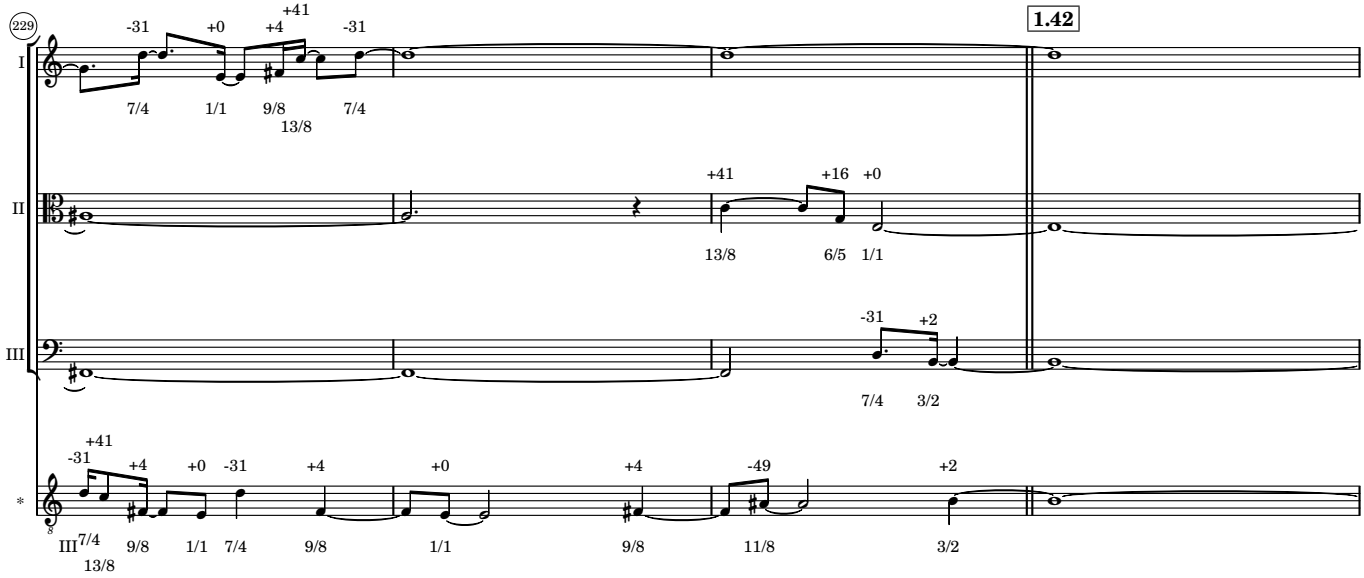

1.32
177
I
II
III
*
-49 +16 +41 +0 +16
11/8 6/5 13/8 1/1 6/5
+41 +4 +0 +2 +41 -49 +0
13/8 9/8 1/1 3/2 13/8 11/8 1/1
-31 +16 +41
7/4 6/5 13/8
+4 -31 +0 +2 +4 +16 -31 +41 +2 +16
III 9/8 7/4 1/1 9/8 3/2 6/5 7/4 13/8 3/2 6/5

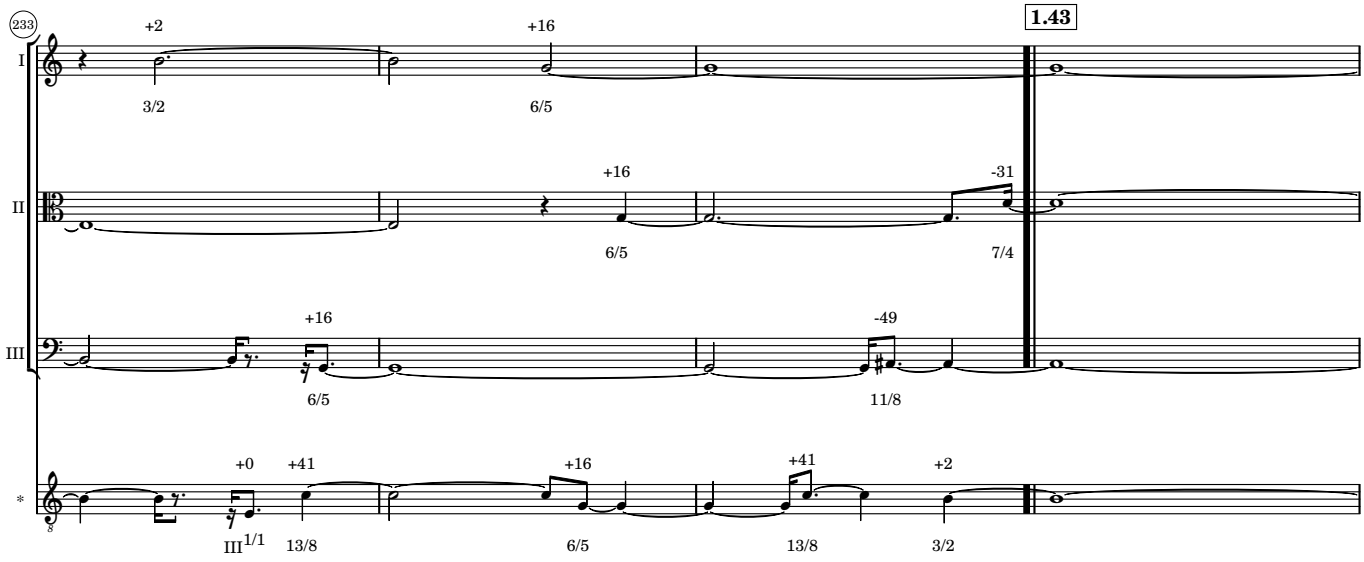

1.33
181
I
II
III
*
+16
6/5
+2 +4
3/2 9/8
-31
7/4
-49
11/8
+4 +16
9/8 6/5
-49 +4
11/8 9/8
+0
1/1
+0
III1/1
-49 +4
11/8 9/8
+41
13/8
+16
6/5
185
+0 +2 -49 +41 +2
1/1 3/2 13/8 3/2
11/8
1.34
+0
1/1
+16
6/5
-31 +4
7/4 9/8
+0
1/1
-49
11/8
-49
III11/8
189
+0
1/1
1.35
+2 +41
3/2 13/8
-49 +4
11/8 9/8
-31
7/4
+0
1/1
-31
7/4
+16
6/5
+2
3/2

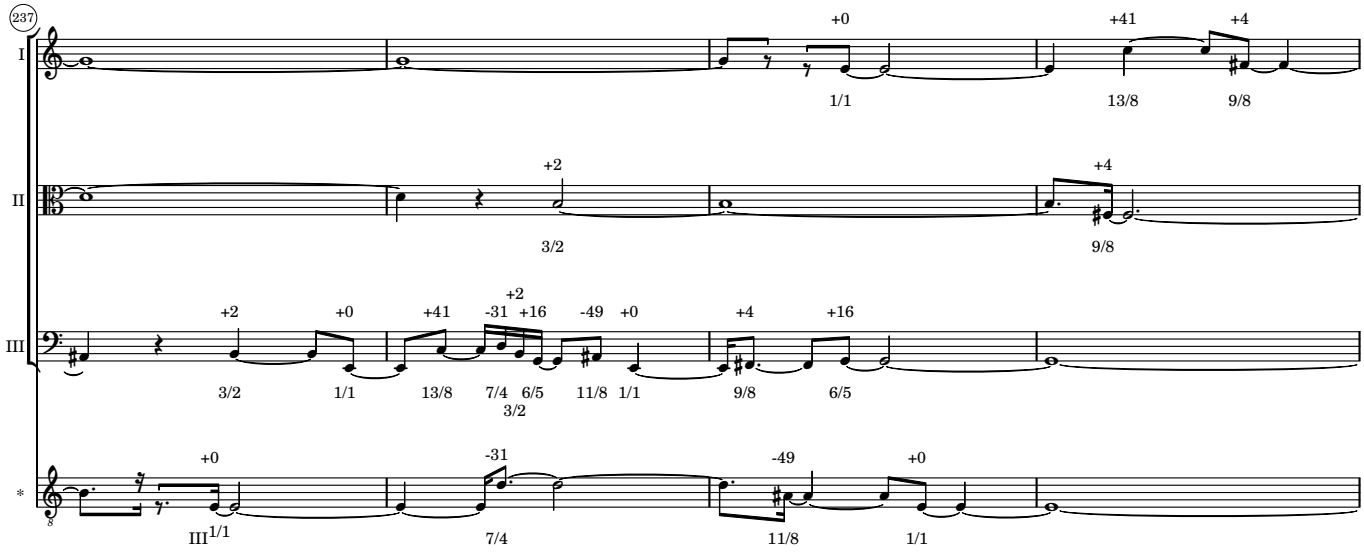

193
I
II
III
*
1.36
197
201
1.37

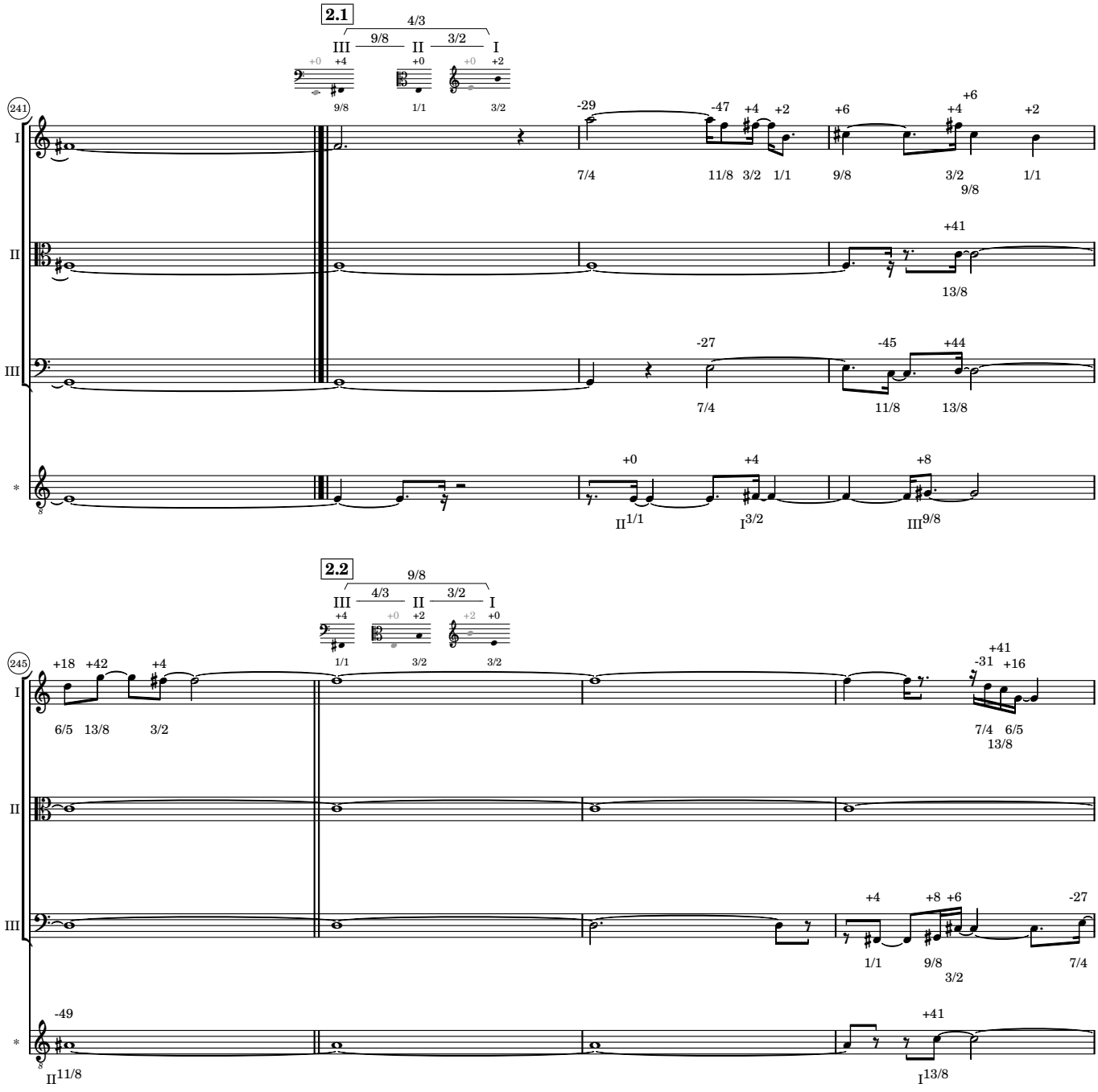

205
1.38
209
1.39
213
I
II
III
*

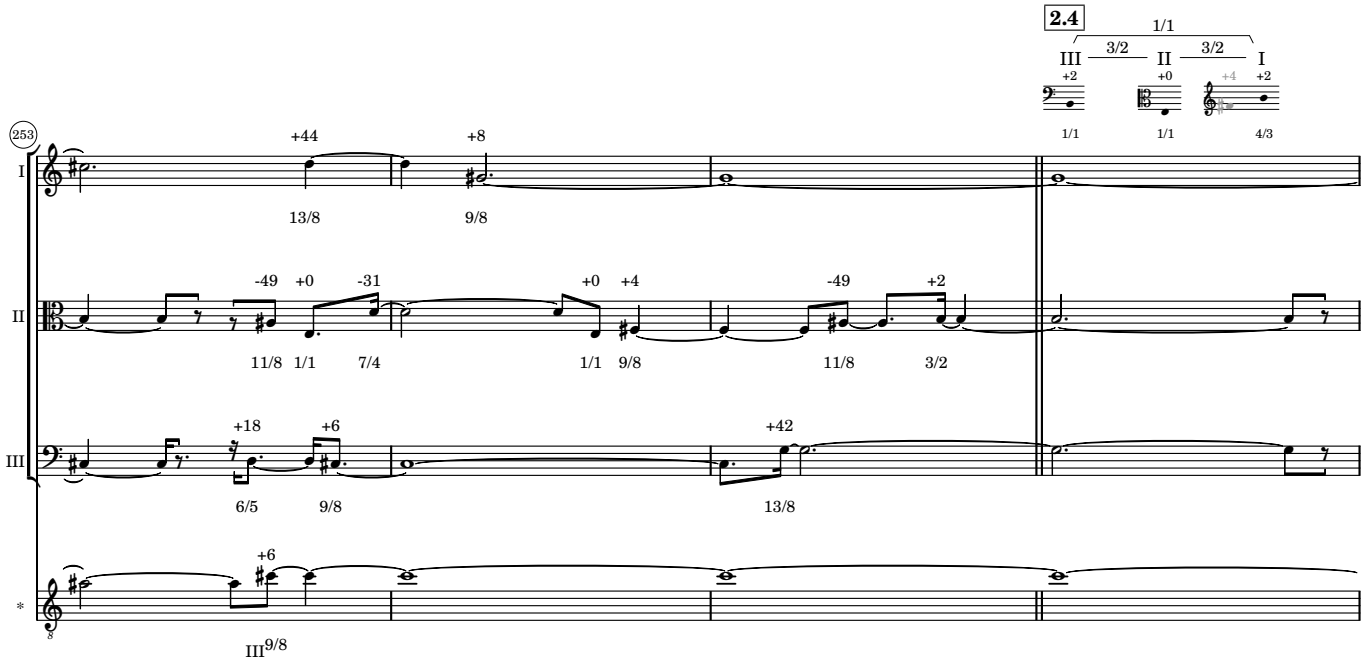

217
I
II
III
*
1.40
221
1.41
225

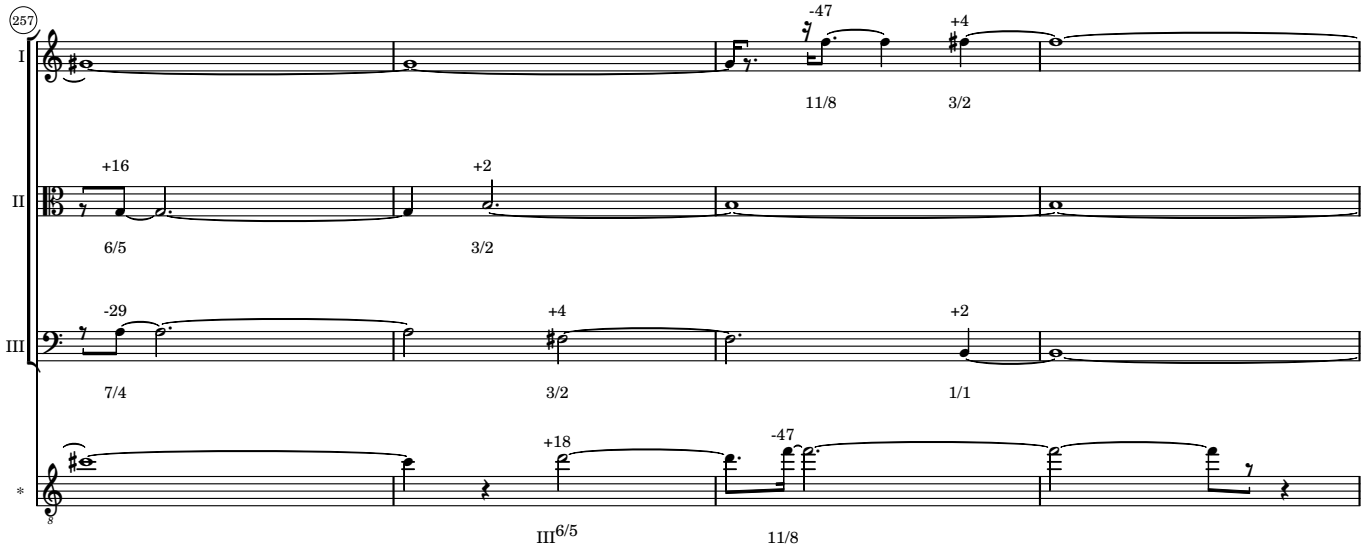

229
I
II
III
*
-31 +0 +4 +41 -31
7/4 1/1 9/8 13/8 7/4
1.42
+41 +16 +0
13/8 6/5 1/1
-31 +2
7/4 3/2
+41 -31 +4 +0 -31 +4 +0 +4 -49 +2
III7/4 13/8 9/8 1/1 7/4 9/8 1/1 9/8 11/8 3/2

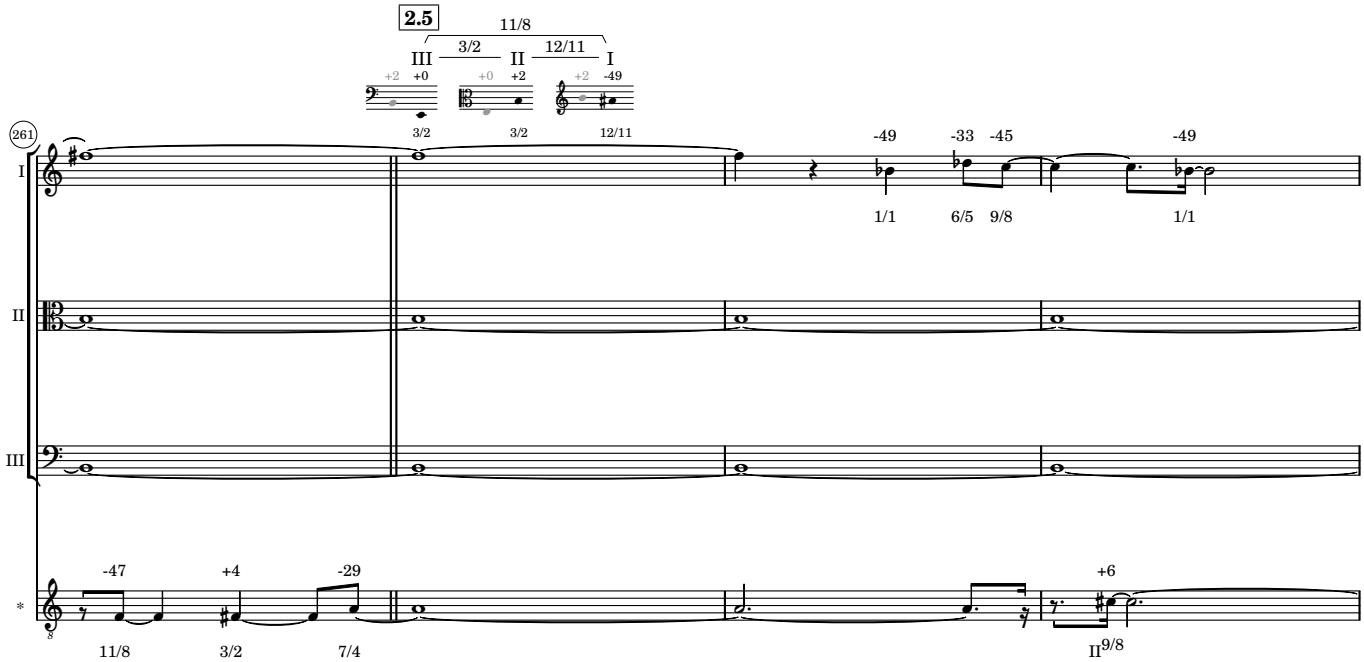

233
I
II
III
*
+2 +16
3/2 6/5
1.43
+16 -31
6/5 7/4
+16 -49
6/5 11/8
+0 +41 +16 +41 +2
III1/1 13/8 6/5 13/8 3/2

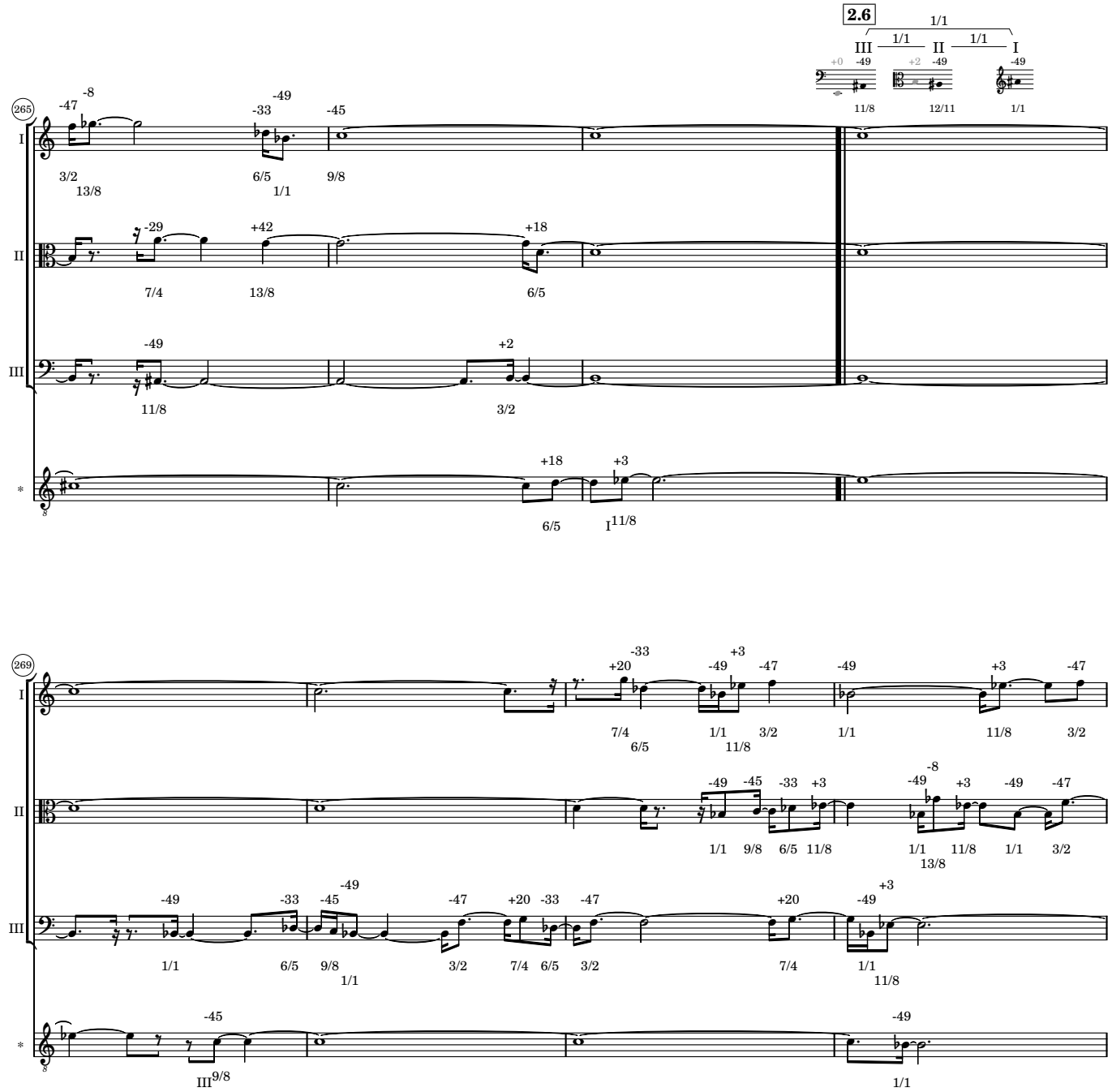

237
I
II
III
*
+0 +41 +4
1/1 13/8 9/8
+2 +4
3/2 9/8
+2 +0 +41 -31 +2 +16 -49 +0 +4 +16
3/2 1/1 13/8 7/4 3/2 6/5 11/8 1/1 9/8 6/5
+0 -31 -49 +0
III1/1 7/4 11/8 1/1

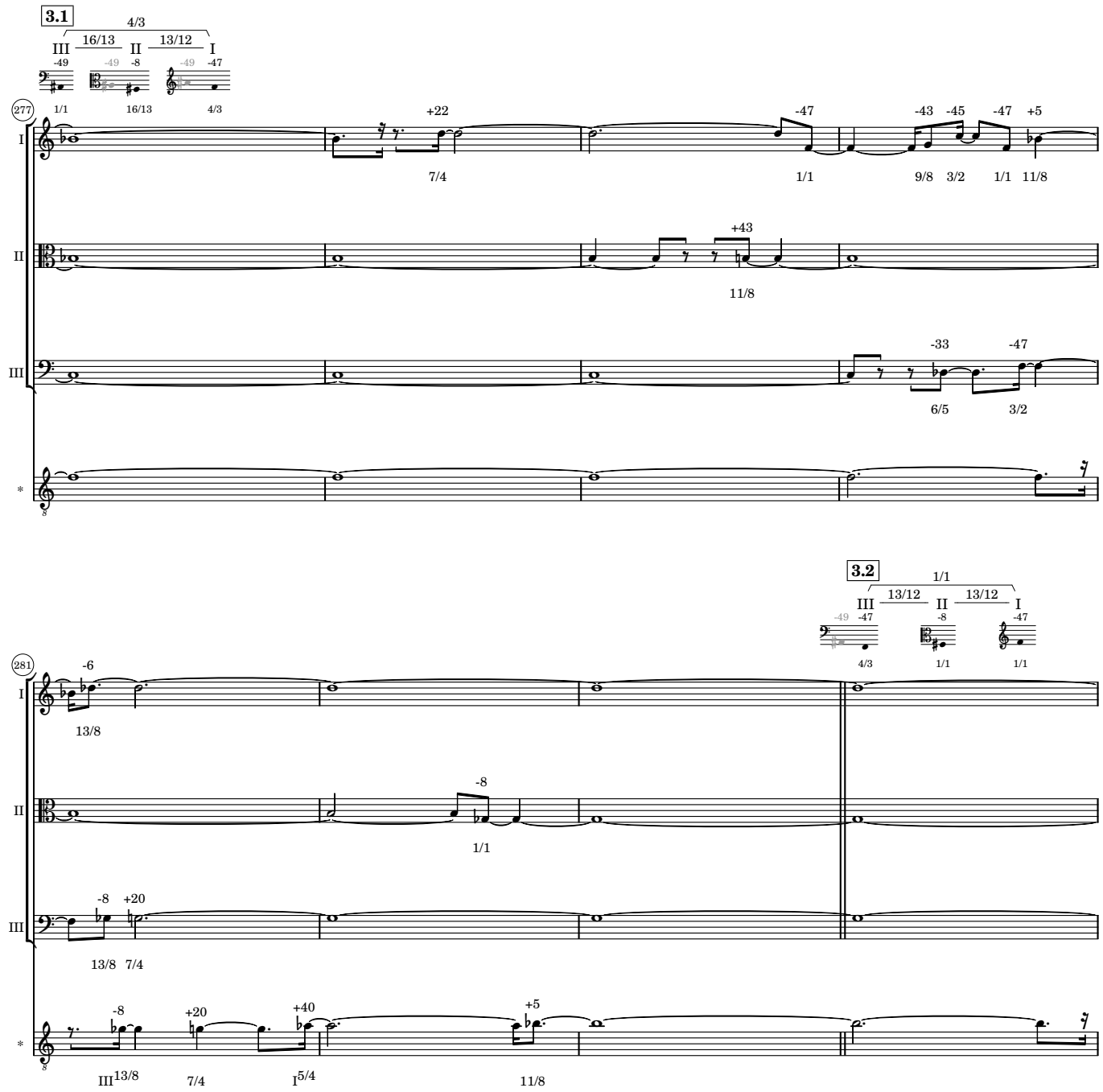

2.1
4/3
III 9/8 II 3/2 I
2.2
9/8
III 4/3 II 3/2 I
2.3
4/3
III 3/2 II 9/8 I
241
245
249
I
II
III
*

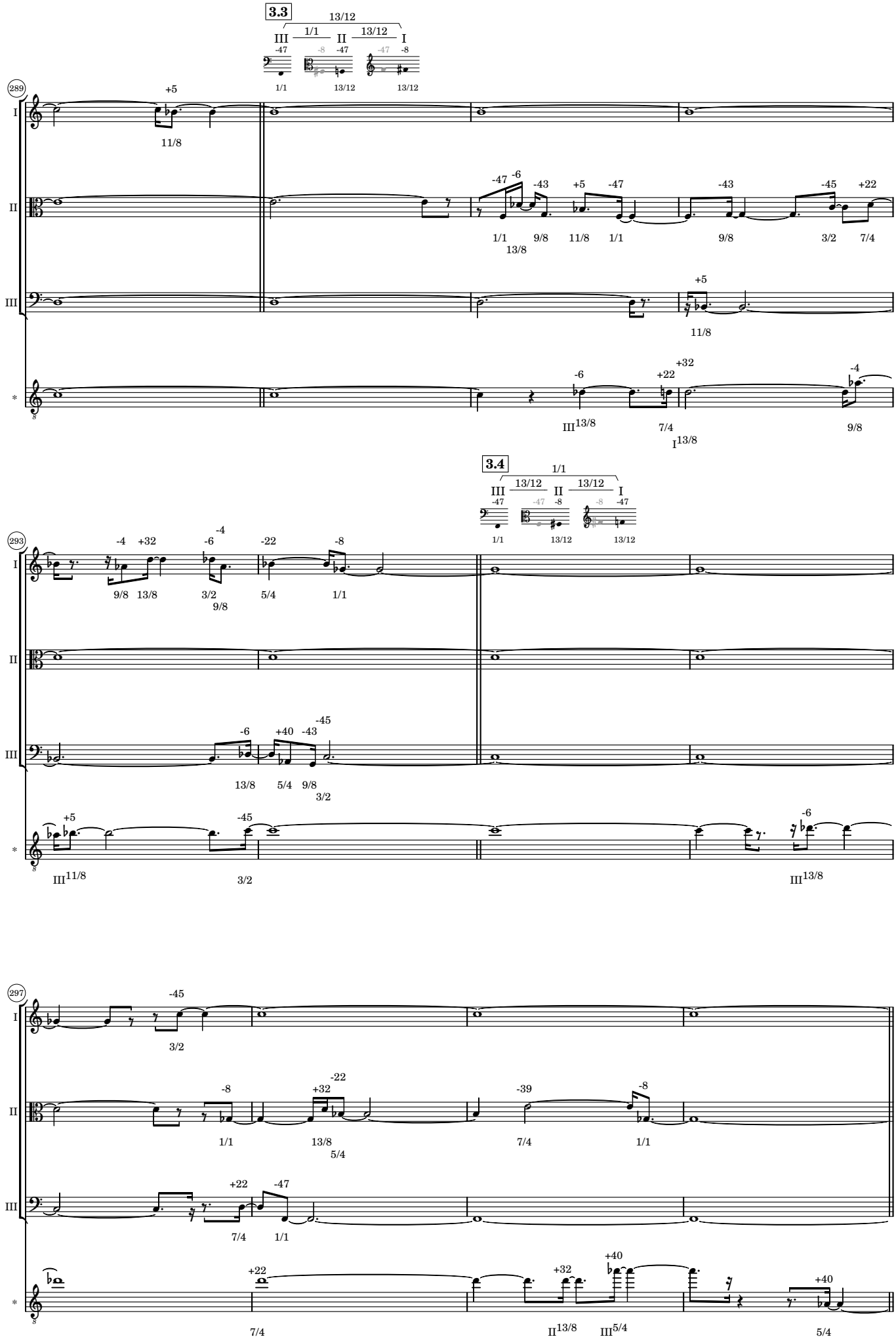

2.4
1/1
III 3/2 II 3/2 I
253
I
II
III
*
III9/8

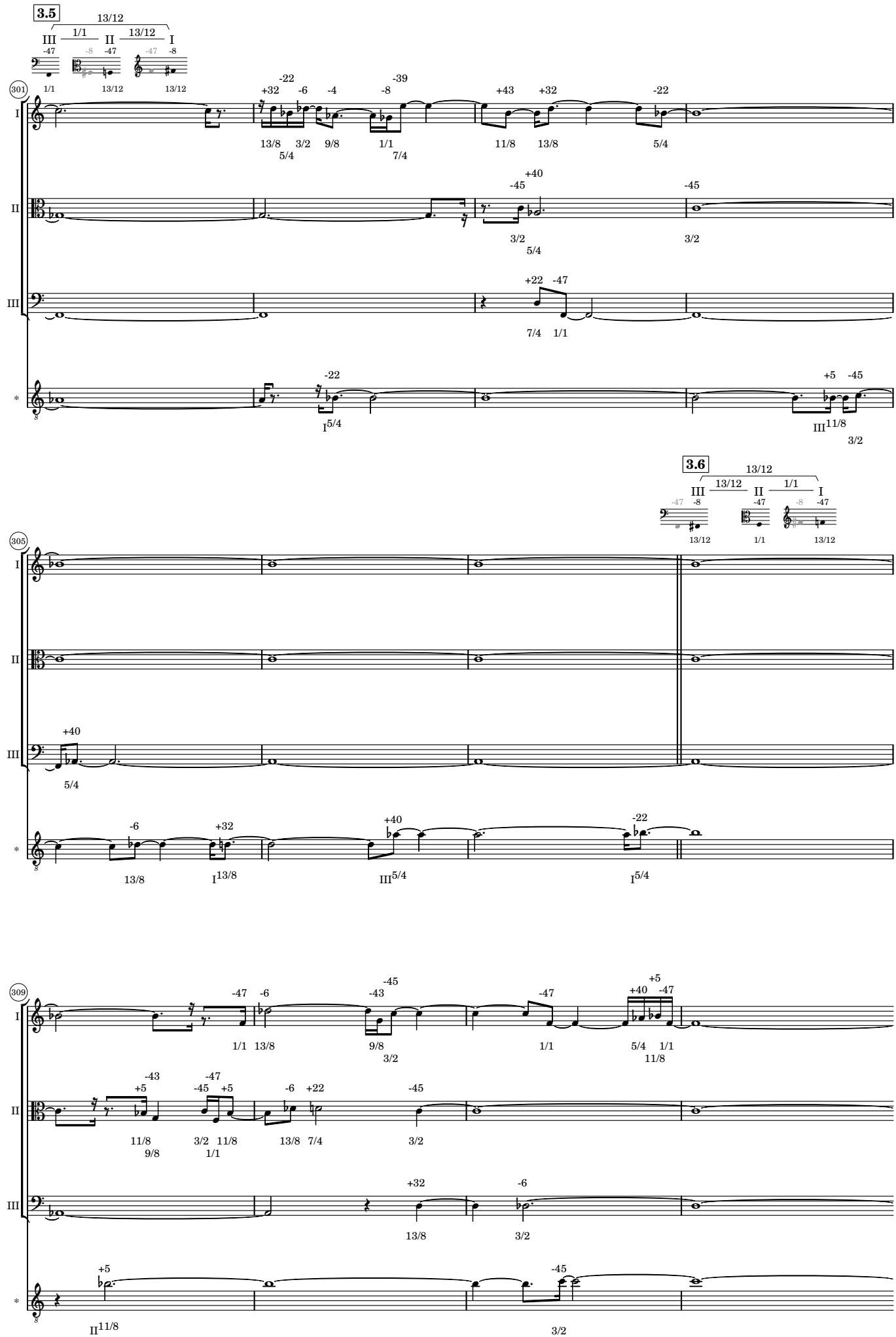

257
I
II
III
*
III6/5

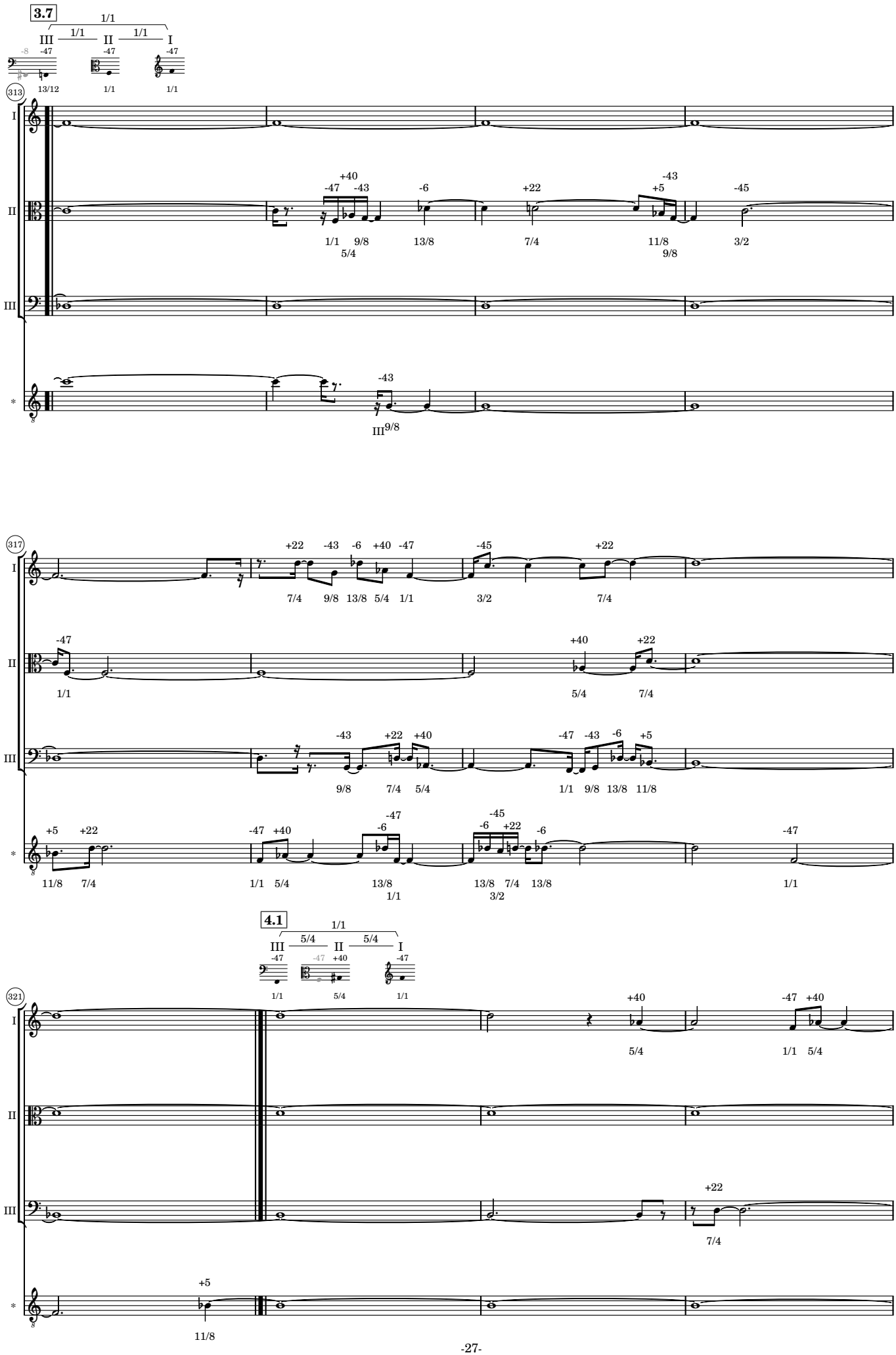

2.5
11/8
III 3/2 II 12/11 I
261
I
II
III
*
II9/8

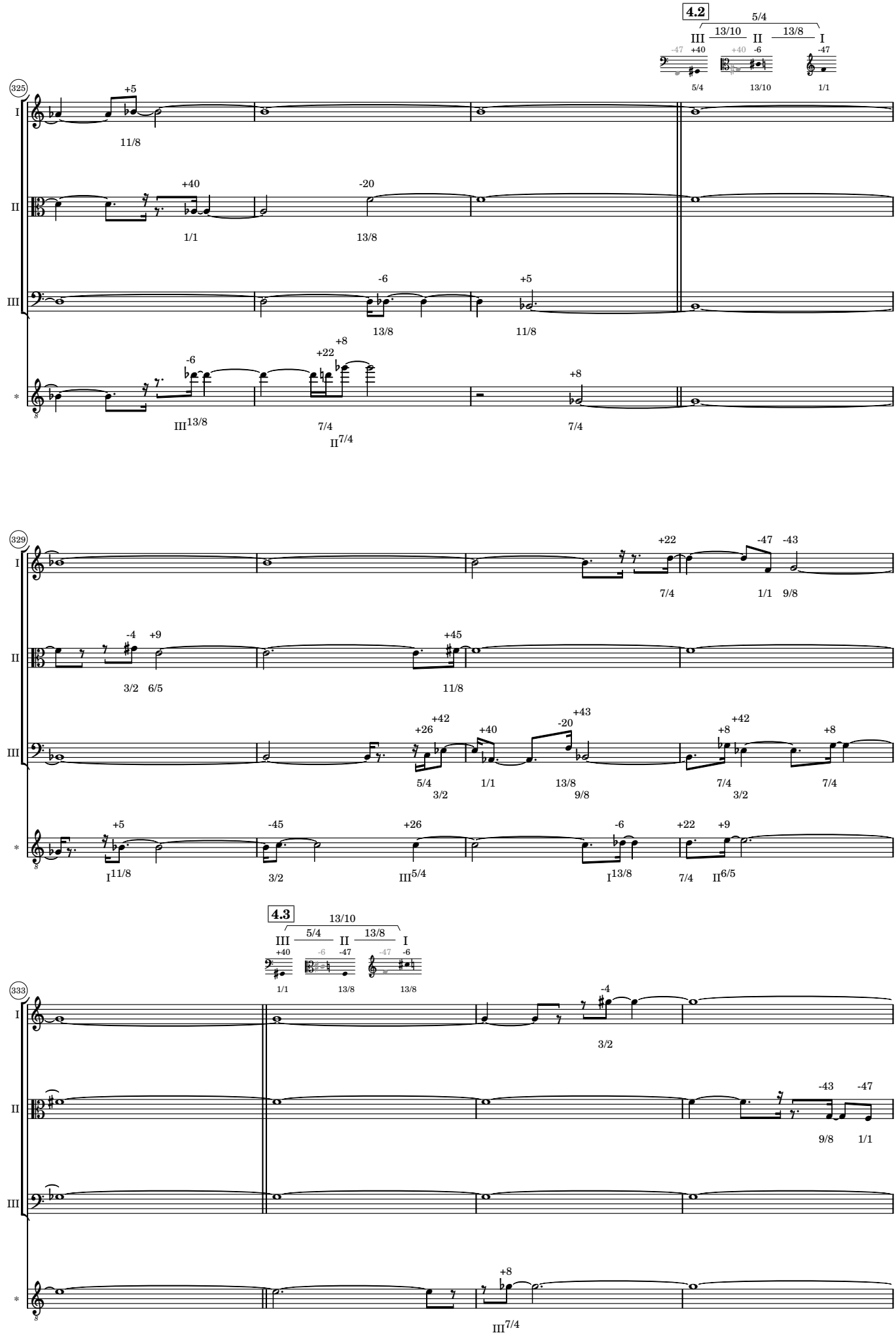

2.6
1/1
III 1/1 II 1/1 I
265
I
II
III
*
269
273

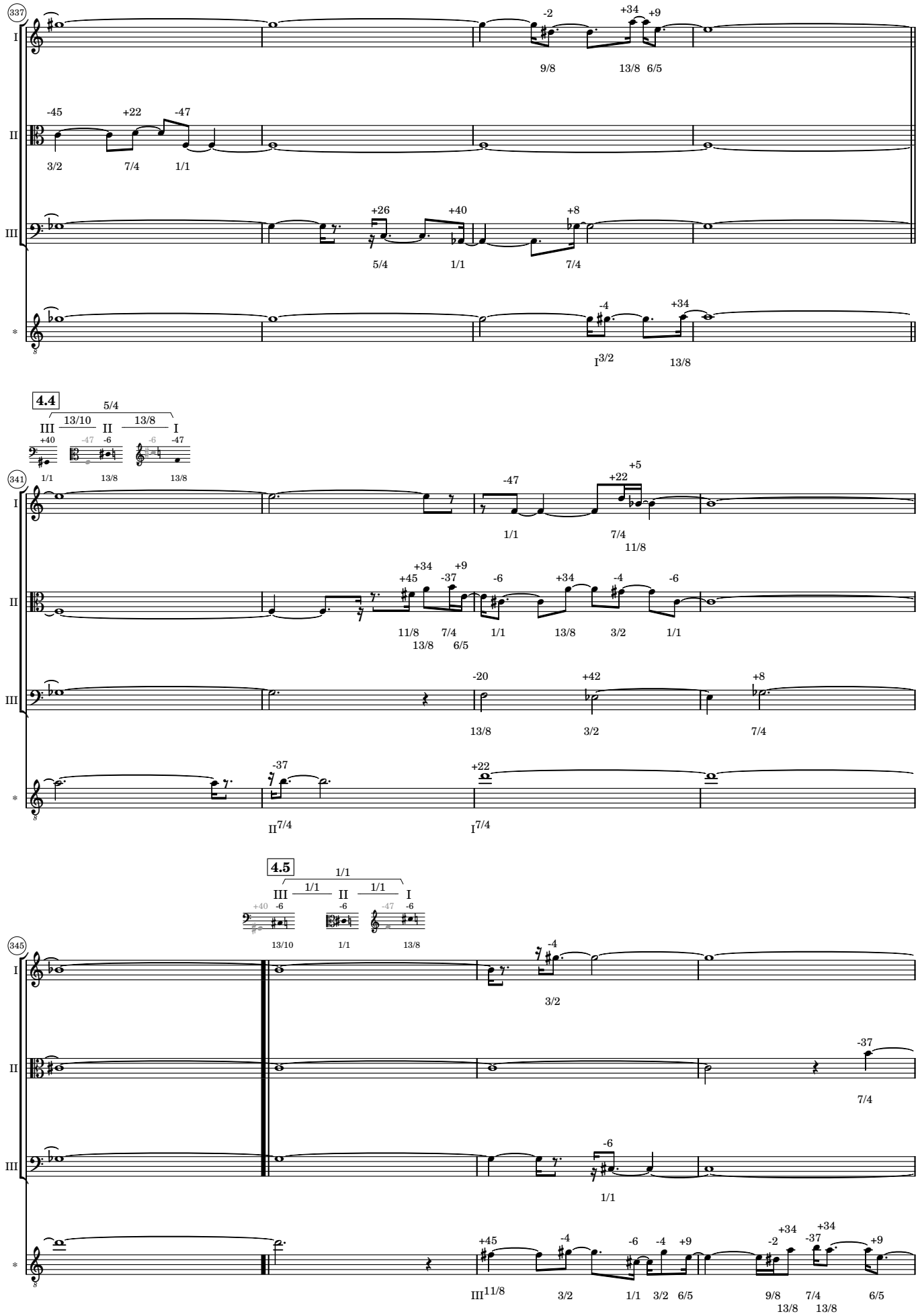

3.1
4/3
III 16/13 II 13/12 I
277
I
II
III
*
3.2
1/1
III 13/12 II 13/12 I
281
285

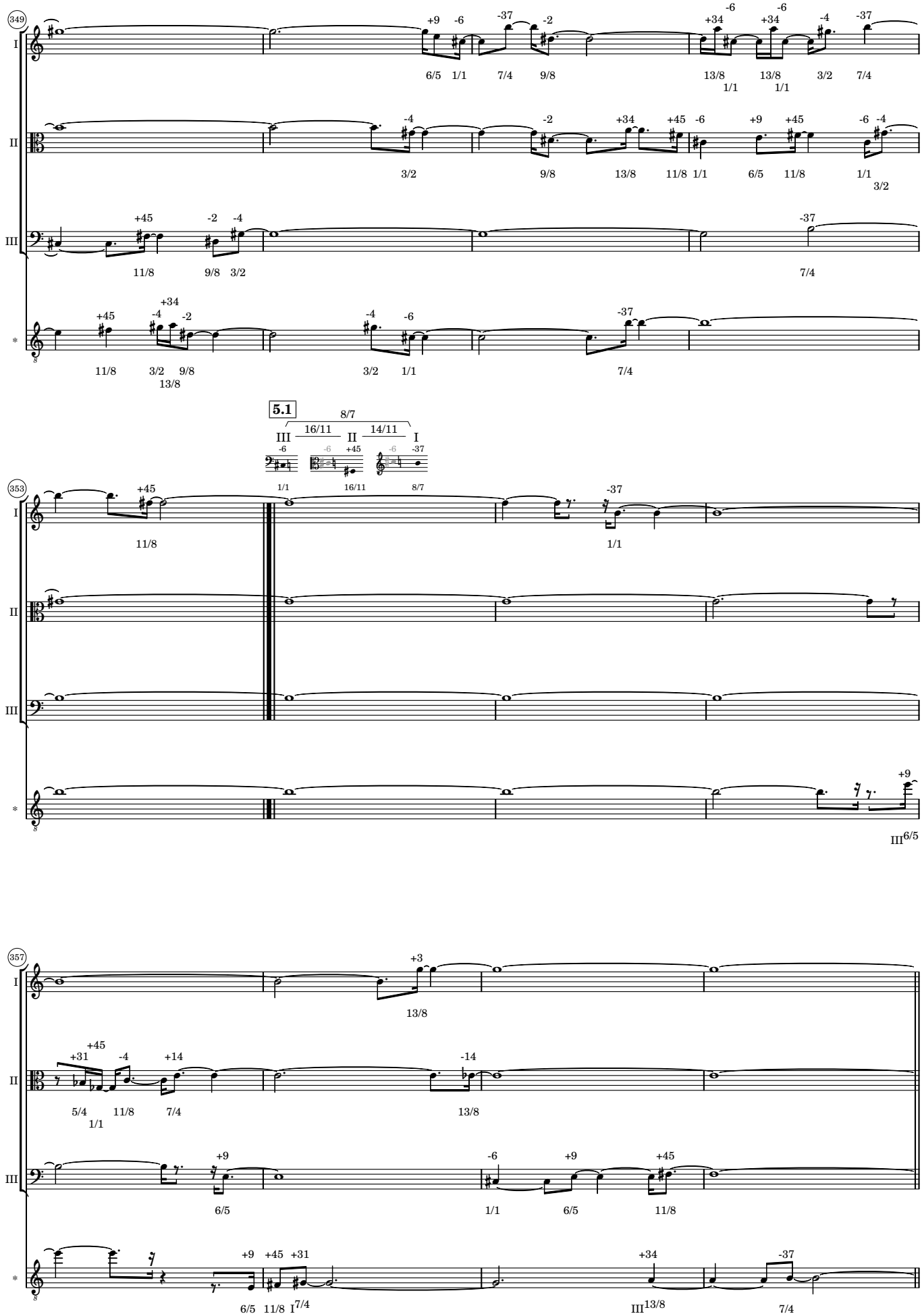

3.3
13/12
III 1/1 II 13/12 I
289
I
II
III
*
11/8
13/8
9/8
11/8
1/1
3/2
7/4
III13/8
I13/8
3.4
1/1
III 13/12 II 13/12 I
293
9/8
13/8
3/2
5/4
1/1
13/8
5/4
9/8
3/2
III11/8
III13/8
297
3/2
1/1
13/8
5/4
7/4
1/1
II13/8
III5/4
5/4

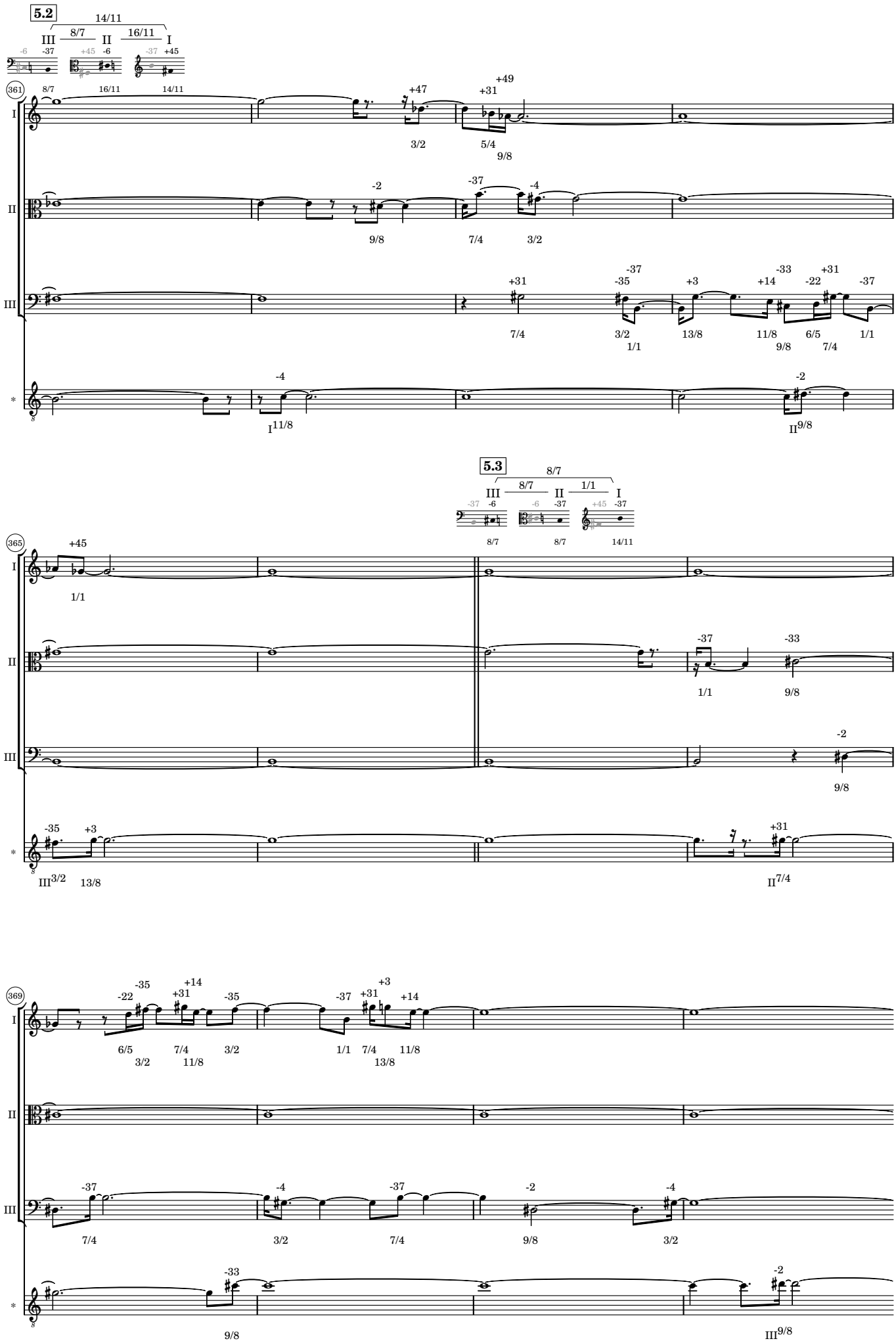

3.5
3.6
301
305
309
I
II
III
*

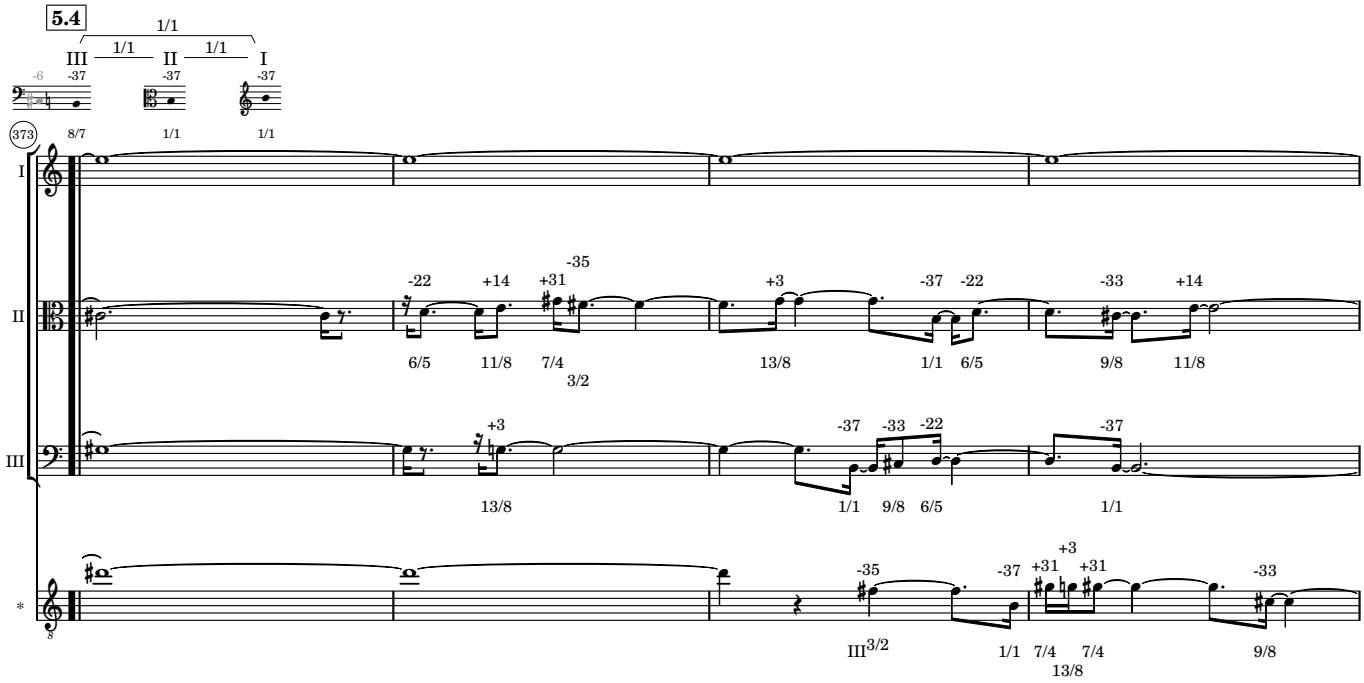

3.7
1/1
III 1/1 II 1/1 I
313
I
II
III
*
317
4.1
321
-27-

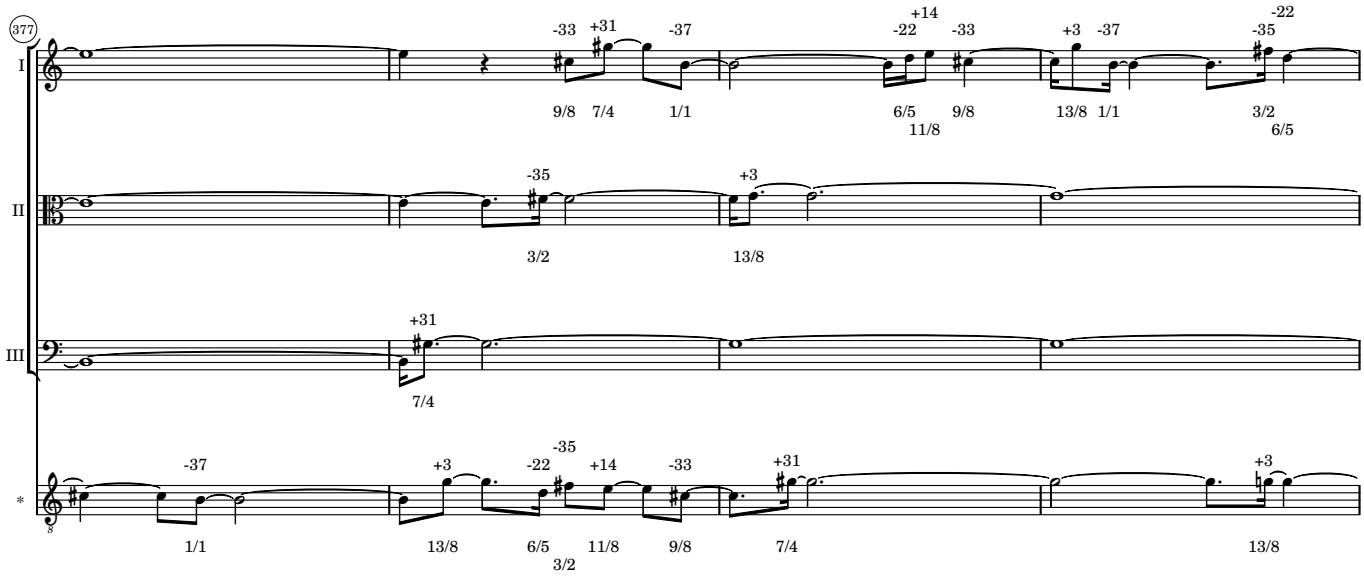

4.2
5/4
III 13/10 II 13/8 I
4.3
13/10
III 5/4 II 13/8 I
325
329
333
I
II
III
*

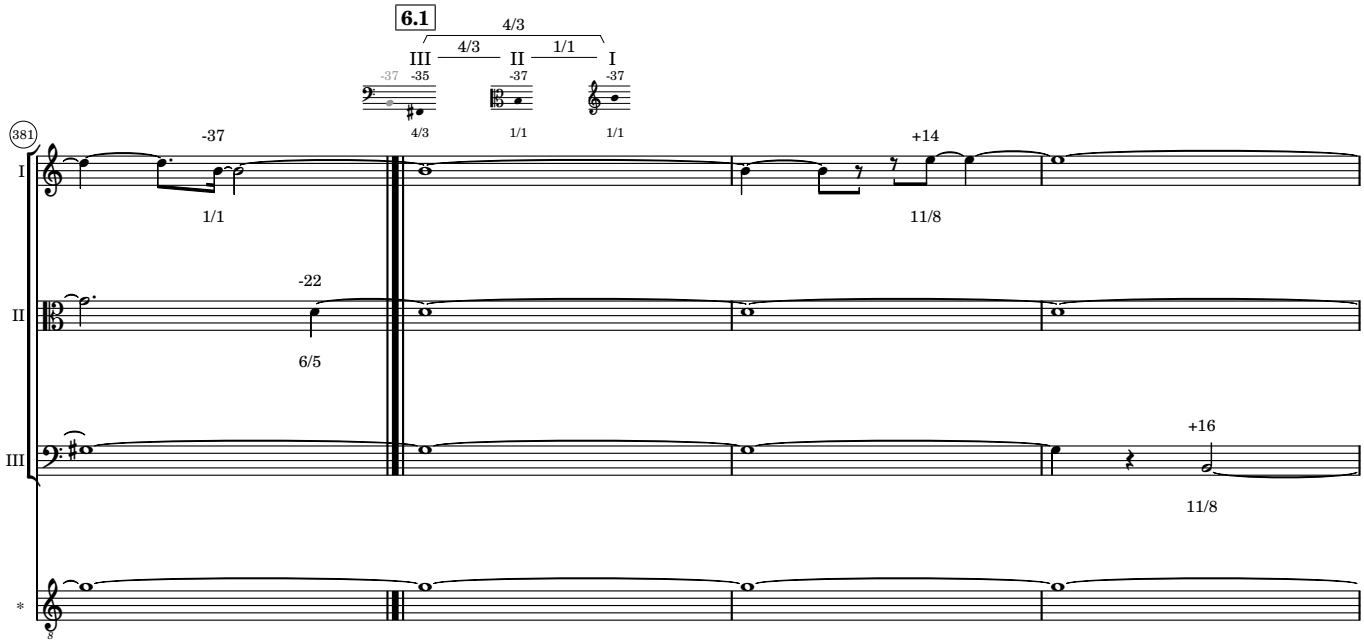

337
I
II
III
*
-2
+34
+9
9/8
13/8
6/5
-45
+22
-47
3/2
7/4
1/1
+26
+40
+8
5/4
1/1
7/4
-4
+34
I3/2
13/8
4.4
5/4
III
13/10
II
13/8
I
+40
-47
-6
-6
-47
1/1
13/8
13/8
341
-47
+22
+5
1/1
7/4
11/8
+45
+34
-37
+9
-6
+34
-4
-6
11/8
13/8
7/4
6/5
1/1
13/8
3/2
1/1
-20
+42
+8
13/8
3/2
7/4
-37
+22
II7/4
I7/4
4.5
1/1
III
1/1
II
1/1
I
+40
-6
-6
-47
-6
13/10
1/1
13/8
345
-4
3/2
-37
7/4
-6
1/1
+45
-4
-6
-4
+9
-2
+34
-37
+34
+9
III11/8
3/2
1/1
3/2
6/5
9/8
13/8
7/4
13/8
6/5

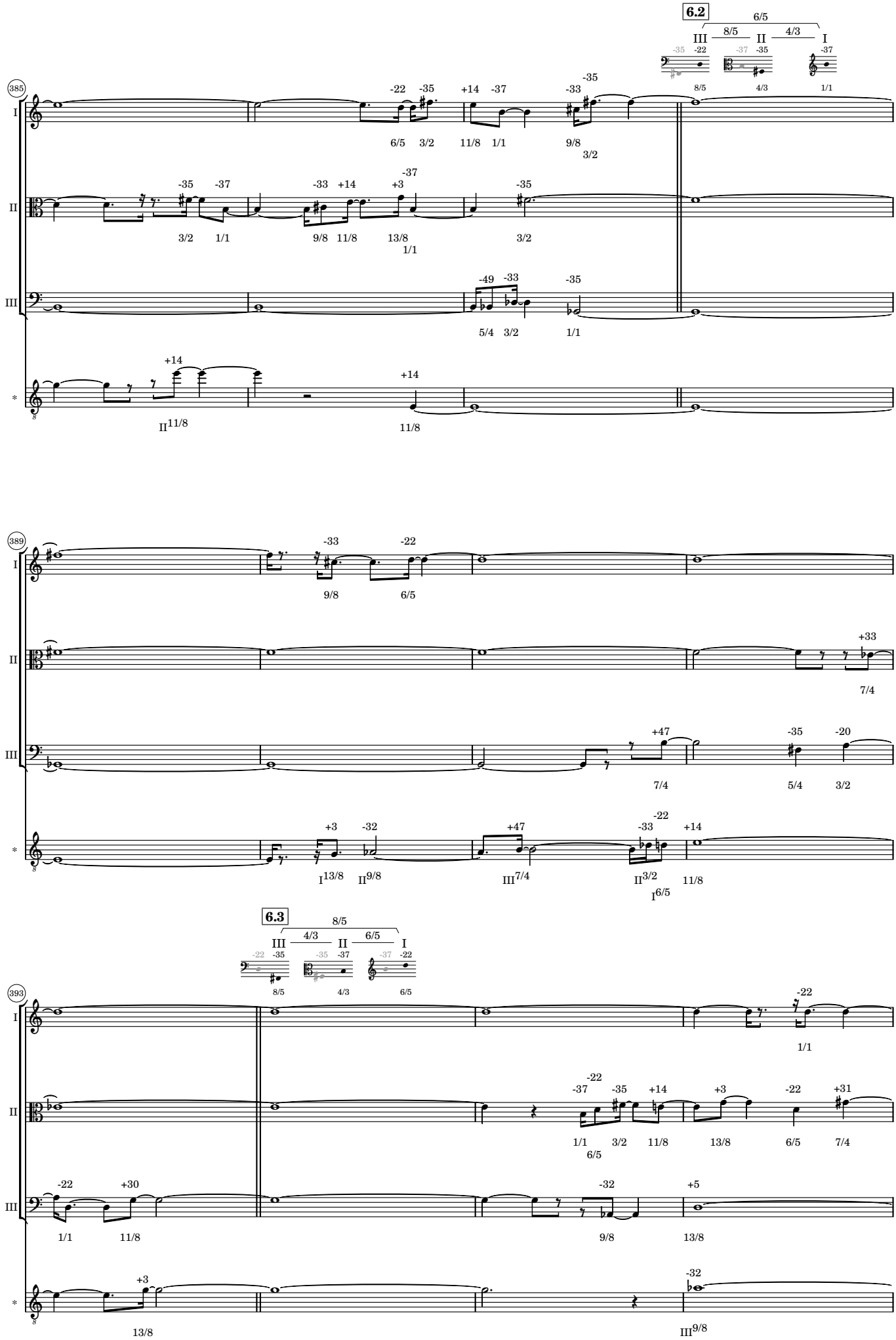

349
I
II
III
*
+9 -6 -37 -2
6/5 1/1 7/4 9/8
+34 -6 +34 -6 -4 -37
13/8 1/1 13/8 1/1 3/2 7/4
-4 -2 +34 +45 -6 +9 +45 -6 -4
3/2 9/8 13/8 11/8 1/1 6/5 11/8 1/1 3/2
+45 -2 -4 -37
11/8 9/8 3/2 7/4
+45 +34 -4 -2 -4 -6 -37
11/8 3/2 9/8 13/8 3/2 1/1 7/4
5.1
8/7
III 16/11 II 14/11 I
-6 +45 -37
1/1 16/11 8/7
353
+45 -37
11/8 1/1
+9
III6/5
357
+3
13/8
+31 +45 -4 +14 -14
5/4 1/1 11/8 7/4 13/8
+9 -6 +9 +45
6/5 1/1 6/5 11/8
+9 +45 +31 +34 -37
6/5 11/8 I7/4 III13/8 7/4

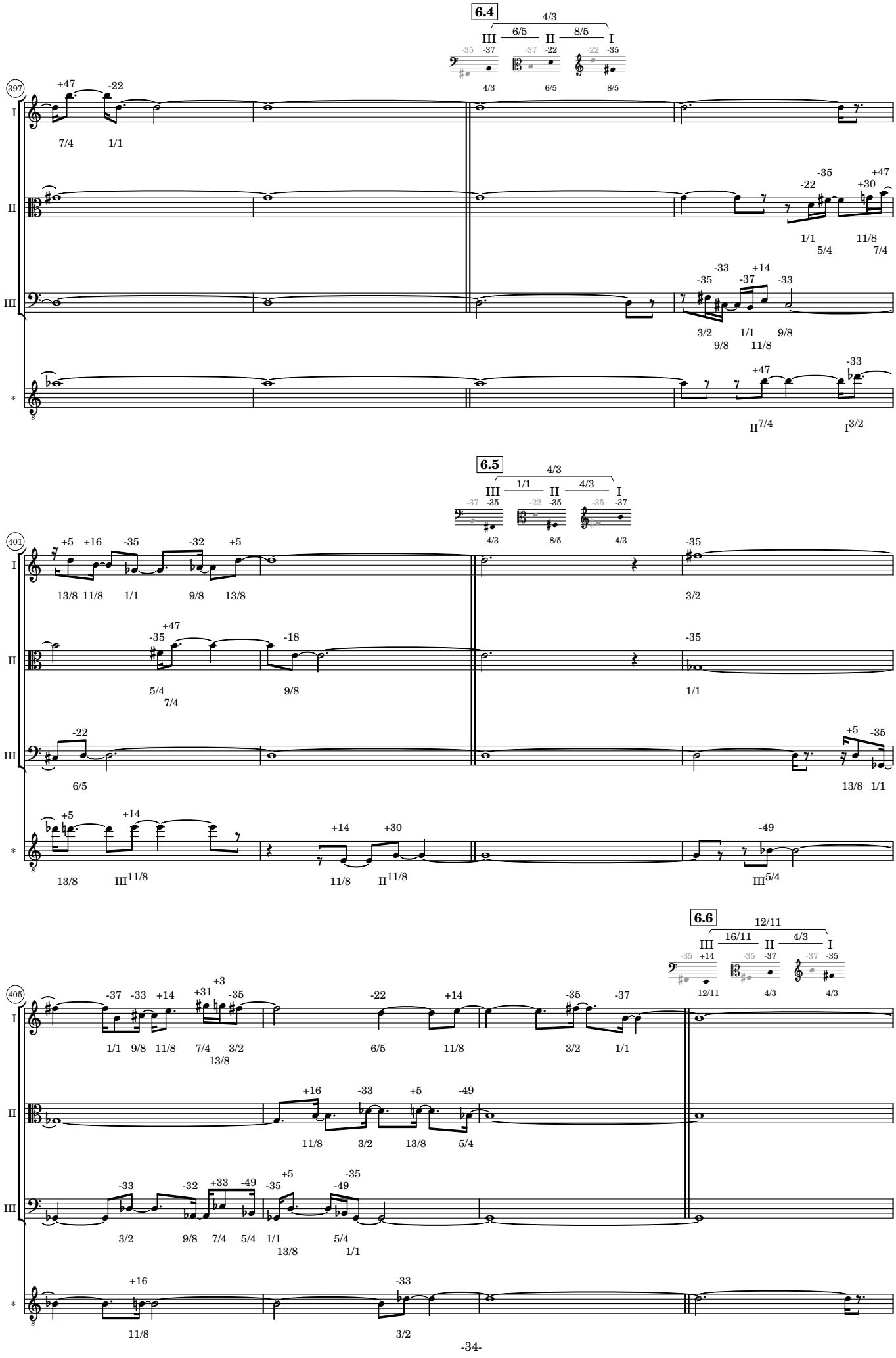

5.2
14/11
III 8/7 II 16/11 I
361
I
II
III
*
5.3
8/7
III 8/7 II 1/1 I
365
369

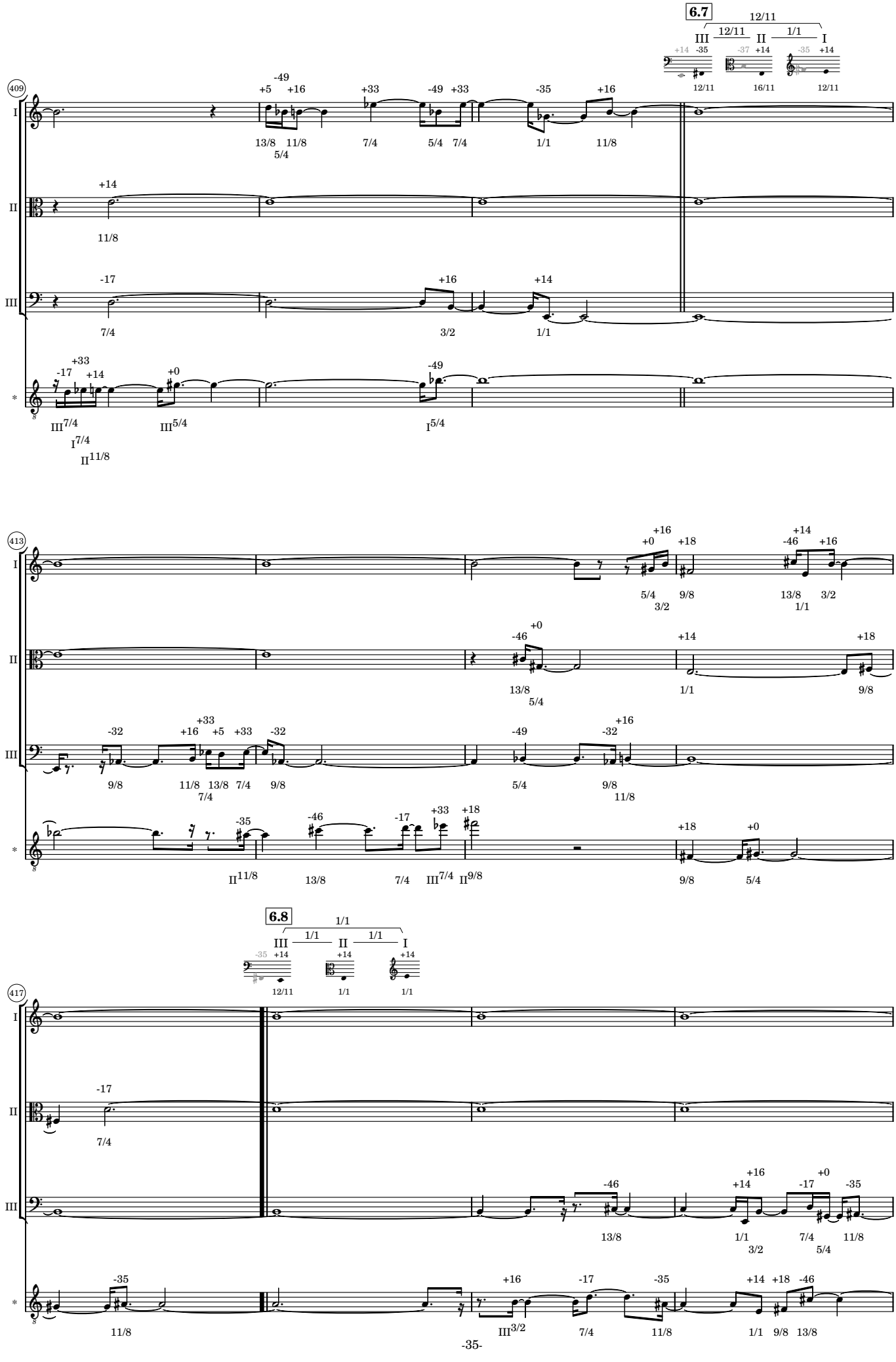

5.4
1/1
III 1/1 II 1/1 I
373
8/7
I
II
III
*

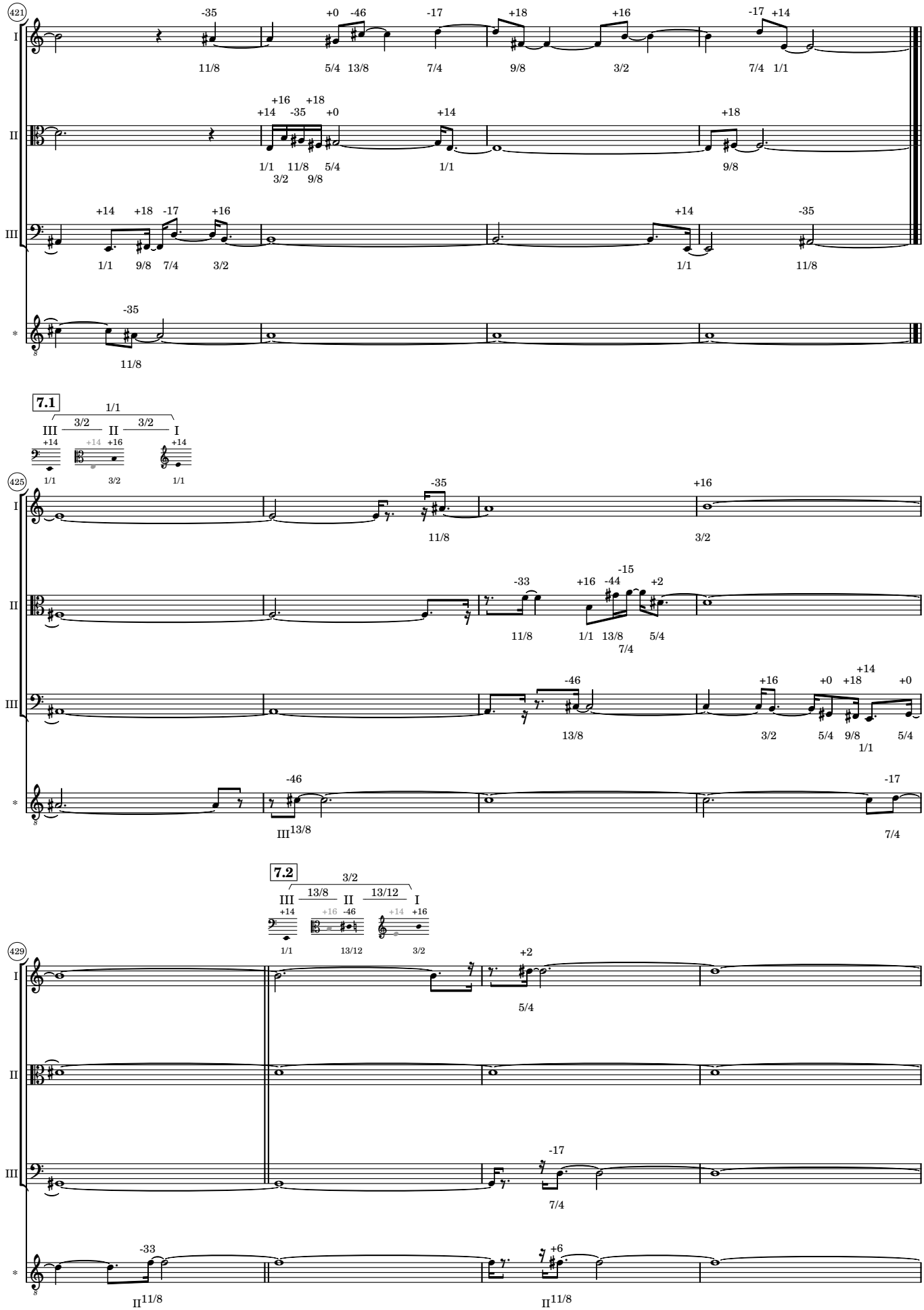

377
I
II
III
*

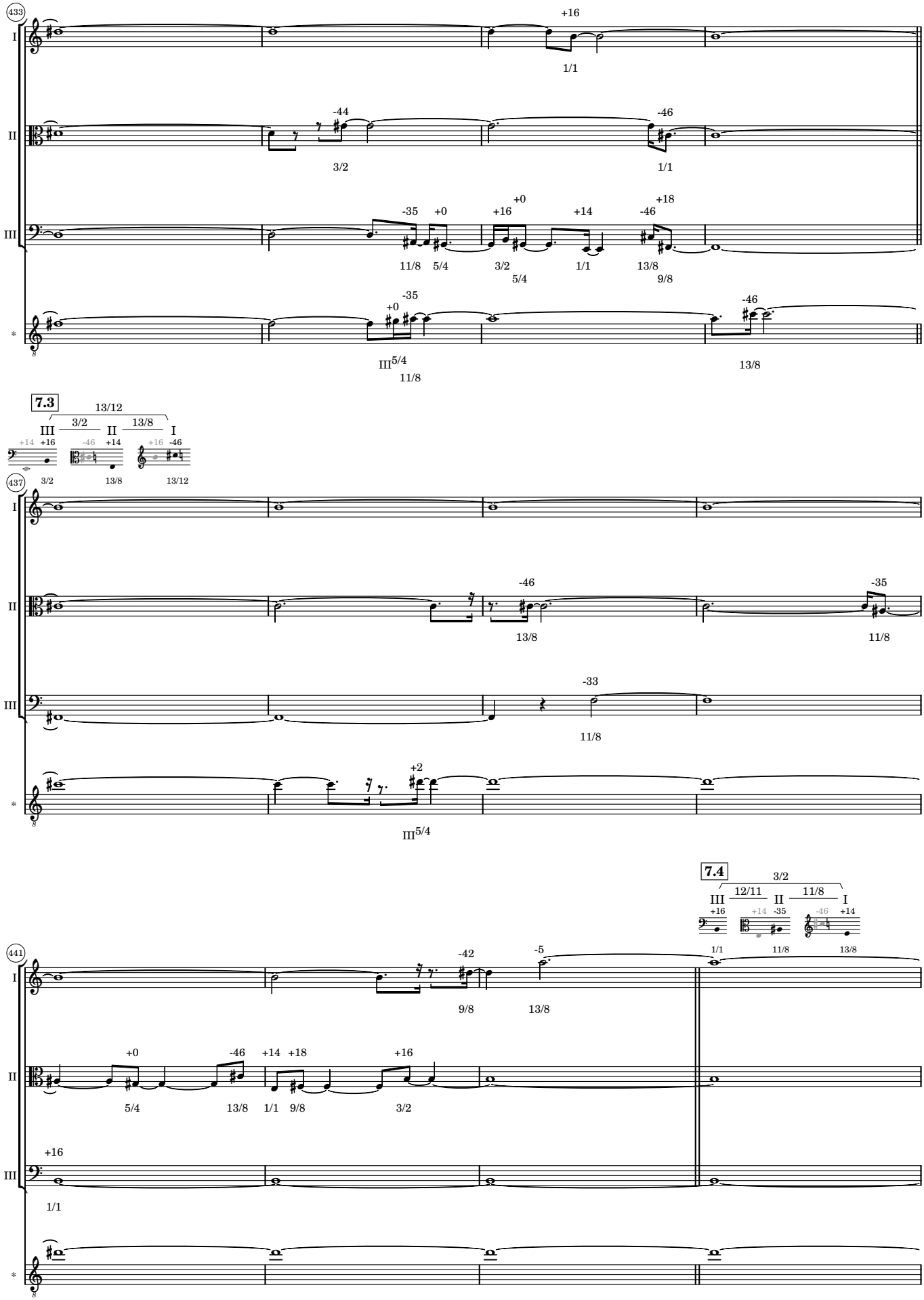

6.1
4/3
III 4/3 II 1/1 I
381
I
II
III
*

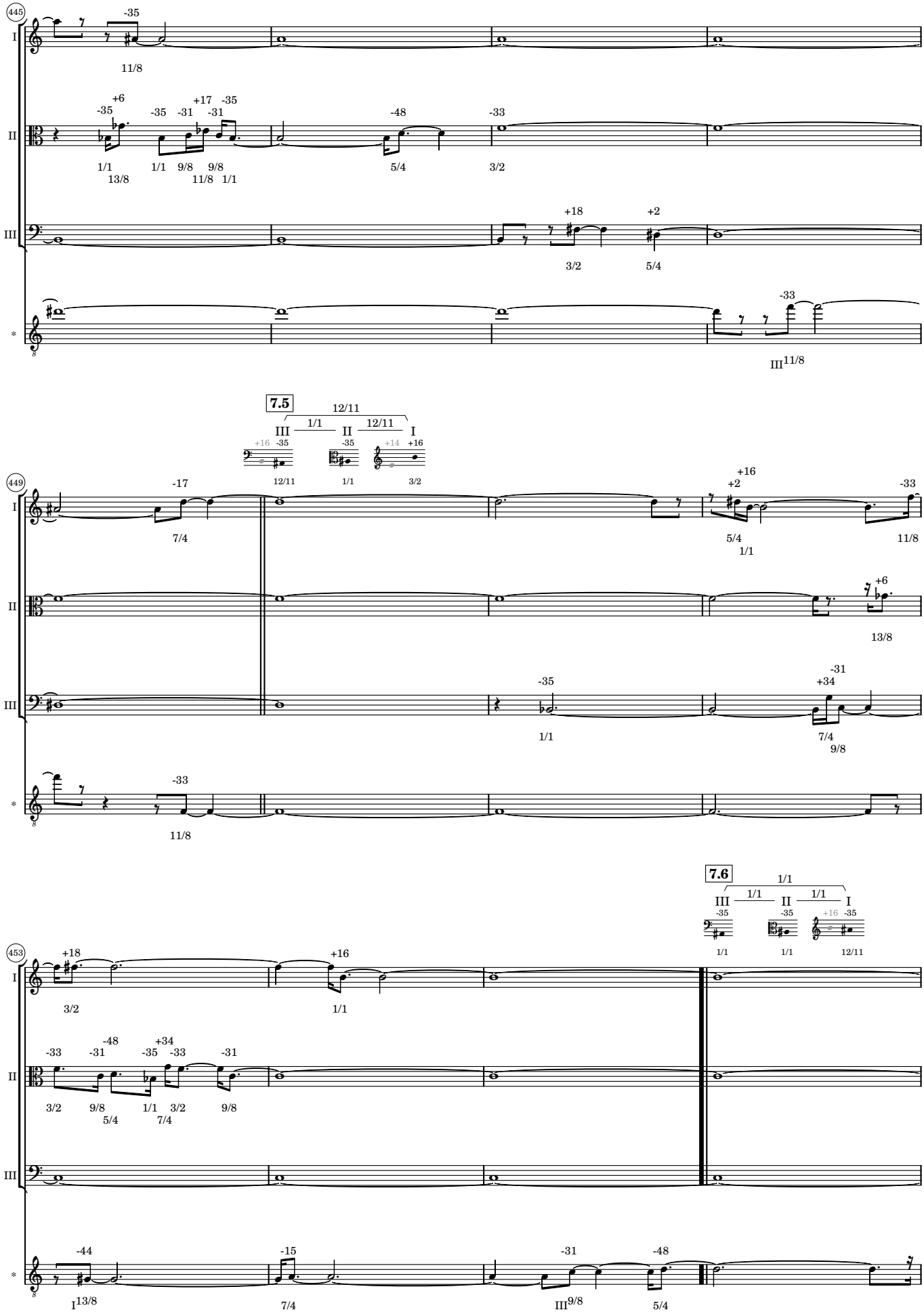

6.2
6.3
385
389
393
I
II
III
*

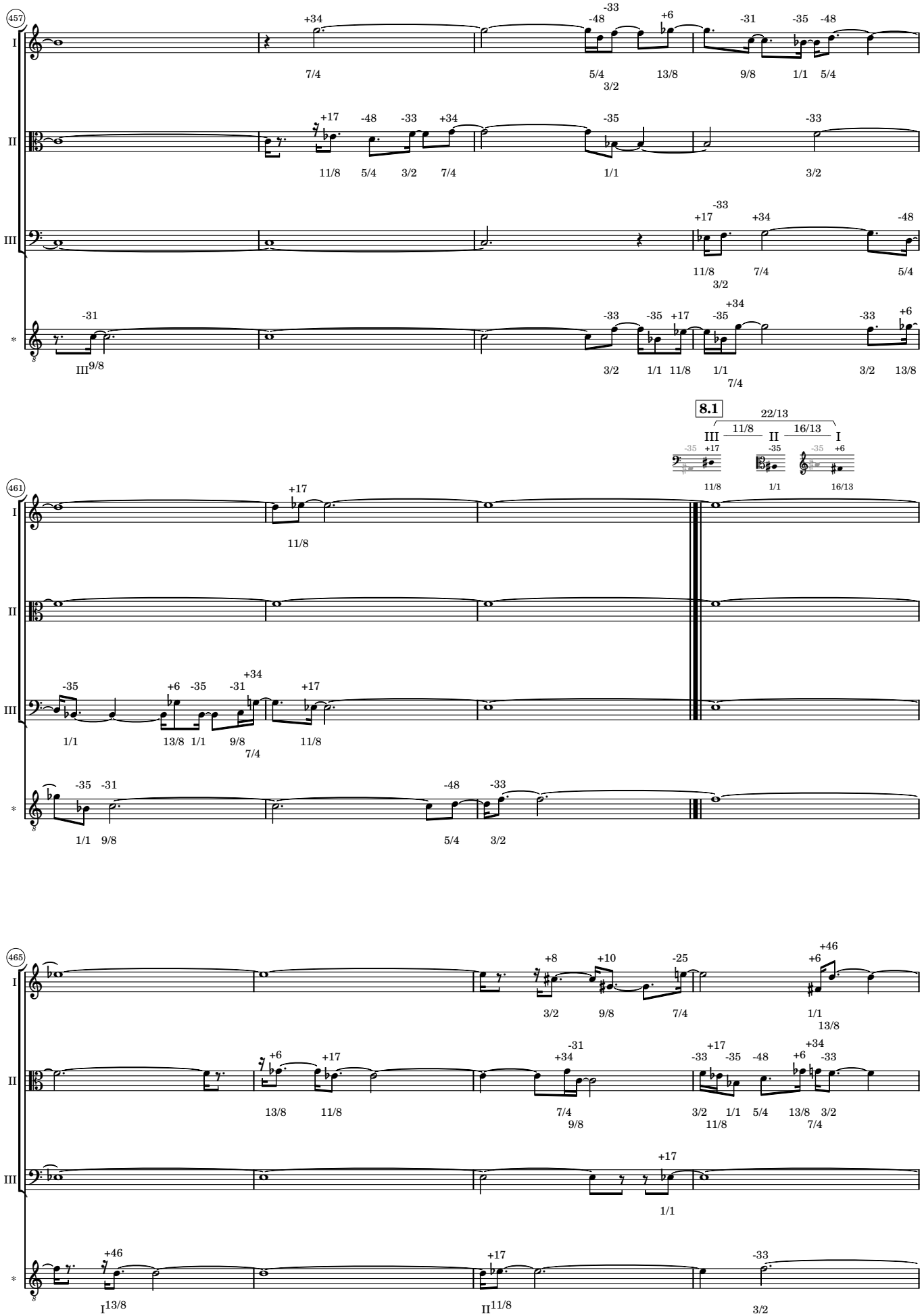

6.4
4/3
III 6/5 II 8/5 I
6.5
4/3
III 1/1 II 4/3 I
6.6
12/11
III 16/11 II 4/3 I
397
401
405
I
II
III
*

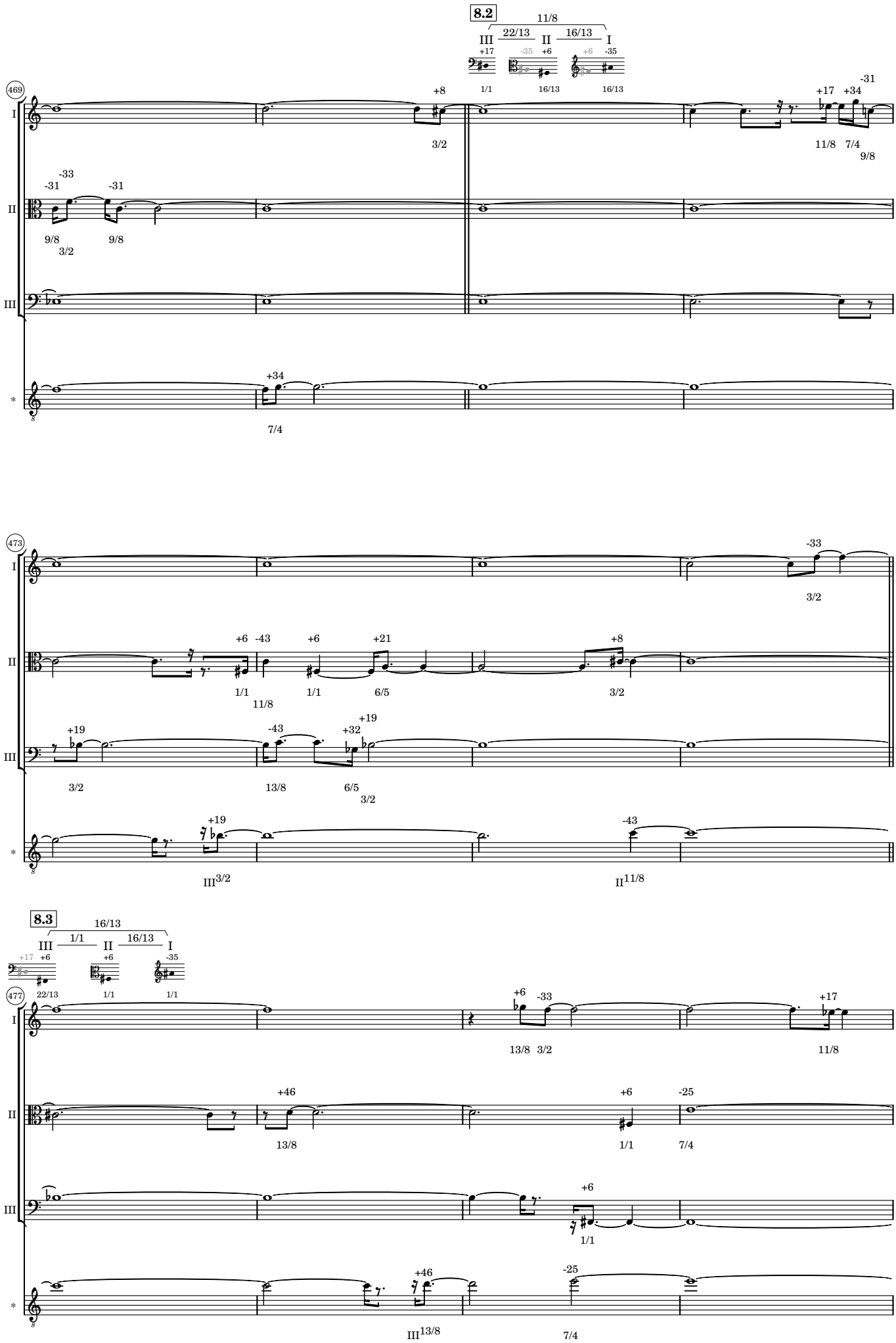

6.7
12/11
III 12/11 II 1/1 I
409
I
II
III
*
413
417
6.8
1/1
III 1/1 II 1/1 I

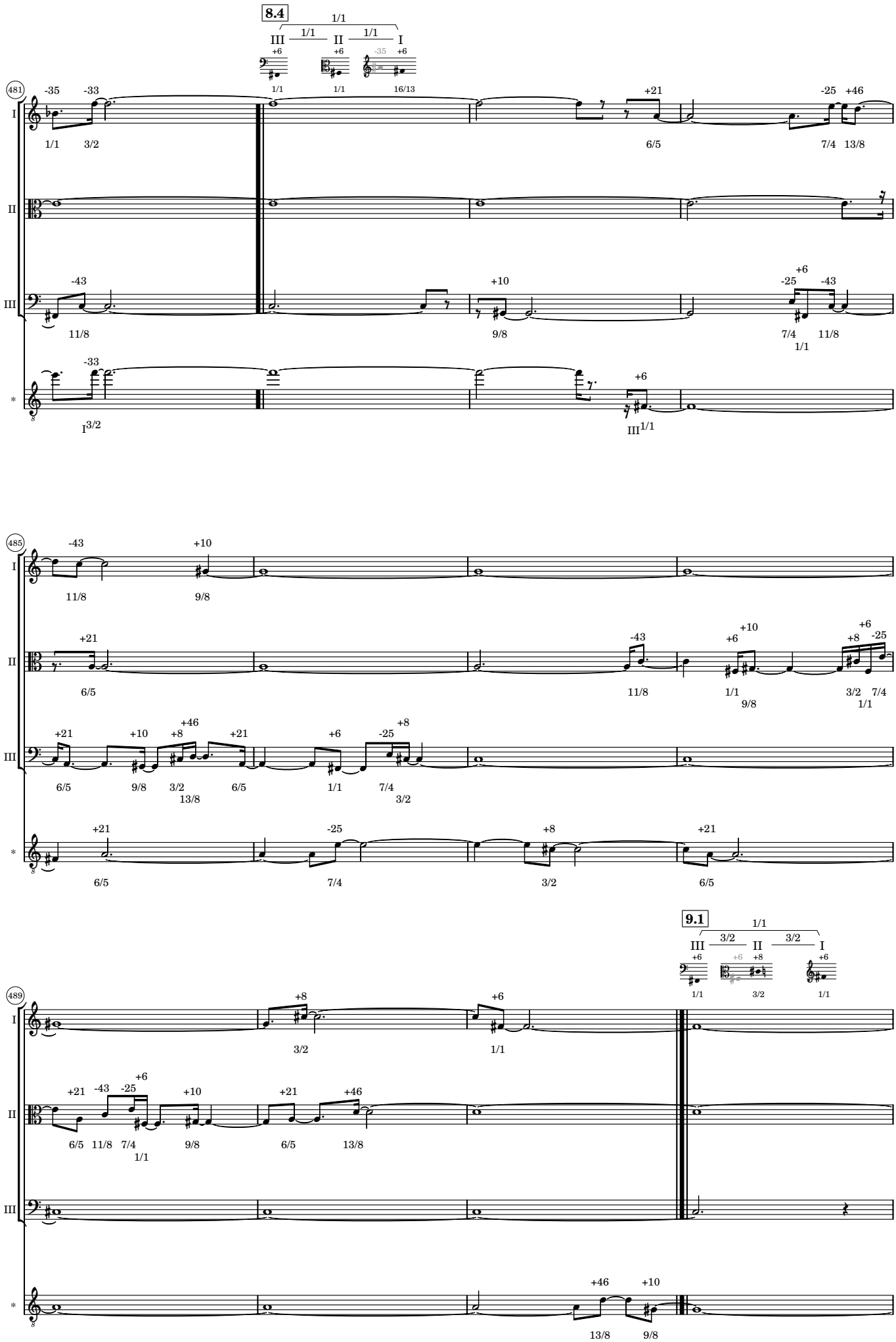

421
I
II
III
*
-35
11/8
+0 -46
5/4 13/8
-17
7/4
+18
9/8
+16
3/2
-17 +14
7/4 1/1
+16 +18
+14 -35 +0
1/1 11/8 5/4
3/2 9/8
+14
1/1
+18
9/8
+14 +18 -17 +16
1/1 9/8 7/4 3/2
+14
1/1
-35
11/8
-35
11/8
7.1
1/1
III 3/2 II 3/2 I
+14
1/1
+14 +16
3/2
+14
1/1
425
-35
11/8
+16
3/2
-33
11/8
+16 -44
-15
+2
1/1 13/8
7/4
5/4
-46
13/8
+16
3/2
+0 +18
+14
5/4 9/8
1/1
+0
5/4
-46
III13/8
-17
7/4
7.2
3/2
III 13/8 II 13/12 I
+14
1/1
+16 -46
13/12
+14 +16
3/2
429
+2
5/4
-17
7/4
-33
II11/8
+6
II11/8

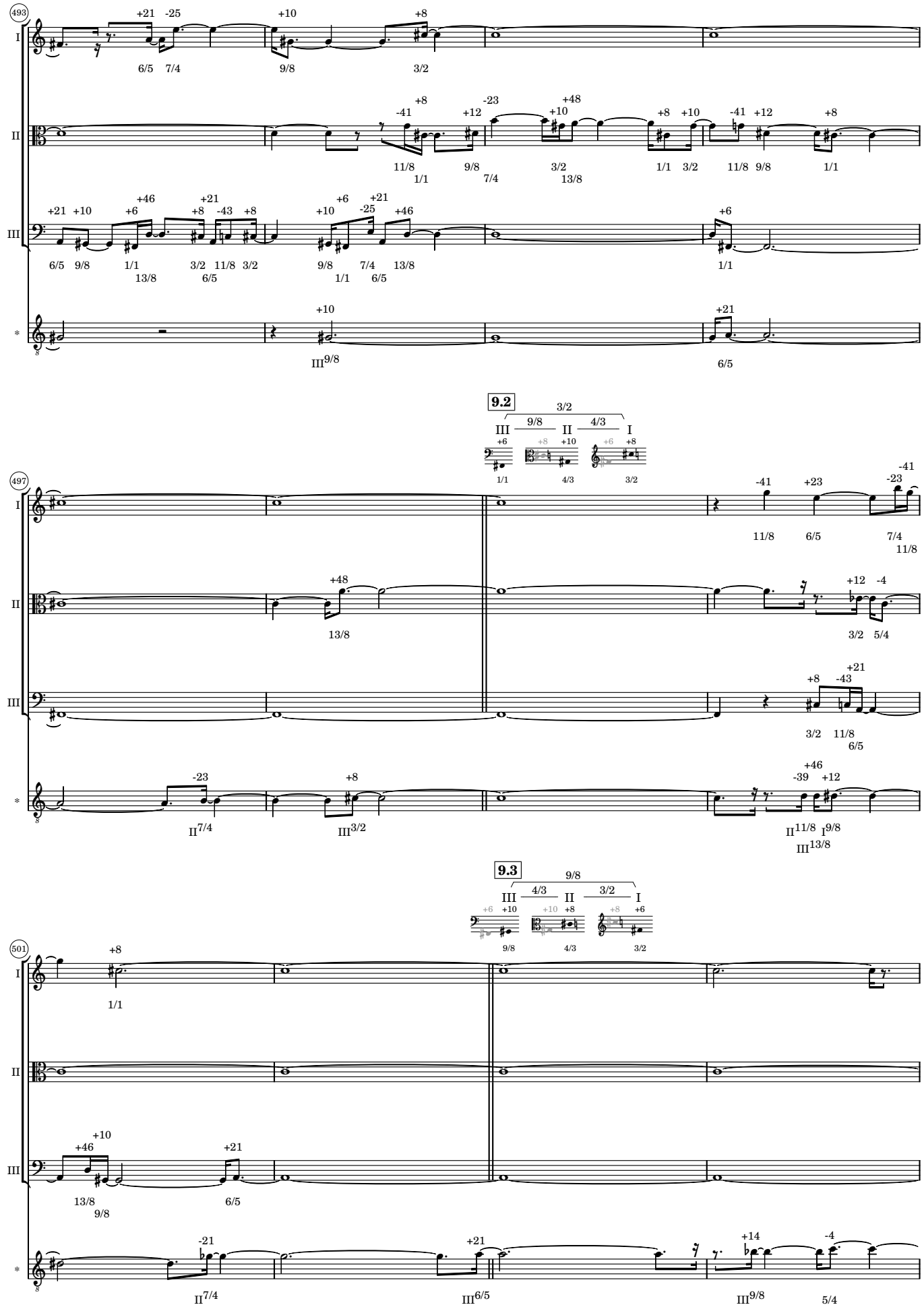

433
I
II
III
*
+16
1/1
-44
-46
3/2
1/1
-35
+0
+16
+14
-46
+18
+0
11/8
5/4
3/2
1/1
13/8
5/4
9/8
-35
+0
-46
III5/4
11/8
13/8
7.3
13/12
III
3/2
II
13/8
I
+14
+16
-46
+14
+16
-46
3/2
13/8
13/12
437
-46
-35
13/8
11/8
-33
11/8
+2
III5/4
7.4
3/2
III
12/11
II
11/8
I
+16
+14
-35
-46
+14
1/1
11/8
13/8
441
-42
-5
9/8
13/8
+0
-46
+14
+18
+16
5/4
13/8
1/1
9/8
3/2
+16
1/1

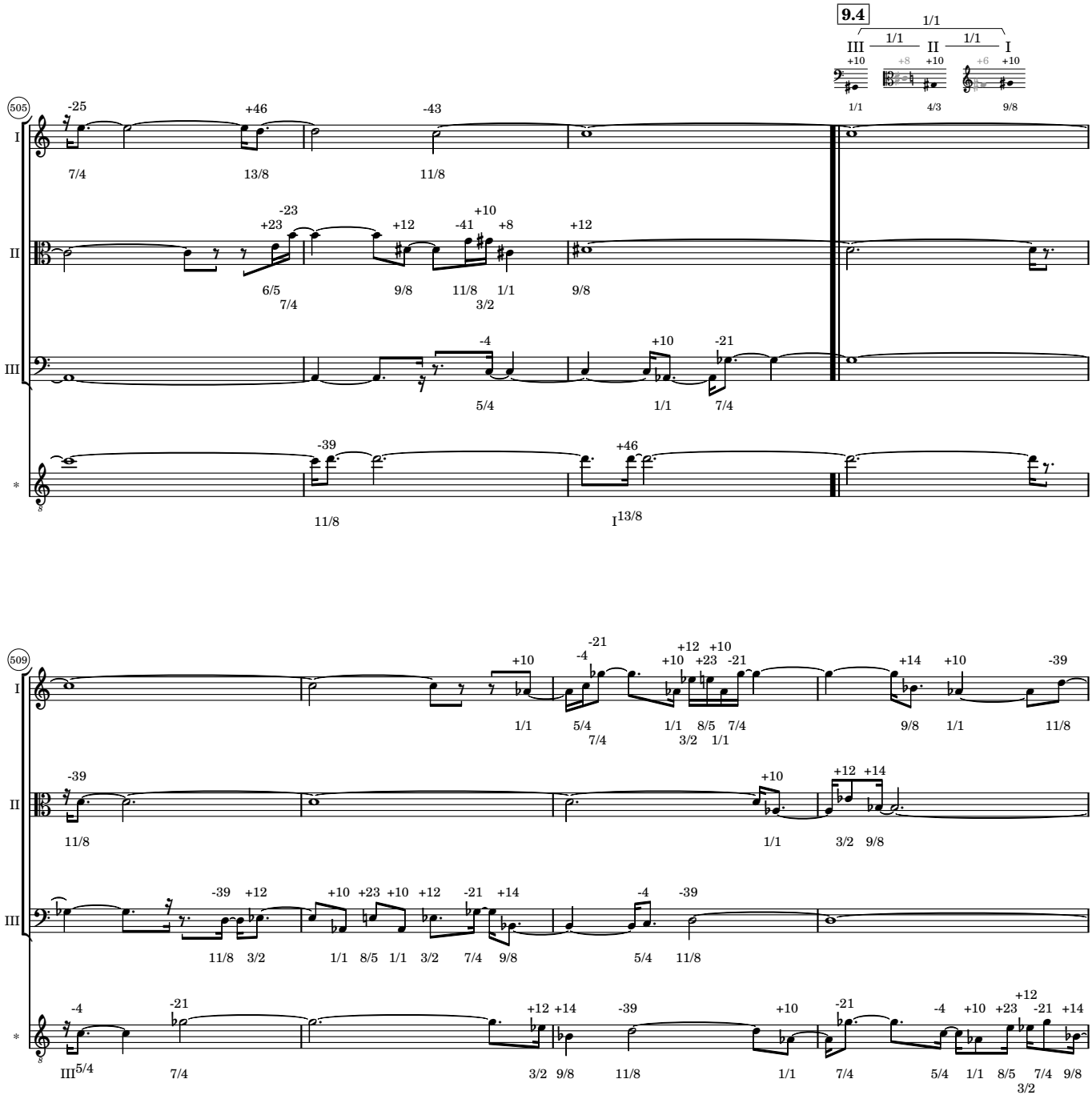

445
I
II
III
*
-35
11/8
+6
-35
-35 -31
+17 -35
-31
1/1 1/1 9/8 9/8
13/8 11/8 1/1
-48
5/4
-33
3/2
+18
+2
3/2
5/4
-33
III11/8
7.5
12/11
III
1/1
II
12/11
I
+16 -35
-35
+14 +16
12/11
1/1
3/2
449
-17
7/4
+16
+2
5/4
1/1
-33
11/8
+6
13/8
-35
1/1
-31
+34
7/4
9/8
-33
11/8
7.6
1/1
III
1/1
II
1/1
I
-35
-35
+16 -35
1/1
1/1
12/11
453
+18
3/2
+16
1/1
-48
-33
-31
-35 -33
+34
-31
3/2 9/8 1/1 3/2 9/8
5/4 7/4
-44
I13/8
-15
7/4
-31
III9/8
-48
5/4


457
I
II
III
*
461
465
8.1
22/13
III 11/8 II 16/13 I

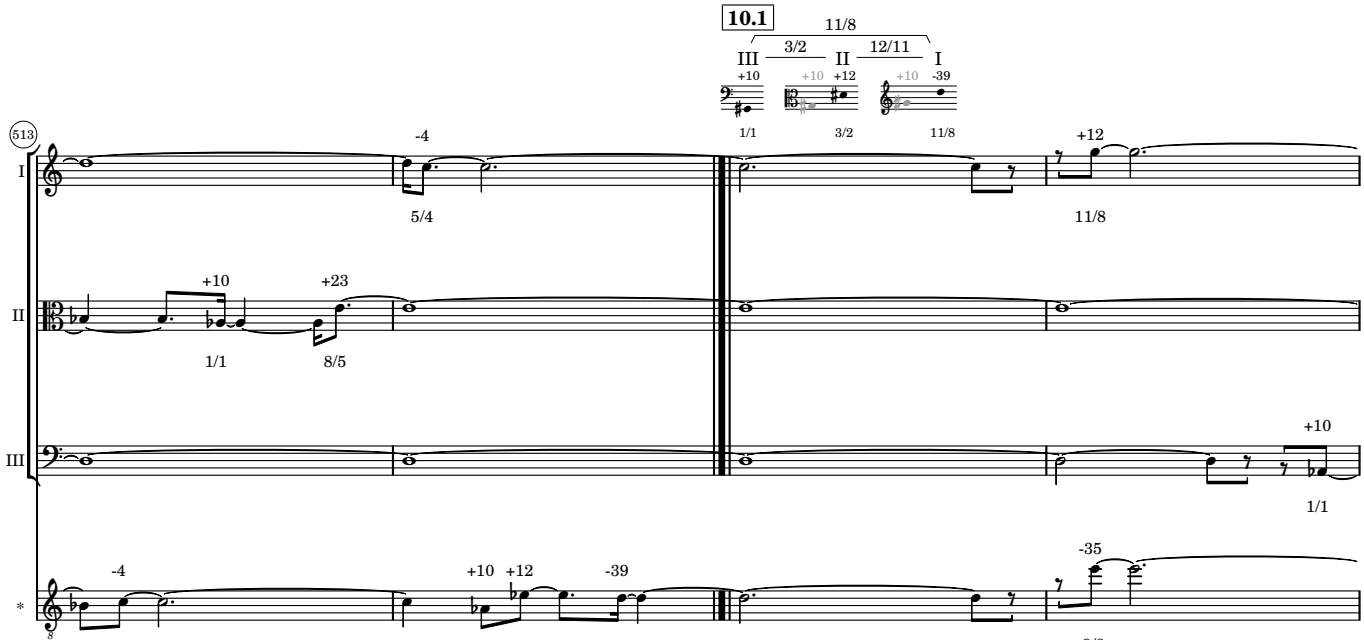

8.2
11/8
III 22/13 II 16/13 I
469
I
II
III
*
473
8.3
16/13
III 1/1 II 16/13 I
477

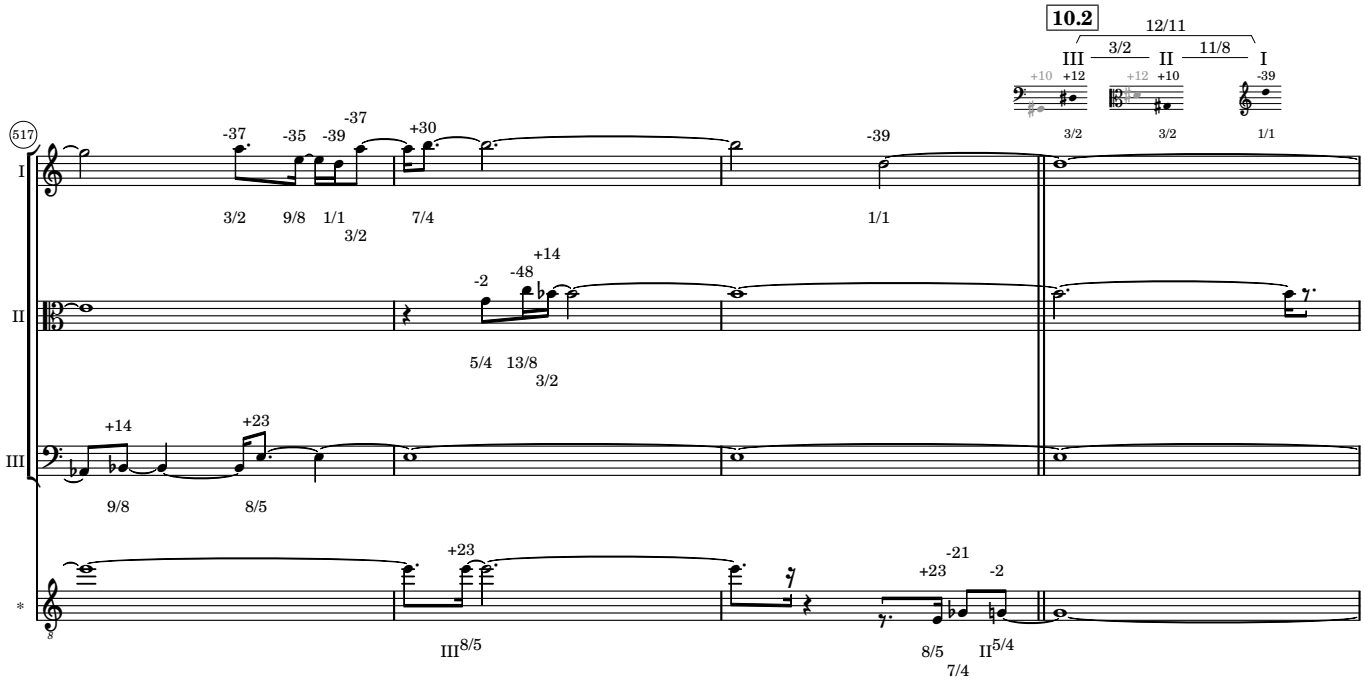

8.4
1/1
III 1/1 II 1/1 I
481
I
II
III
*
485
489
9.1
3/2
-41-

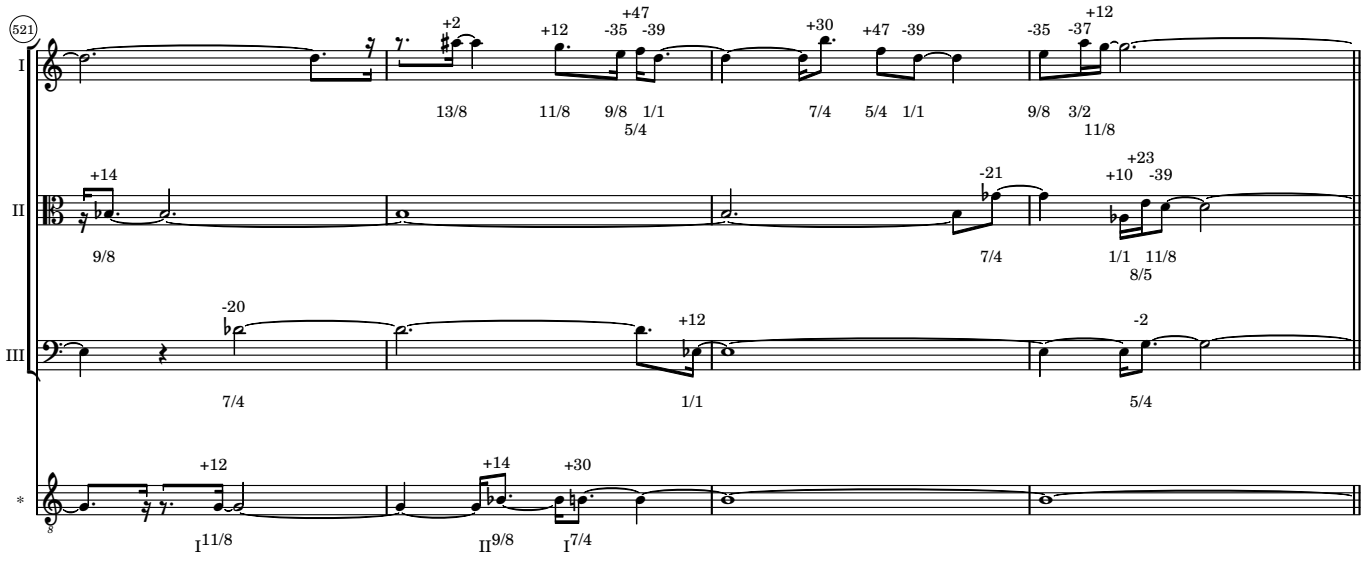

493
I
II
III
*
497
501
9.2
9.3

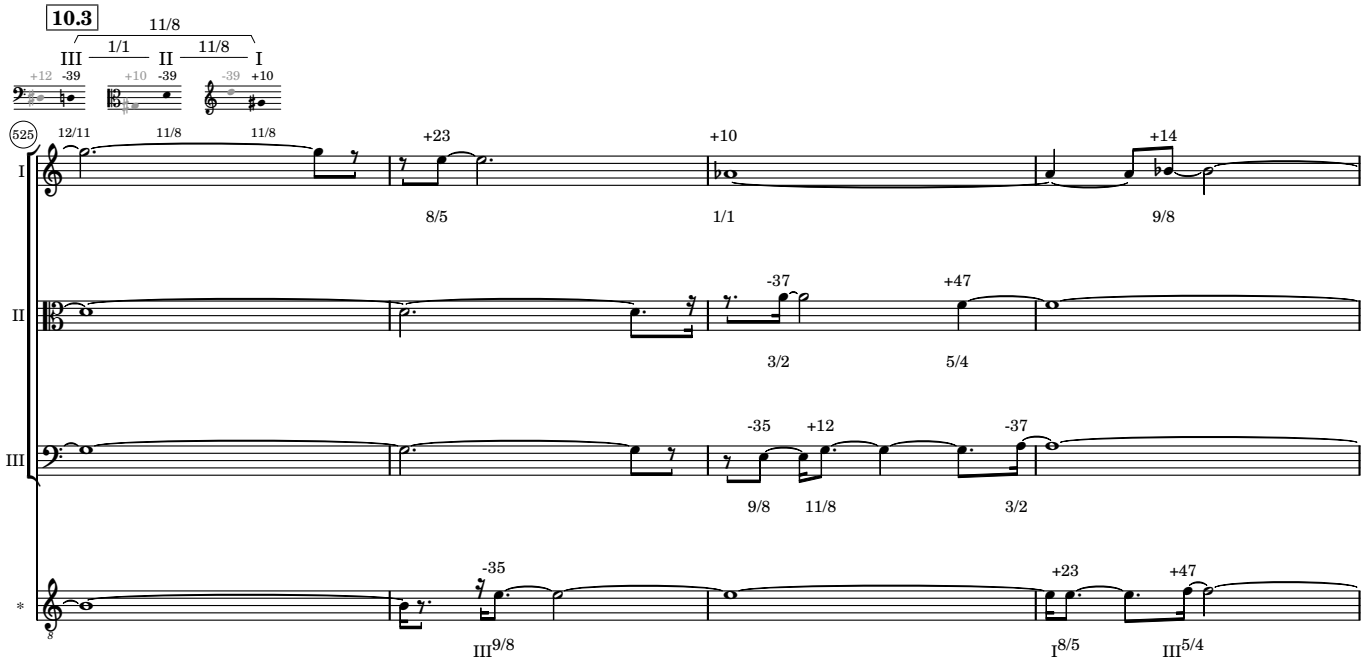

9.4
1/1
III 1/1 II 1/1 I
505
I
II
III
*
-25 +46 -43
7/4 13/8 11/8
+23 -23 +12 -41 +10 +8 +12
6/5 7/4 9/8 11/8 3/2 1/1 9/8
-4 +10 -21
5/4 1/1 7/4
-39 +46
11/8 I13/8

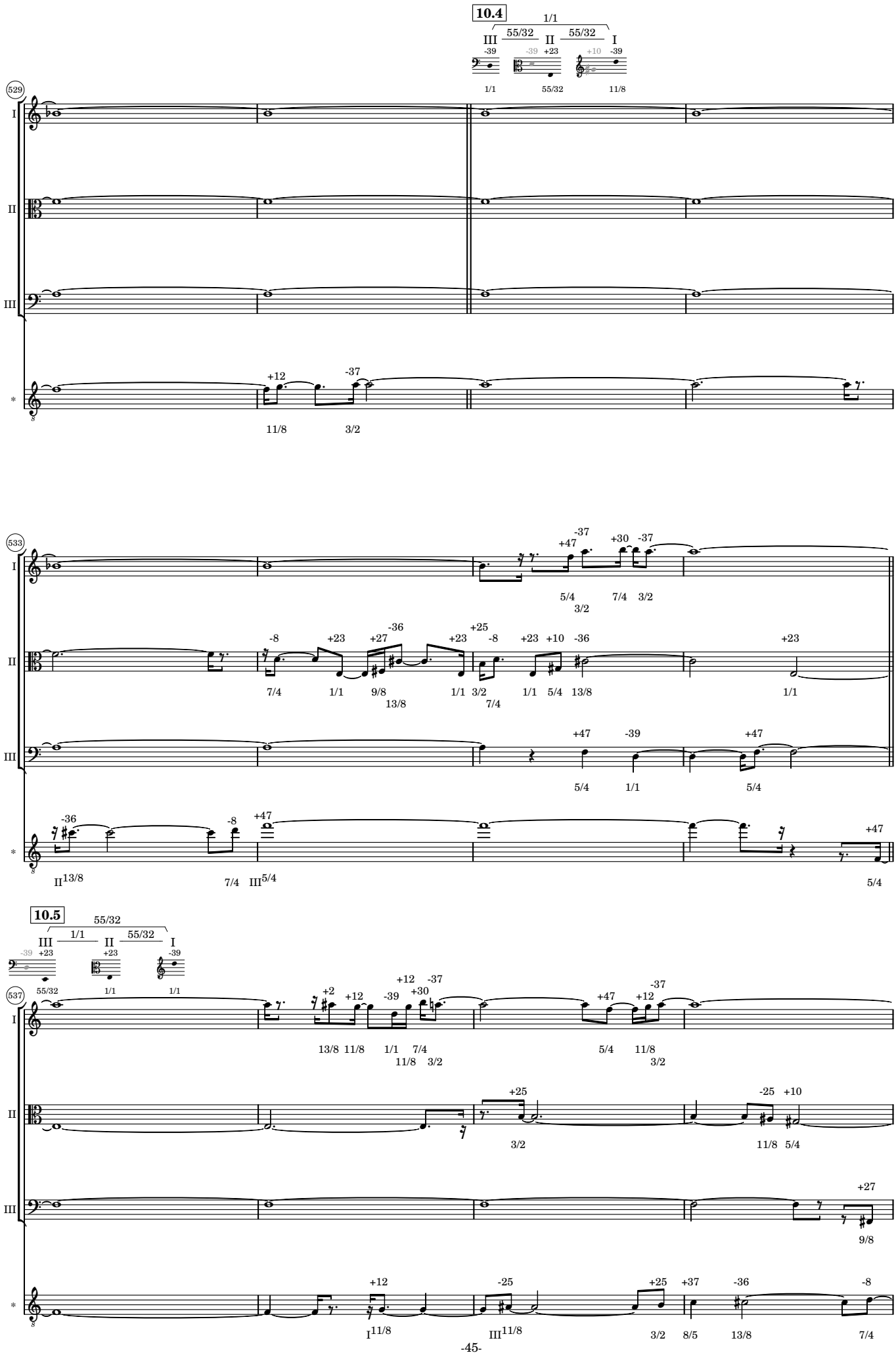
509
I
II
III
*
+10 -4 -21 +12 +10 +10 +23 -21 +14 +10 -39
1/1 5/4 7/4 1/1 8/5 7/4 3/2 1/1 9/8 1/1 11/8
-39 +10 +12 +14
11/8 1/1 3/2 9/8
-39 +12 +10 +23 +10 +12 -21 +14 -4 -39
11/8 3/2 1/1 8/5 1/1 3/2 7/4 9/8 5/4 11/8
-4 -21 +12 +14 -39 +10 -21 -4 +10 +23 +12 -21 +14
III5/4 7/4 3/2 9/8 11/8 1/1 7/4 5/4 1/1 8/5 3/2 7/4 9/8

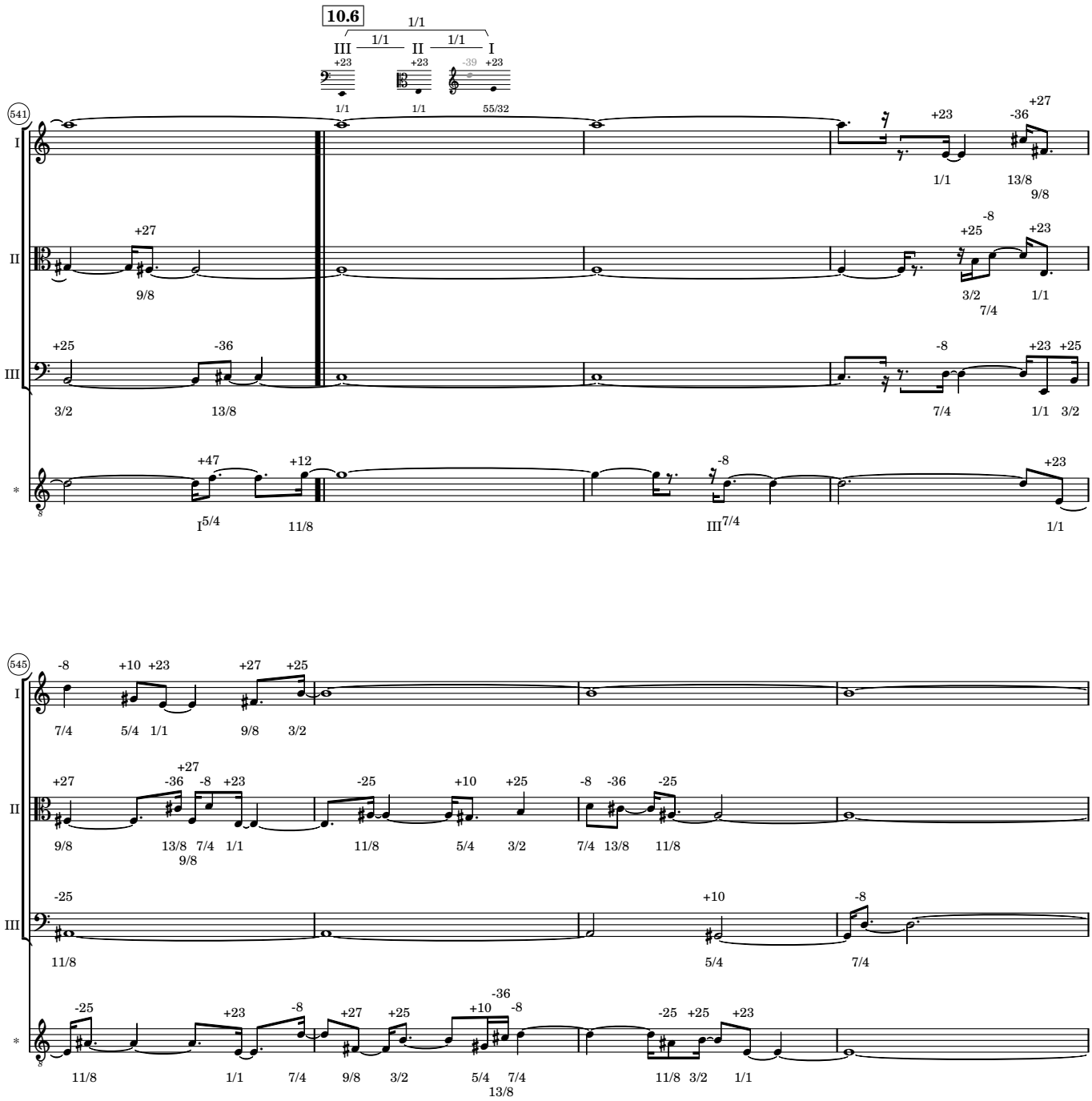

10.1
11/8
III 3/2 II 12/11 I
513
I
II
III
*
-4 +12
5/4 11/8
+10 +23
1/1 8/5
+10
1/1
-4 +10 +12 -39 -35
5/4 1/1 3/2 11/8 I9/8

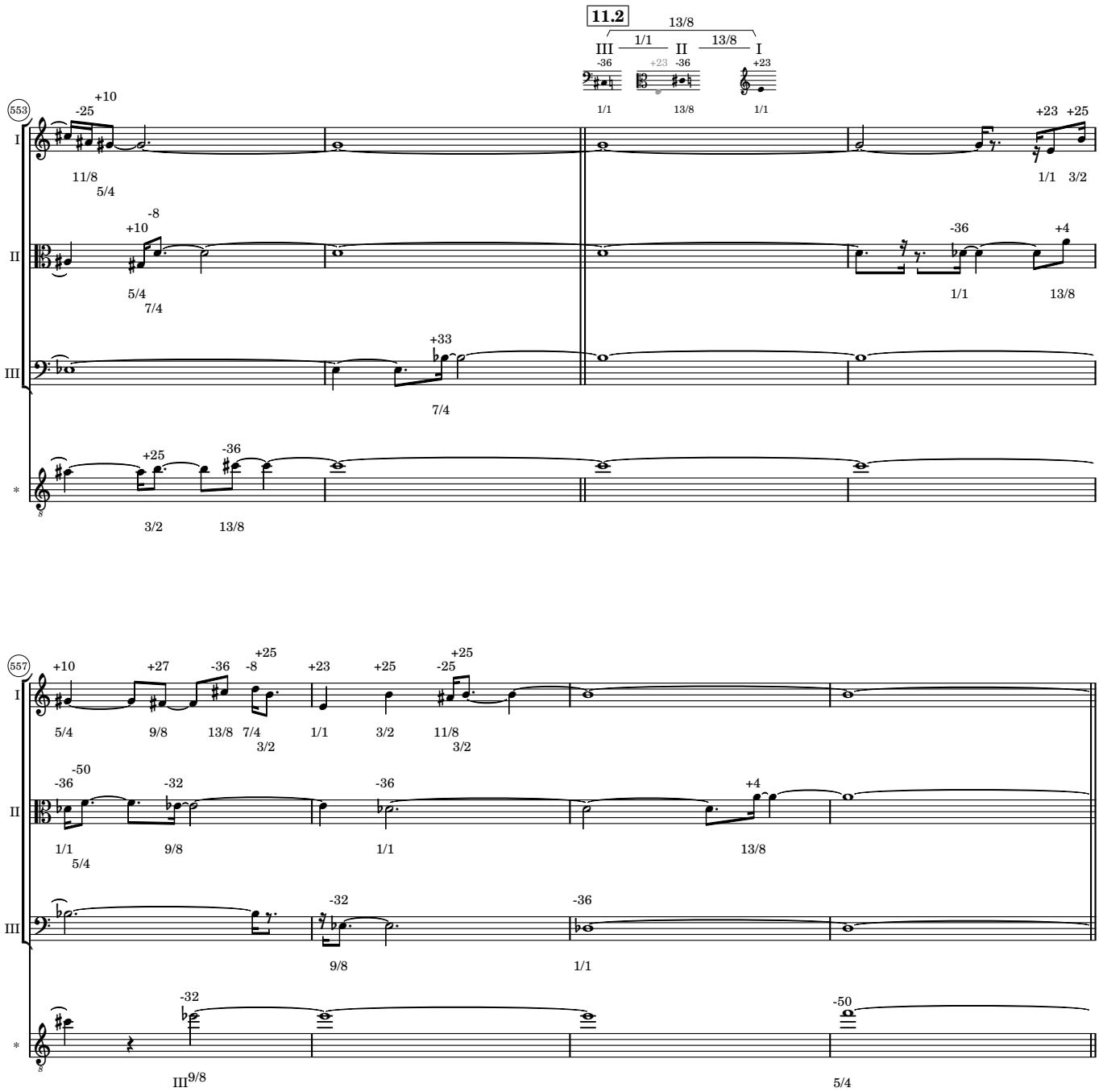

10.2
12/11
III 3/2 II 11/8 I
517
I
II
III
*

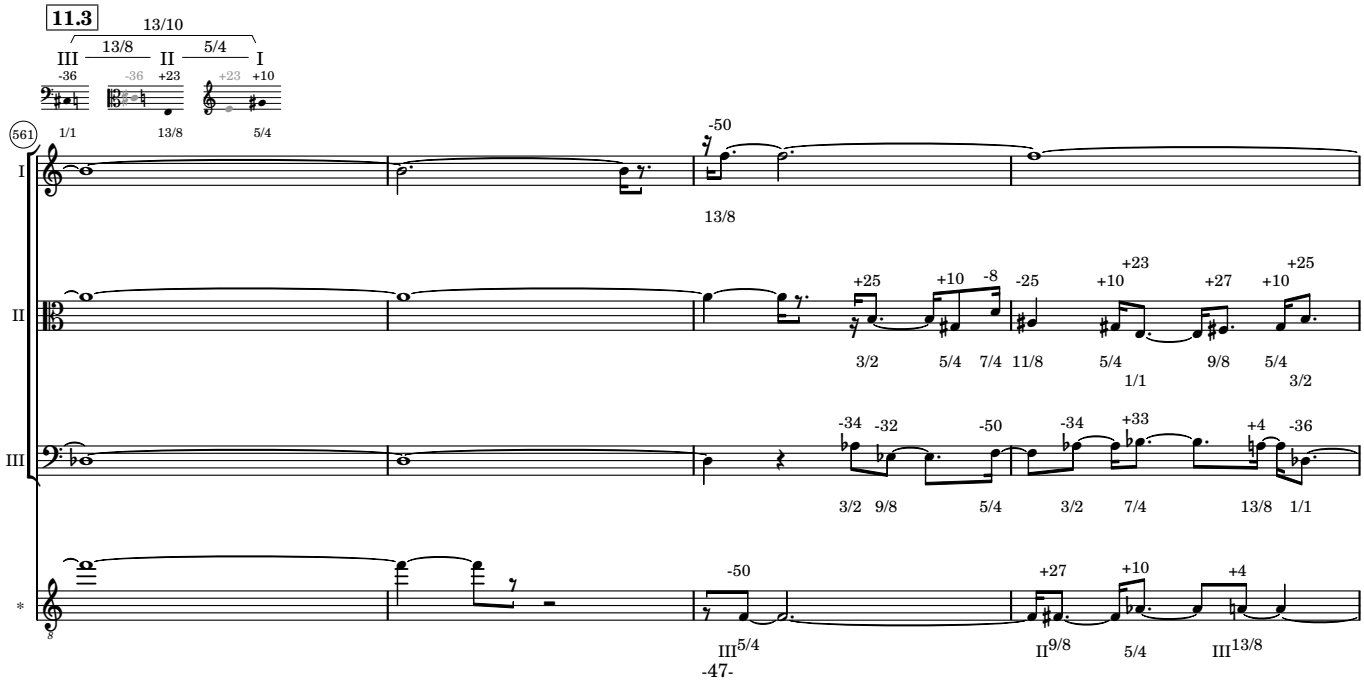

521
I
II
III
*

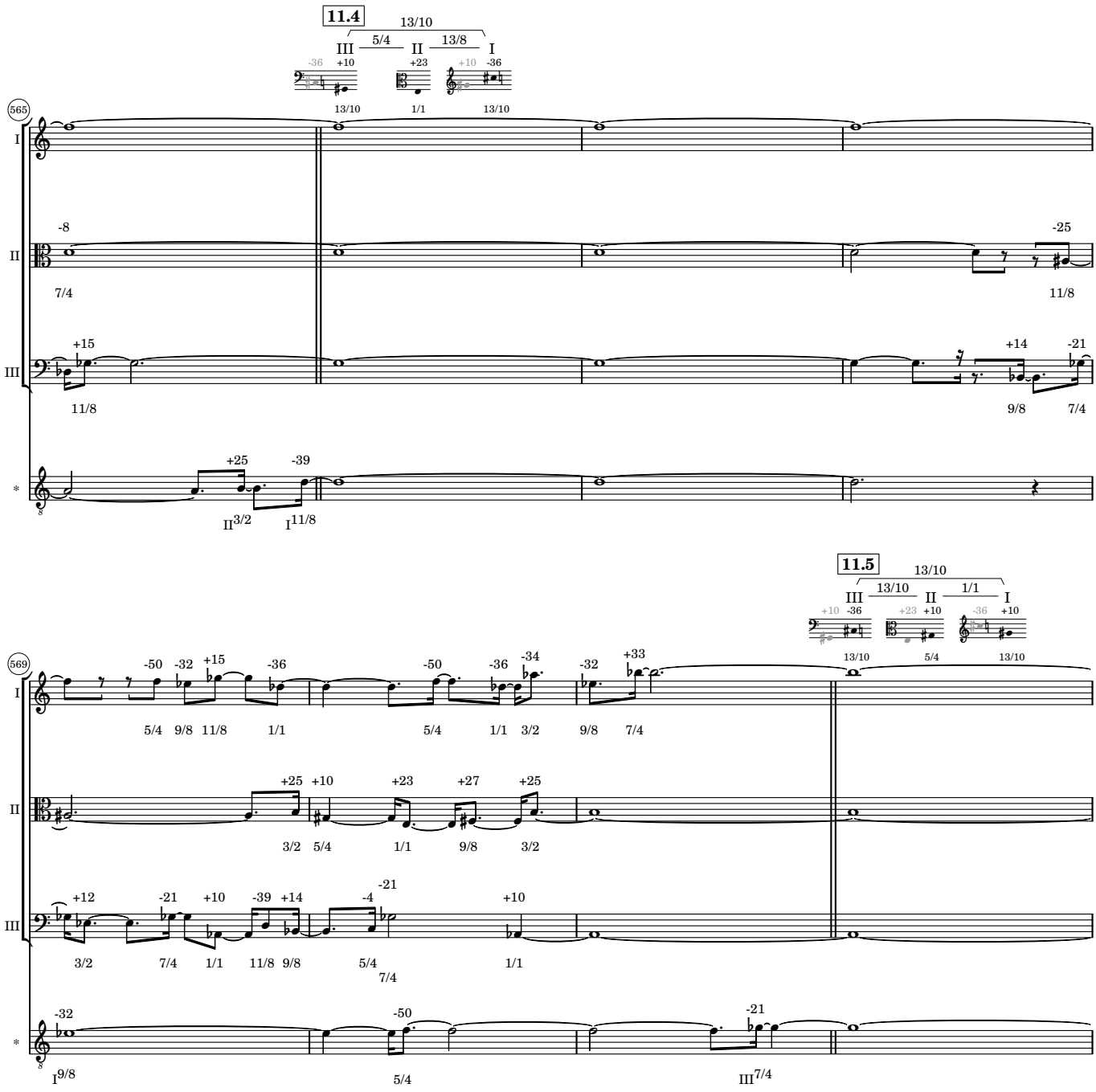

10.3
11/8
III 1/1 II 11/8 I
525
I
II
III
*

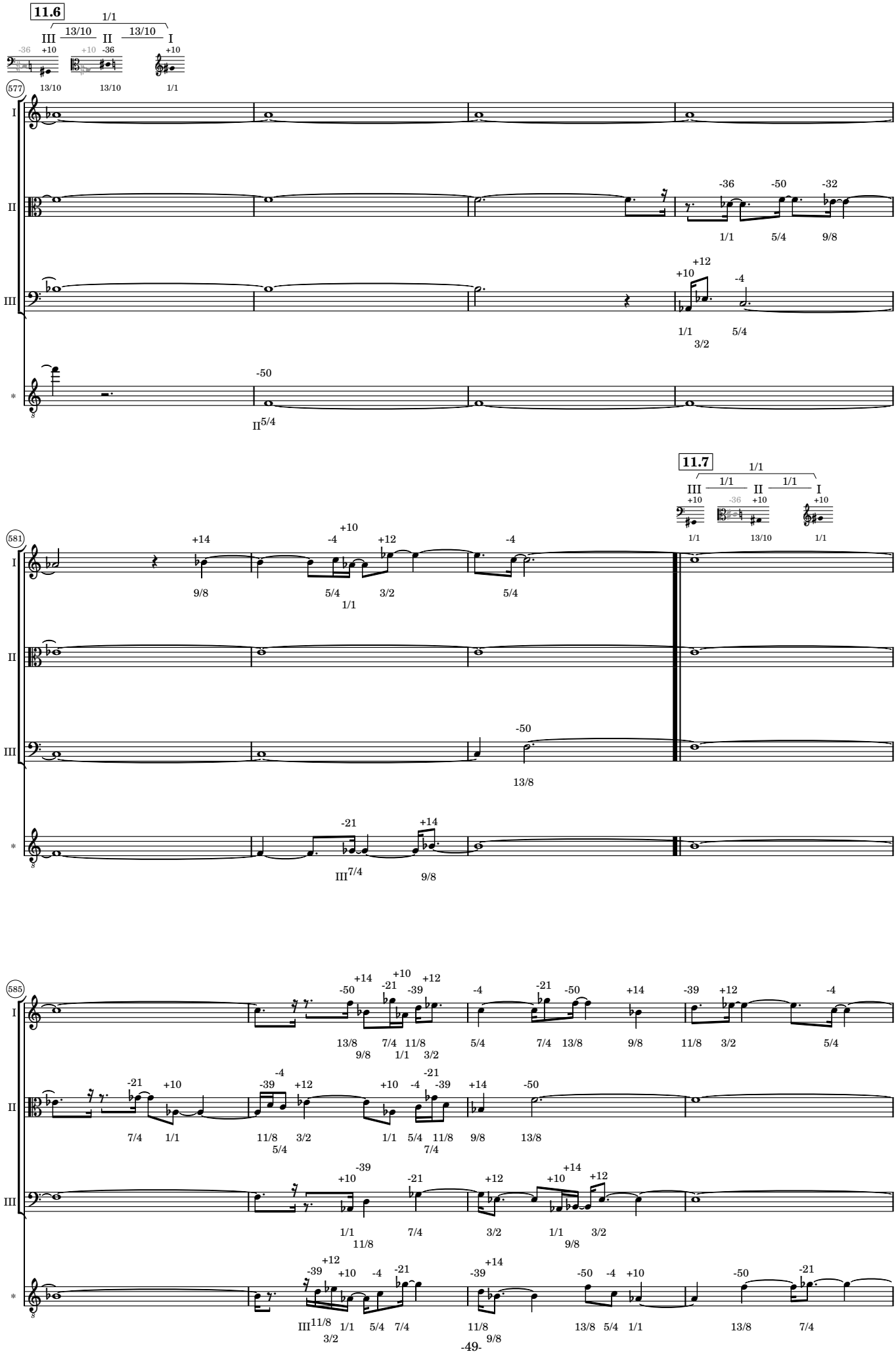

10.4
1/1
III 55/32 II 55/32 I
529
533
10.5
55/32
III 1/1 II 55/32 I
537

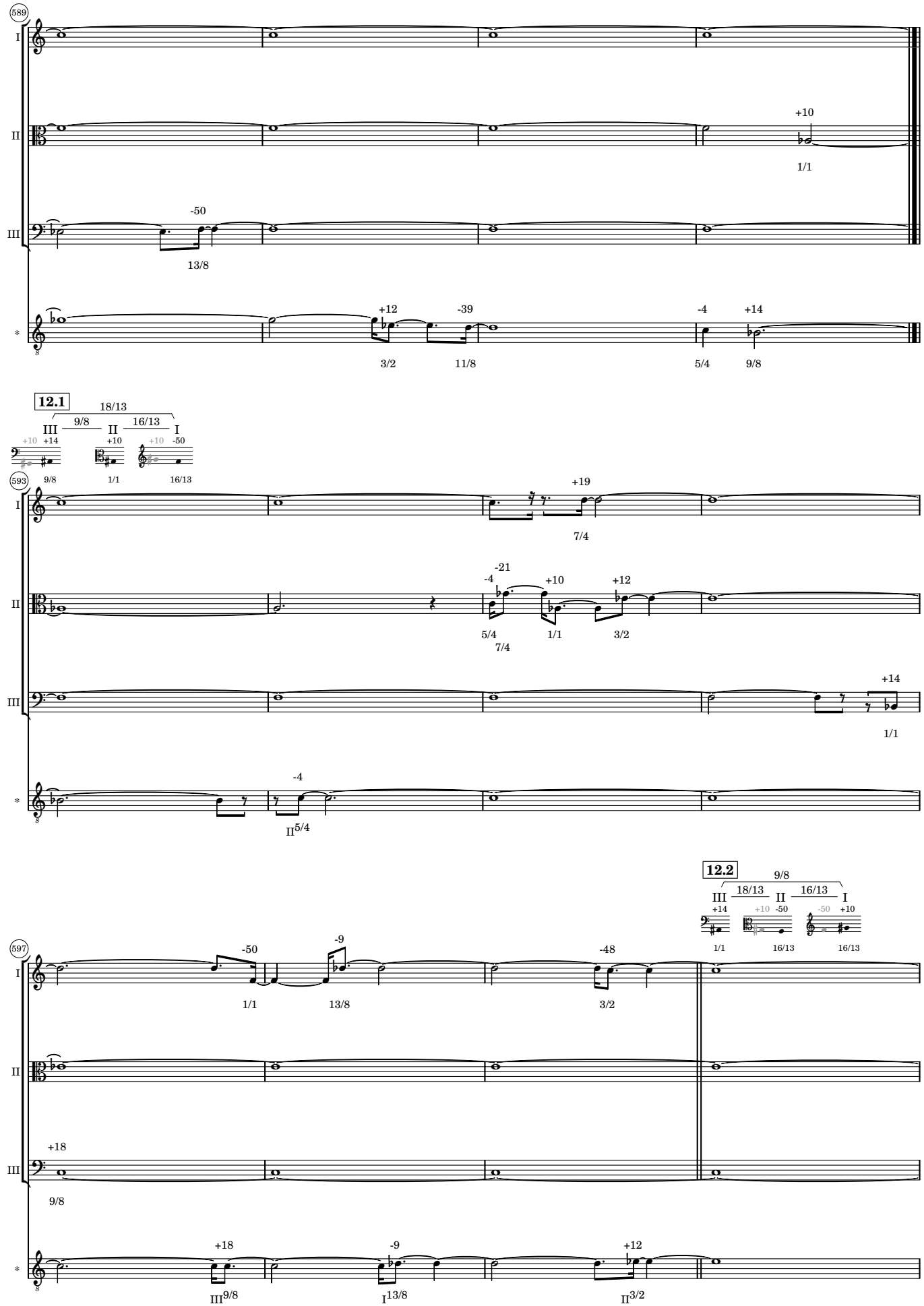

10.6
1/1
III 1/1 II 1/1 I
541
I
II
III
*
545
549
11.1
13/8
III 13/8 II 1/1 I

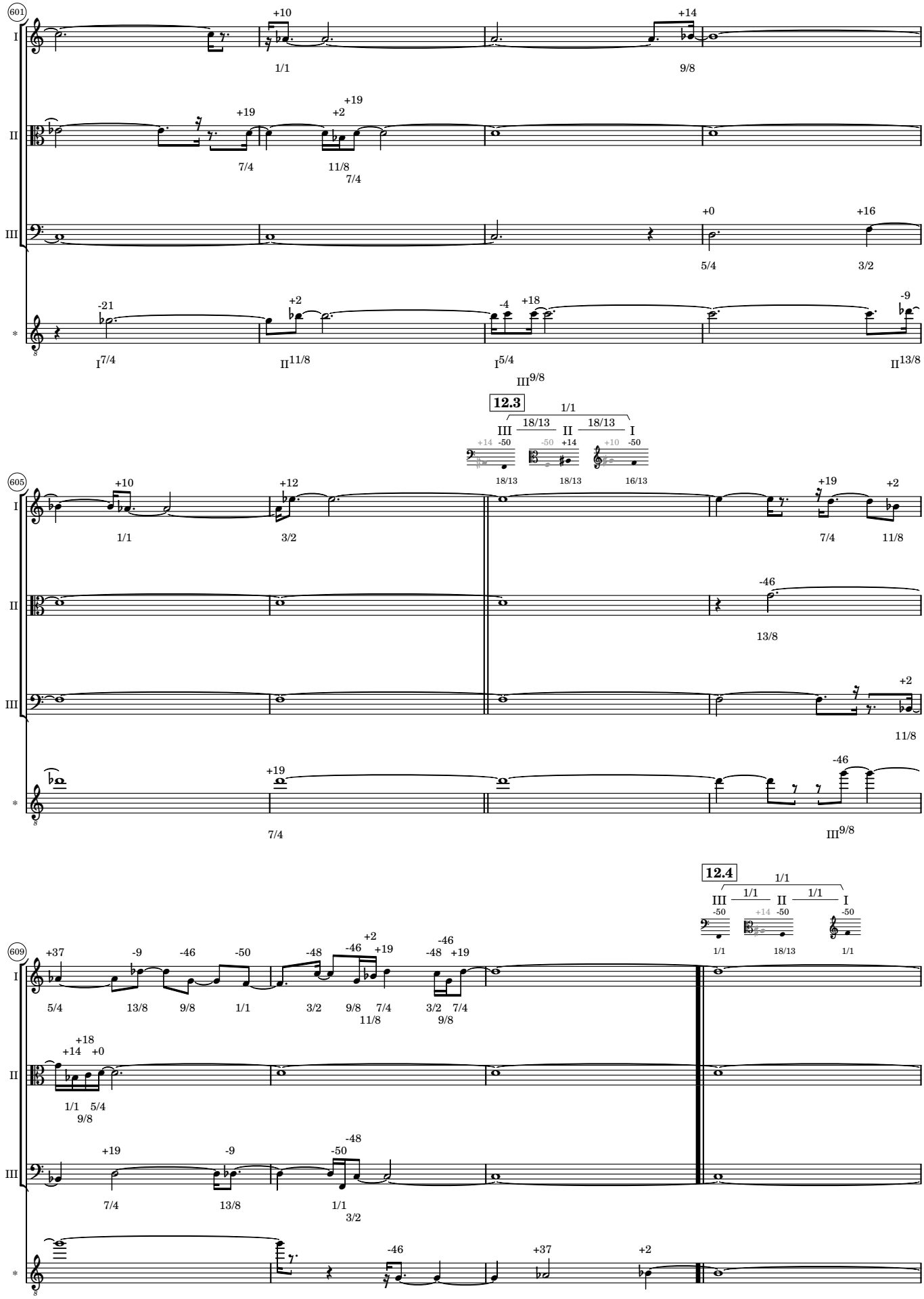

11.2
13/8
III 1/1 II 13/8 I
553
I
II
III
*
11/8
5/4
7/4
3/2
13/8
1/1
557
9/8
3/2
11/8
9/8
5/4
III9/8
5/4

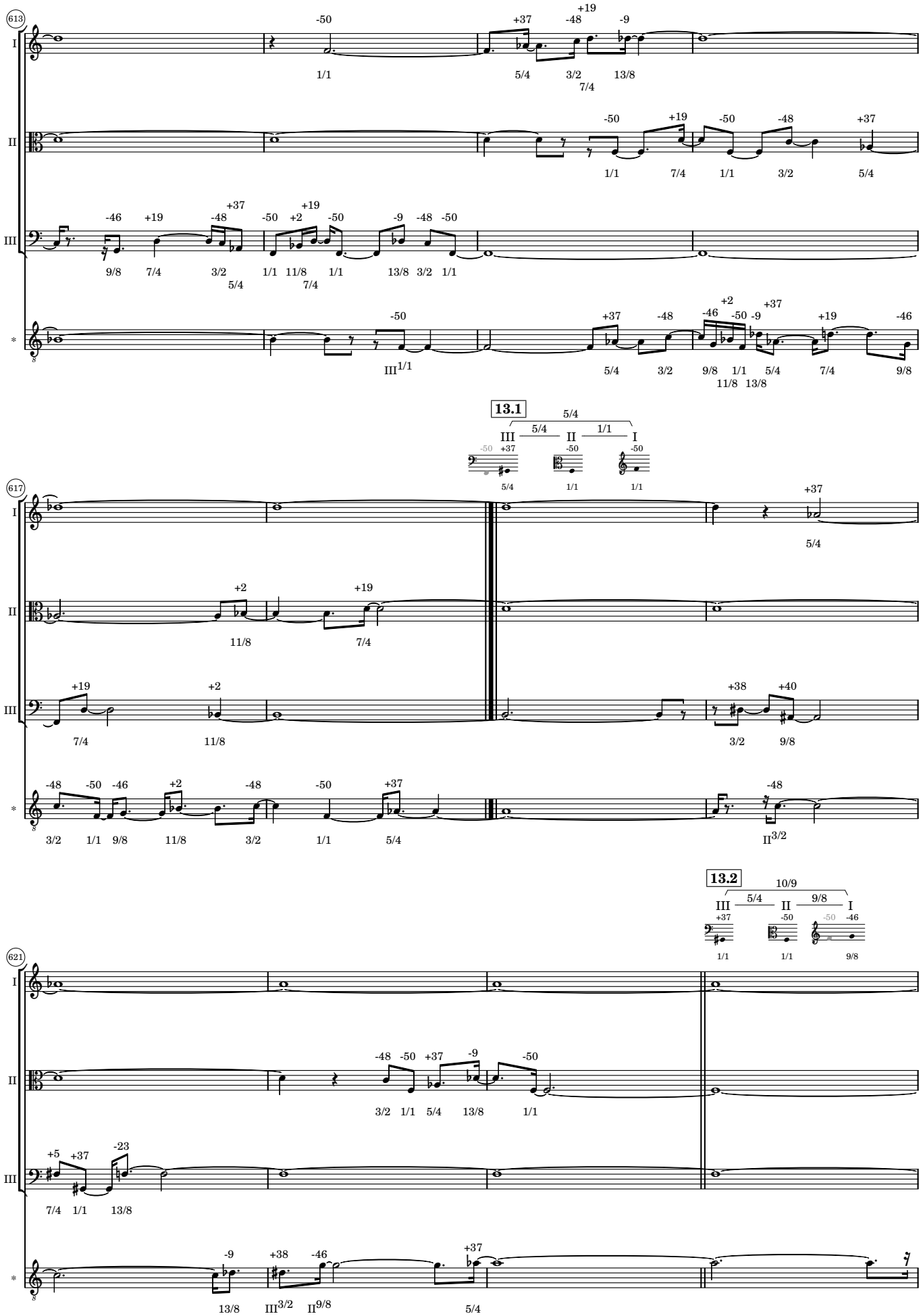

11.3
13/10
III 13/8 II 5/4 I
561
I
II
III
*
13/8
3/2
5/4
7/4
11/8
5/4
1/1
9/8
5/4
3/2
9/8
5/4
7/4
13/8
1/1
III5/4
II9/8
5/4
III13/8

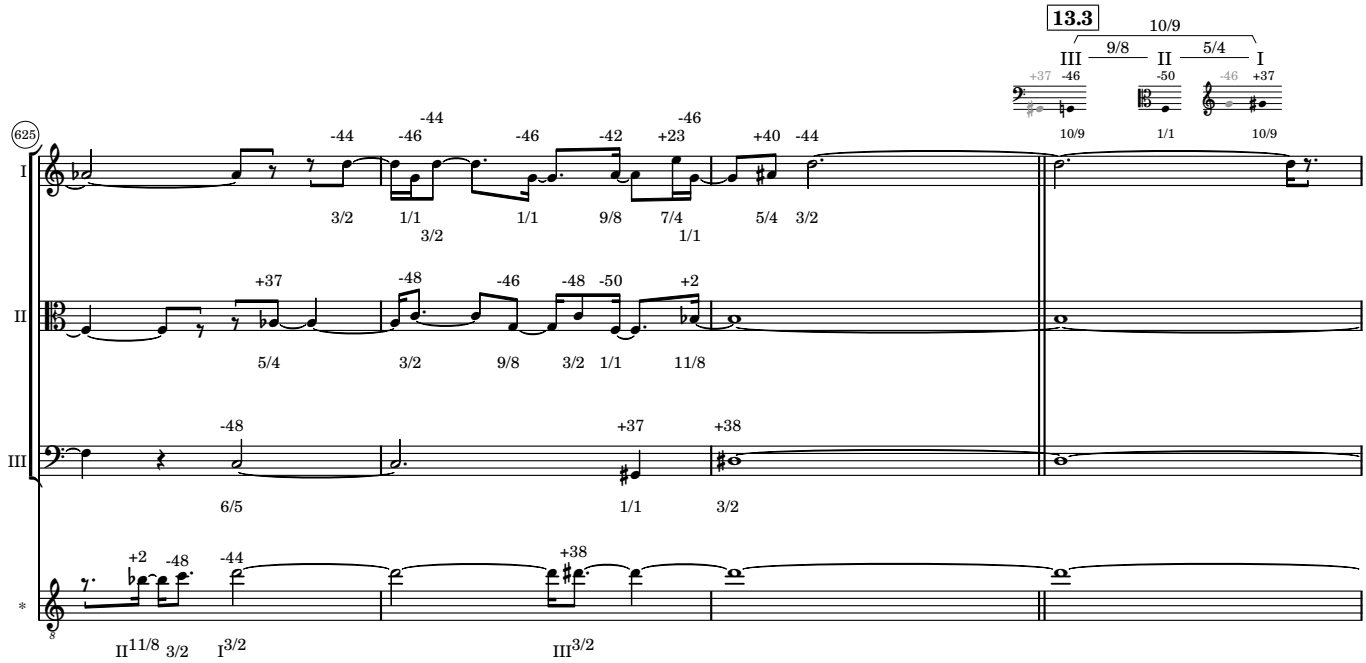

11.4
13/10
III 5/4 II 13/8 I
565
I
II
III
*
11.5
569
573

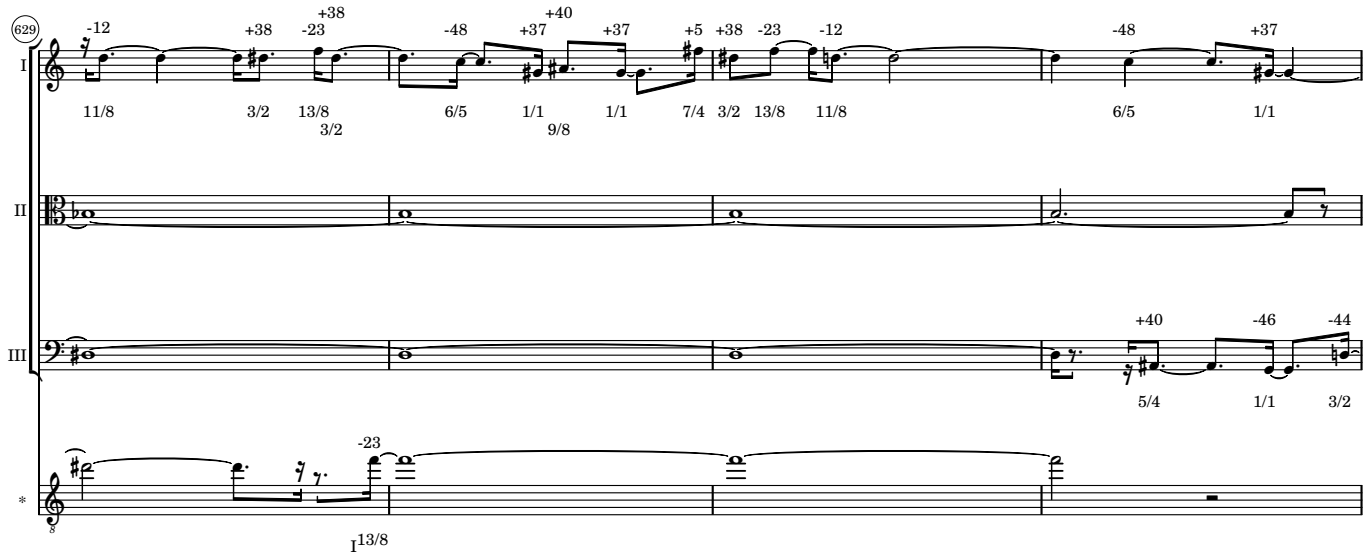

11.6
11.7
577
581
585
I
II
III
*

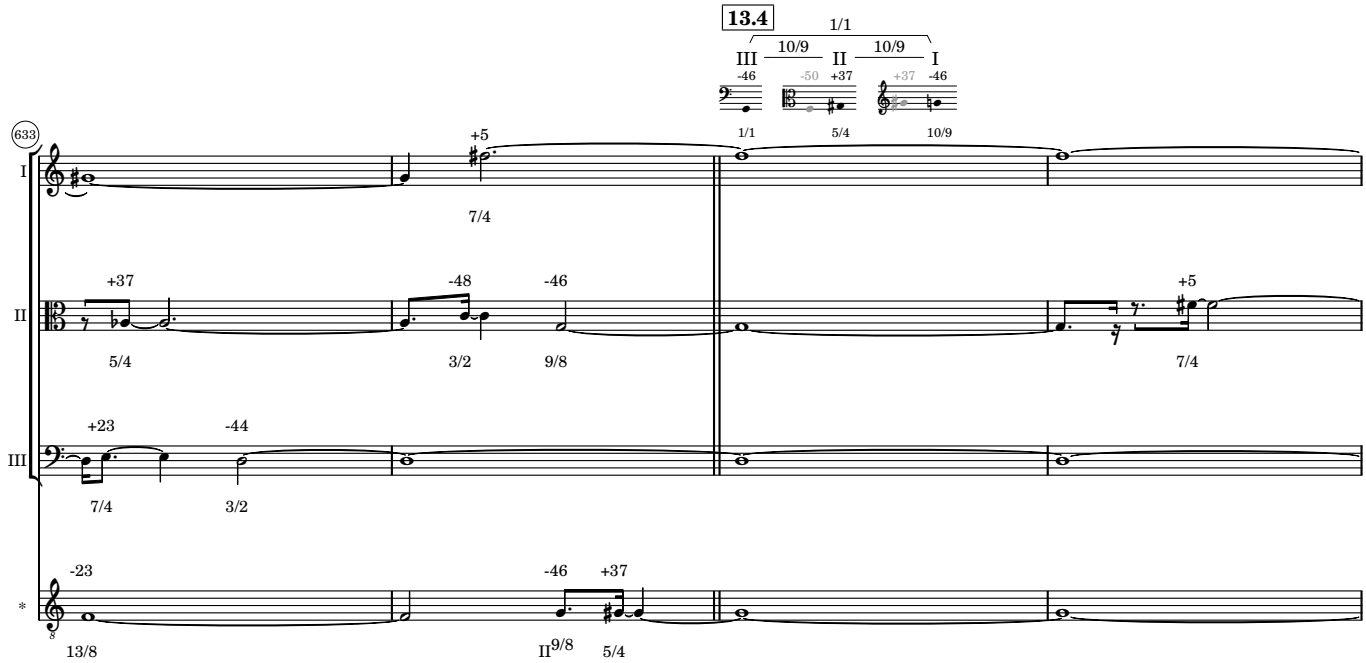

589
I
II
III
*
+10
1/1
-50
13/8
+12
-39
3/2
11/8
-4
+14
5/4
9/8
12.1
18/13
9/8
16/13
III
II
I
+10
+14
+10
+10
-50
9/8
1/1
16/13
593
+19
7/4
-21
-4
+10
+12
5/4
1/1
3/2
7/4
+14
1/1
-4
II5/4
12.2
9/8
18/13
16/13
III
II
I
+14
+10
-50
-50
+10
1/1
16/13
16/13
597
-50
-9
-48
1/1
13/8
3/2
+18
9/8
+18
-9
+12
III9/8
I13/8
II3/2

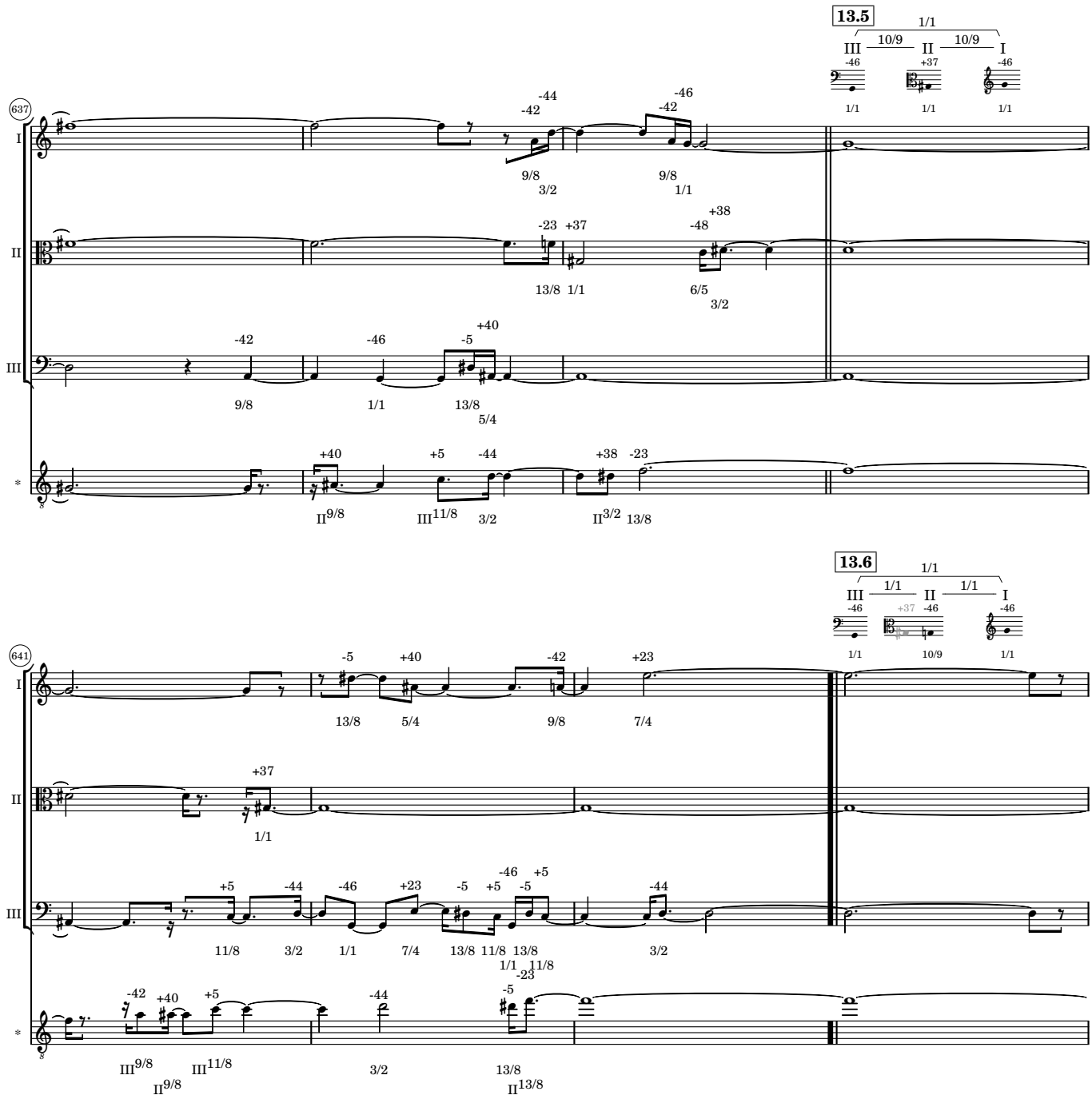

601
I
II
III
*
+10
+14
1/1
9/8
+19
+2
+19
7/4
11/8
7/4
+0
+16
5/4
3/2
-21
+2
-4
+18
-9
I7/4
II11/8
I5/4
III9/8
II13/8
12.3
1/1
III
18/13
II
18/13
I
+14 -50
-50 +14
+10 -50
18/13
18/13
16/13
605
+10
+12
+19
+2
1/1
3/2
7/4
11/8
-46
13/8
+2
11/8
+19
-46
7/4
III9/8
12.4
1/1
III
1/1
II
1/1
I
-50
+14 -50
-50
1/1
18/13
1/1
609
+37
-9
-46
-50
-48
-46
+2
+19
-48
-46
+19
5/4
13/8
9/8
1/1
3/2
9/8
7/4
11/8
3/2
7/4
9/8
+18
+14
+0
1/1
5/4
9/8
+19
-9
-48
-50
7/4
13/8
1/1
3/2
-46
+37
+2
9/8
5/4
11/8

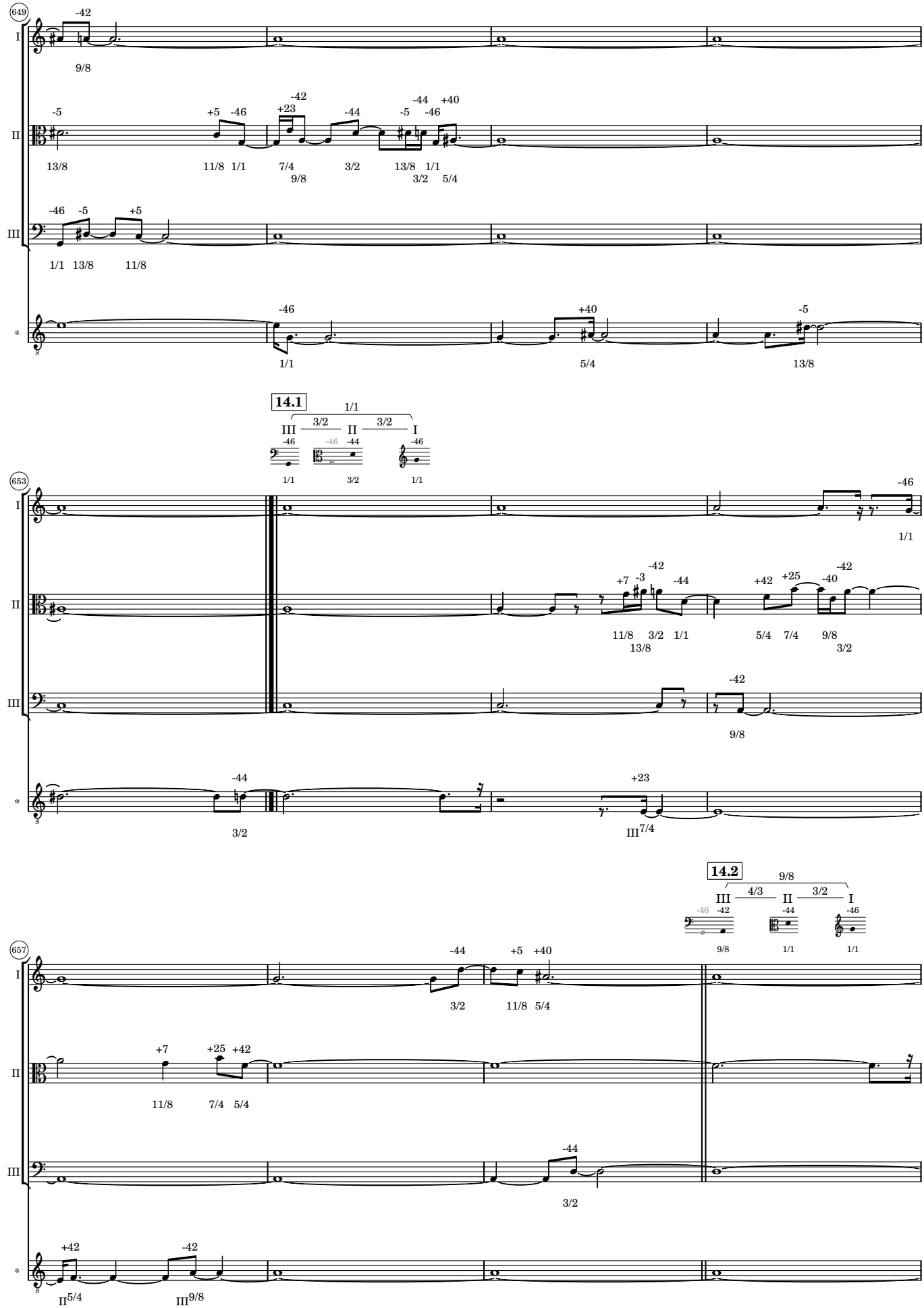

613
I
II
III
*
617
621
13.1
13.2

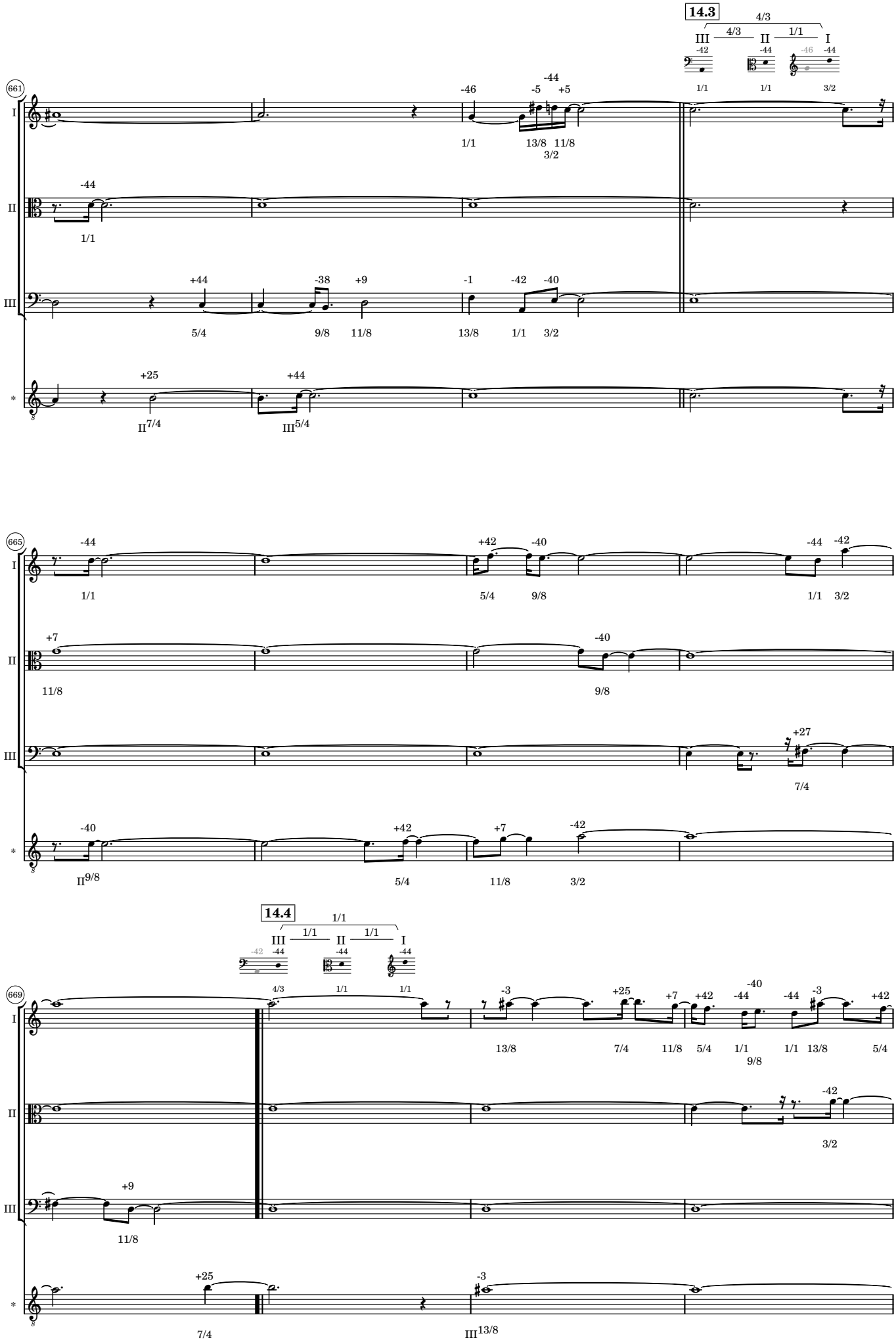

13.3
10/9
III 9/8 II 5/4 I
625
I
II
III
II11/8 3/2 I3/2
III3/2

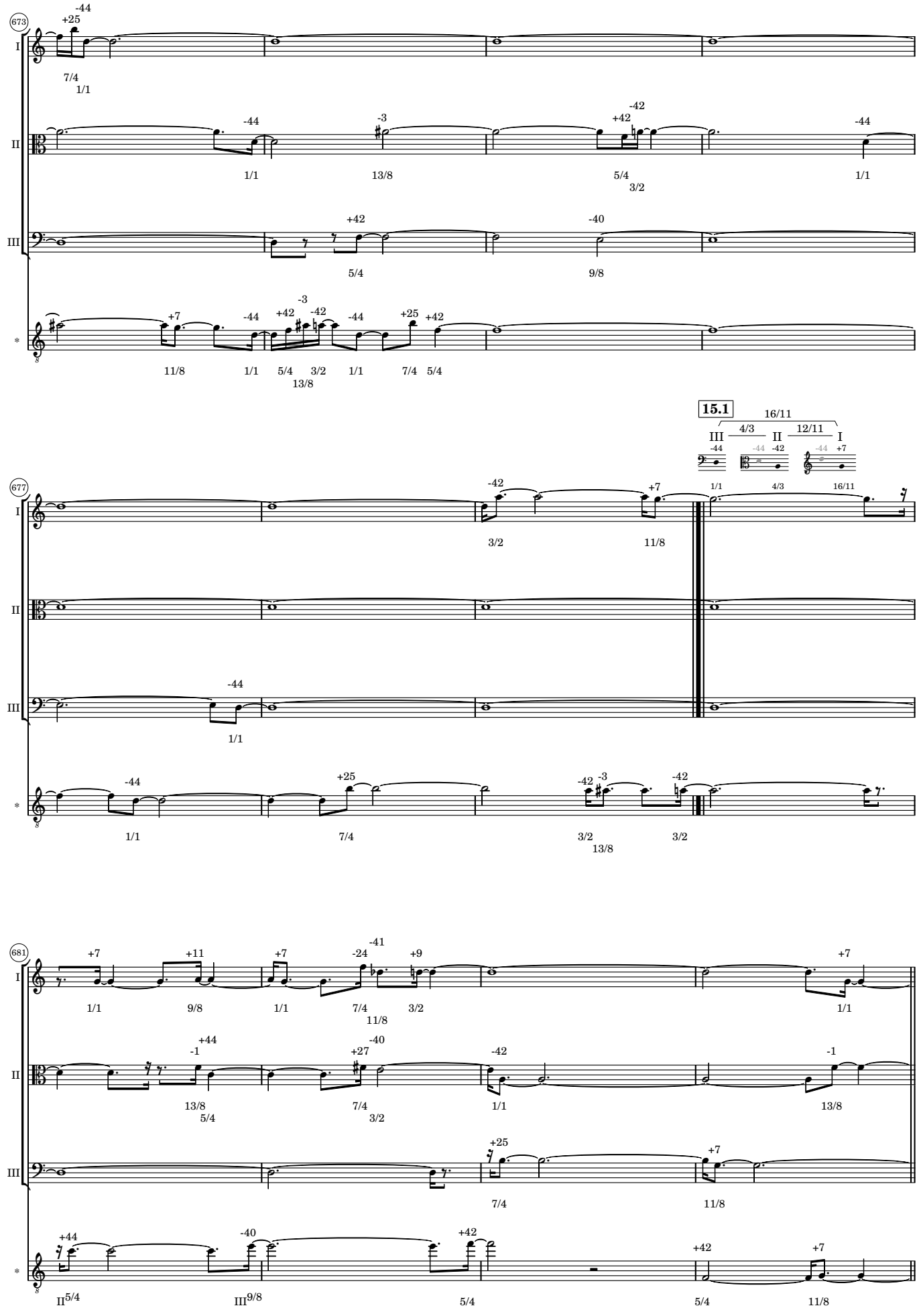

629
I
II
III
I13/8

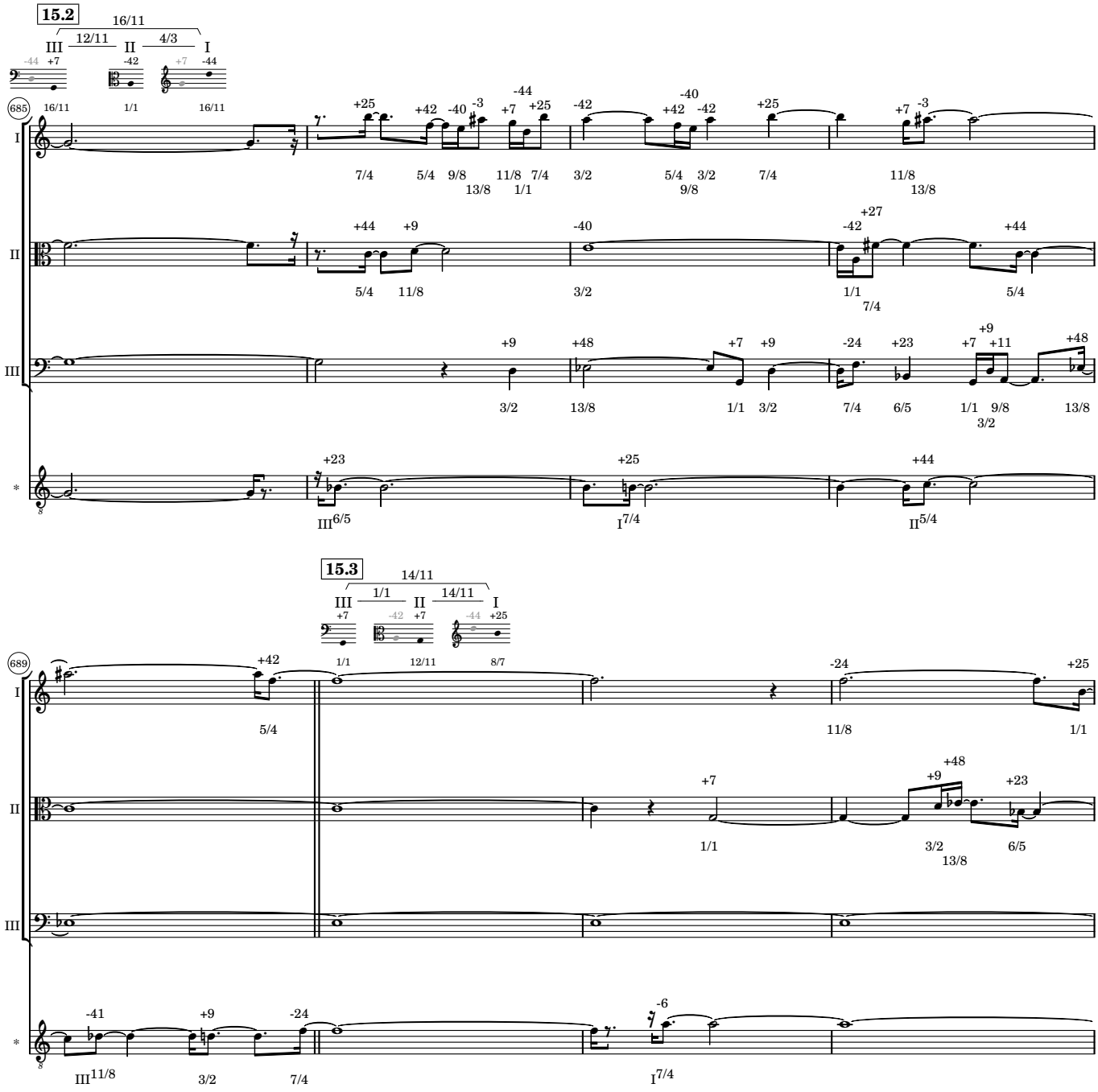

13.4
1/1
III 10/9 II 10/9 I
633
I
II
III
13/8
II9/8 5/4

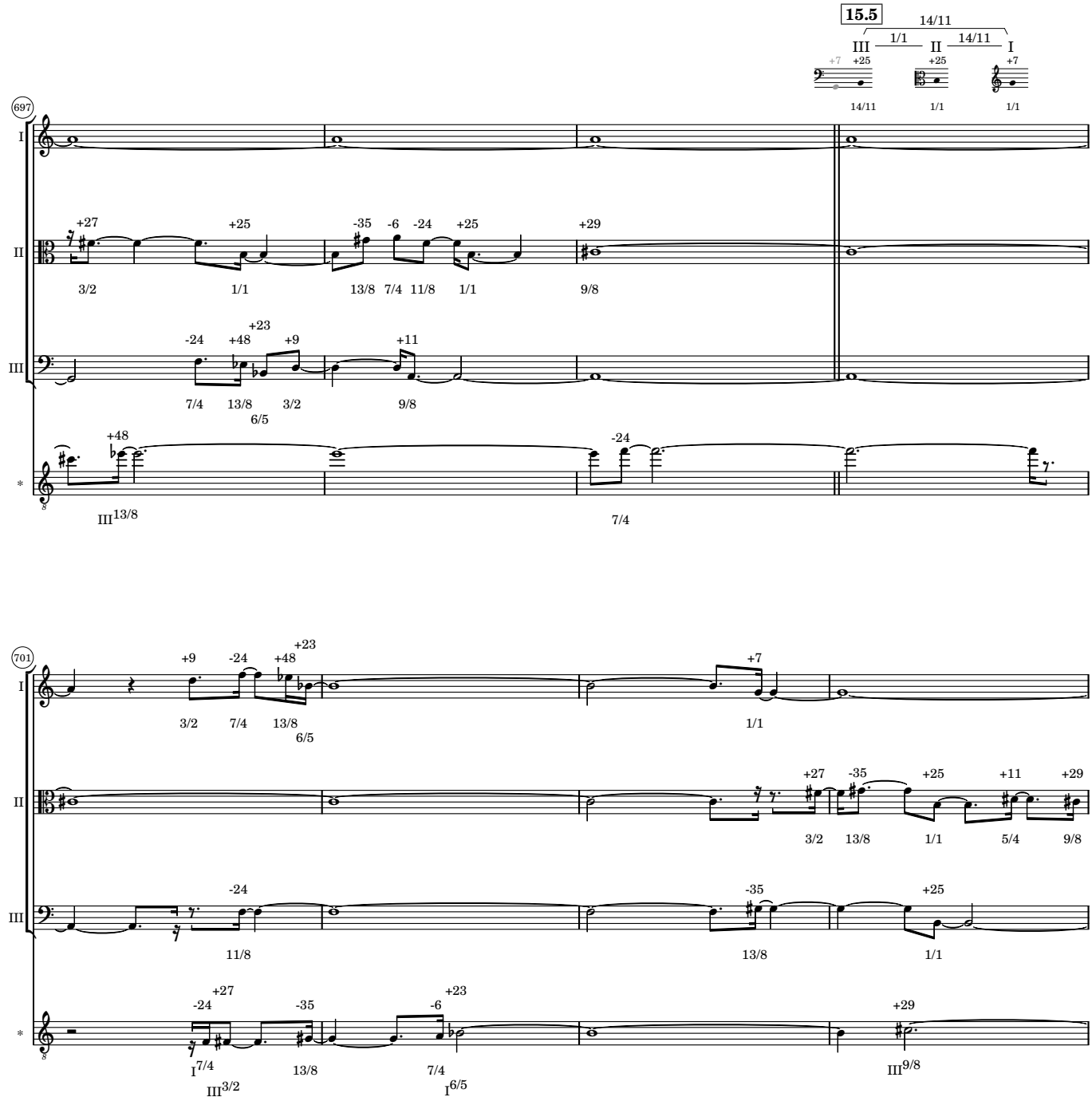

13.5
13.6
637
641
645
I
II
III
*


649
I
II
III
*
-42
9/8
-5
+5 -46
+23
-42
-44
-5 -44 +40 -46
13/8
11/8 1/1
7/4
9/8
3/2
13/8 1/1
3/2 5/4
-46 -5 +5
1/1 13/8
11/8
-46
1/1
+40
5/4
-5
13/8
14.1
1/1
III 3/2 II 3/2 I
-46
-46 -44
-46
1/1
3/2
1/1
653
-46
1/1
+7 -3 -42 -44
+42 +25 -42 -40
11/8 3/2 1/1
13/8
5/4 7/4 9/8
3/2
-42
9/8
-44
3/2
+23
III7/4
14.2
9/8
III 4/3 II 3/2 I
-46 -42
-44
-46
9/8
1/1
1/1
657
-44
+5 +40
3/2
11/8 5/4
+7
+25 +42
11/8
7/4 5/4
-44
3/2
+42
II5/4
-42
III9/8

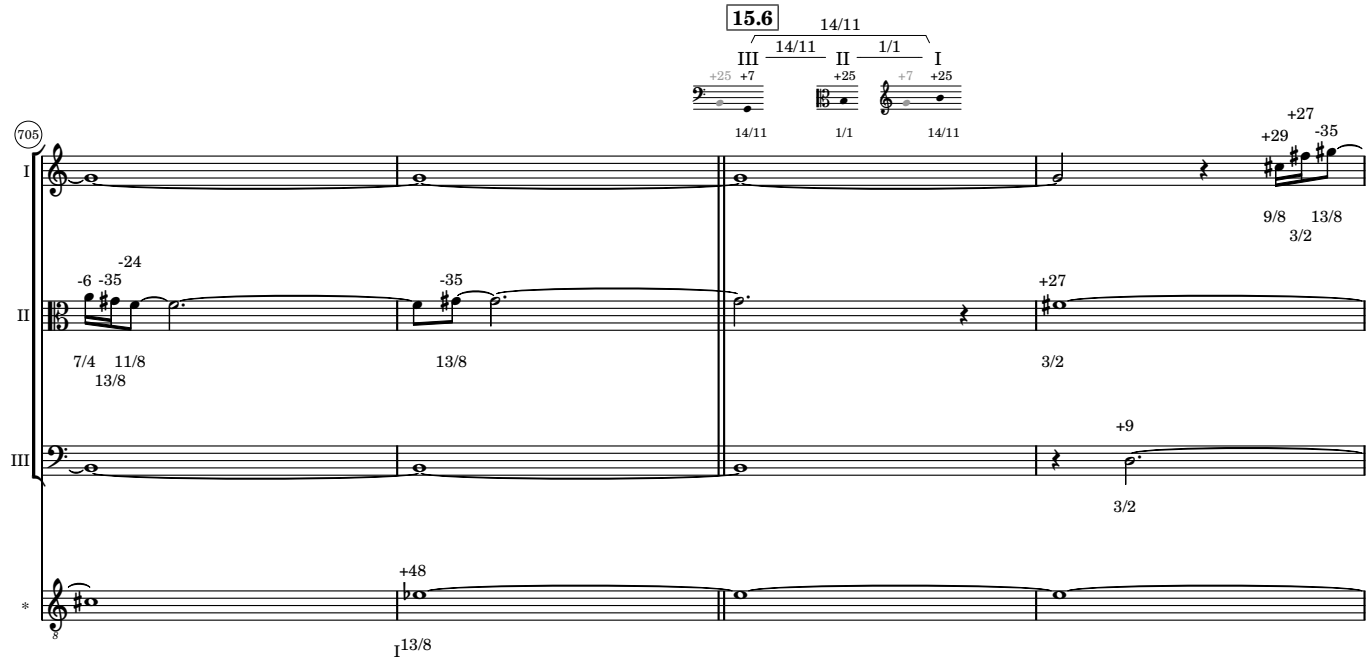

14.3
4/3
III 4/3 II 1/1 I
661
I
II
III
*
665
669
14.4
1/1
III 1/1 II 1/1 I

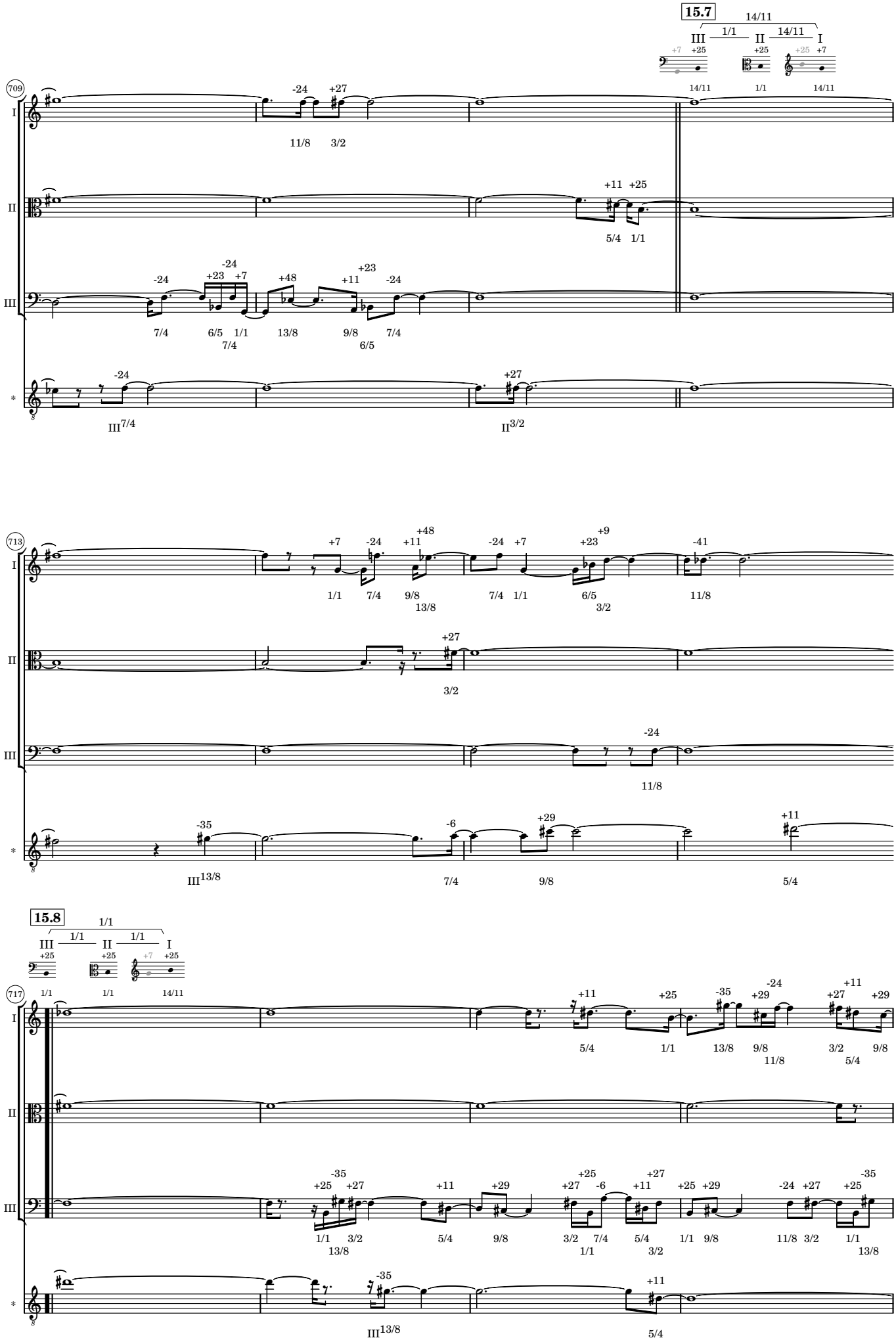

673
677
681
15.1
16/11
4/3
12/11
I
II
III
*

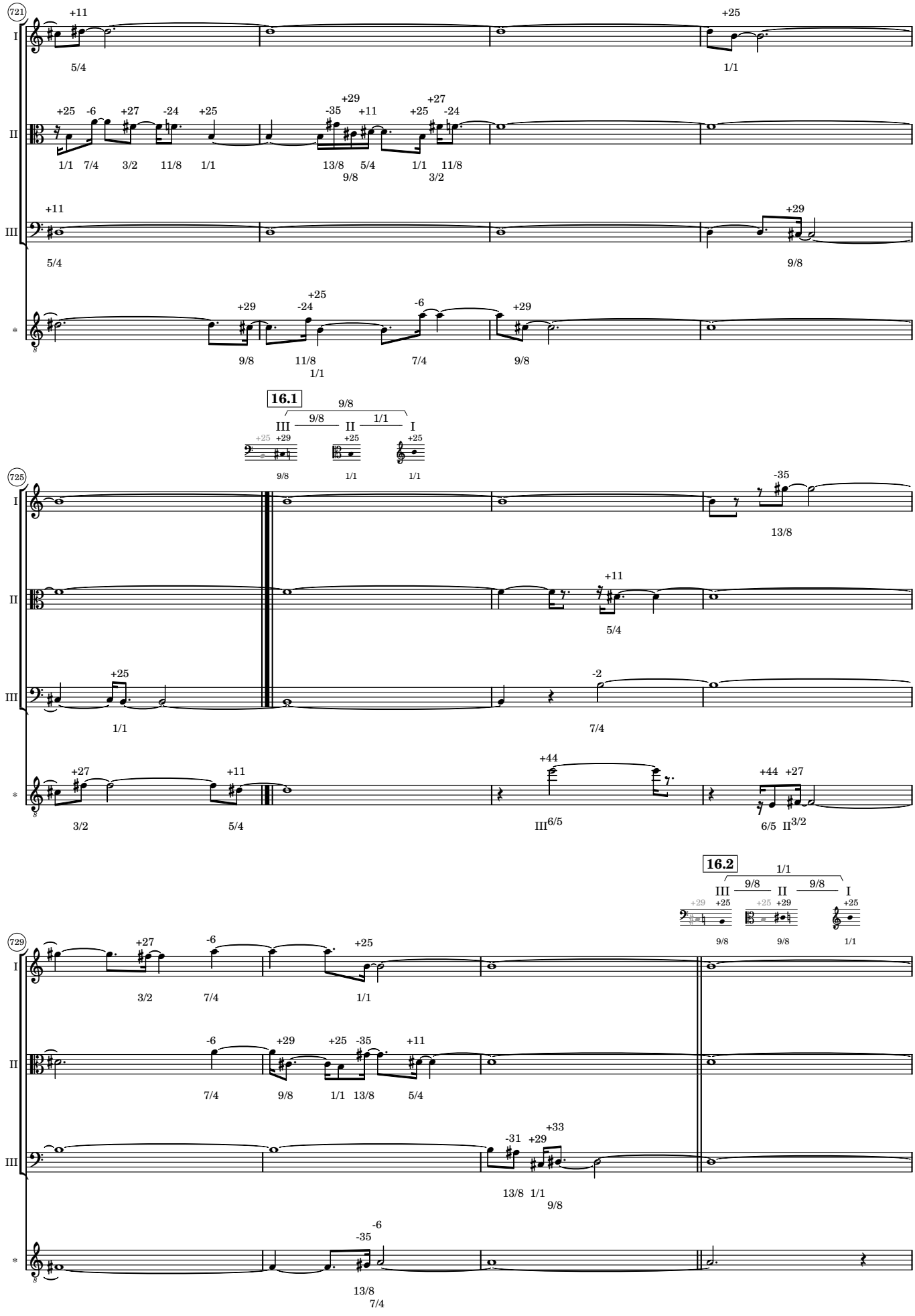

15.2
16/11
III 12/11 II 4/3 I
685
I
II
III
*
III6/5
I7/4
II5/4
15.3
14/11
III 1/1 II 14/11 I
689
III11/8
I7/4
15.4
1/1
III 14/11 II 14/11 I
693
III9/8
II9/8

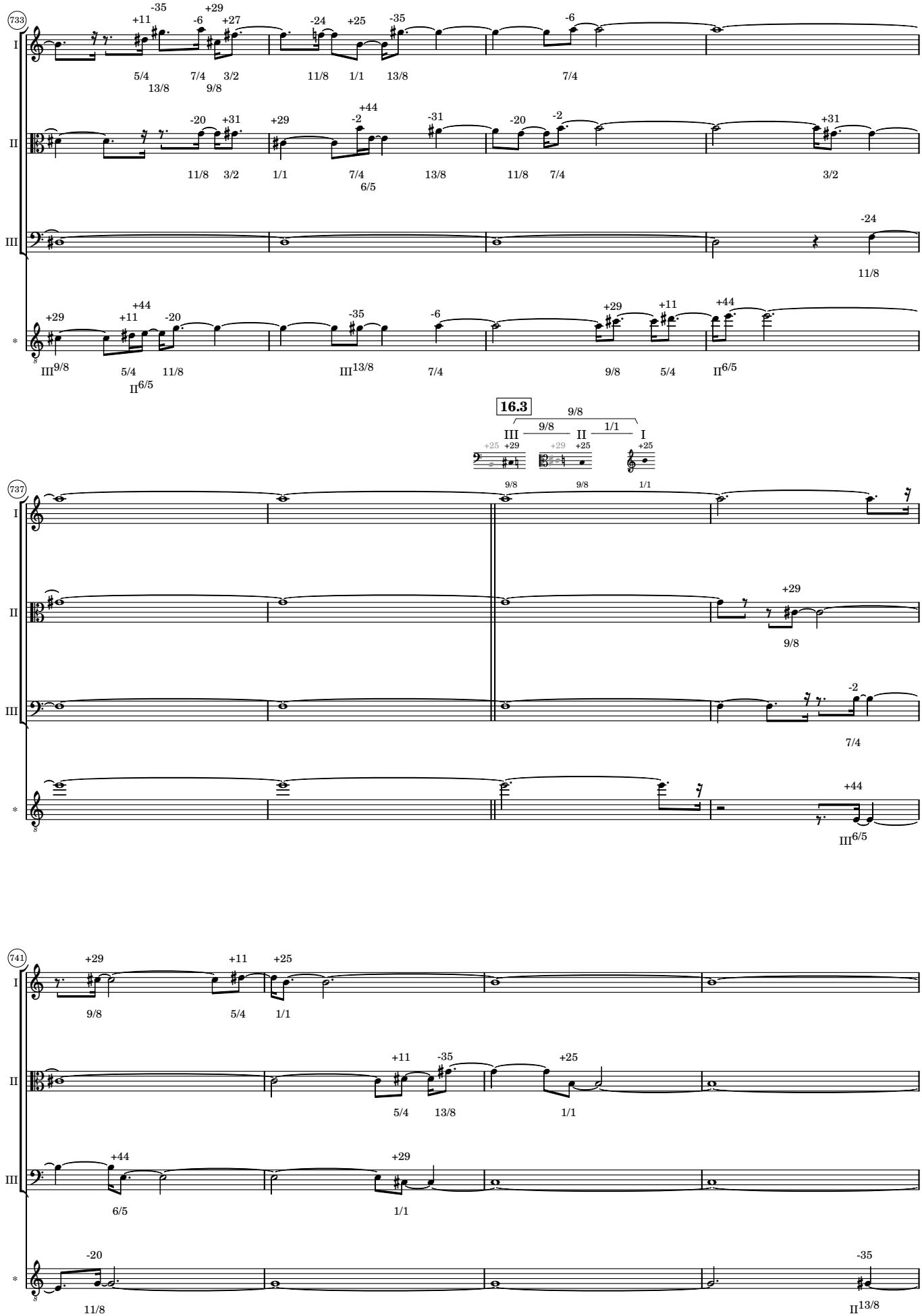

15.5
14/11
III 1/1 II 14/11 I
697
I
II
III
*
+27 +25 -35 -6 -24 +25 +29
3/2 1/1 13/8 7/4 11/8 1/1 9/8
-24 +48 +23 +9 +11
7/4 13/8 3/2 9/8
6/5
+48 -24
III13/8 7/4

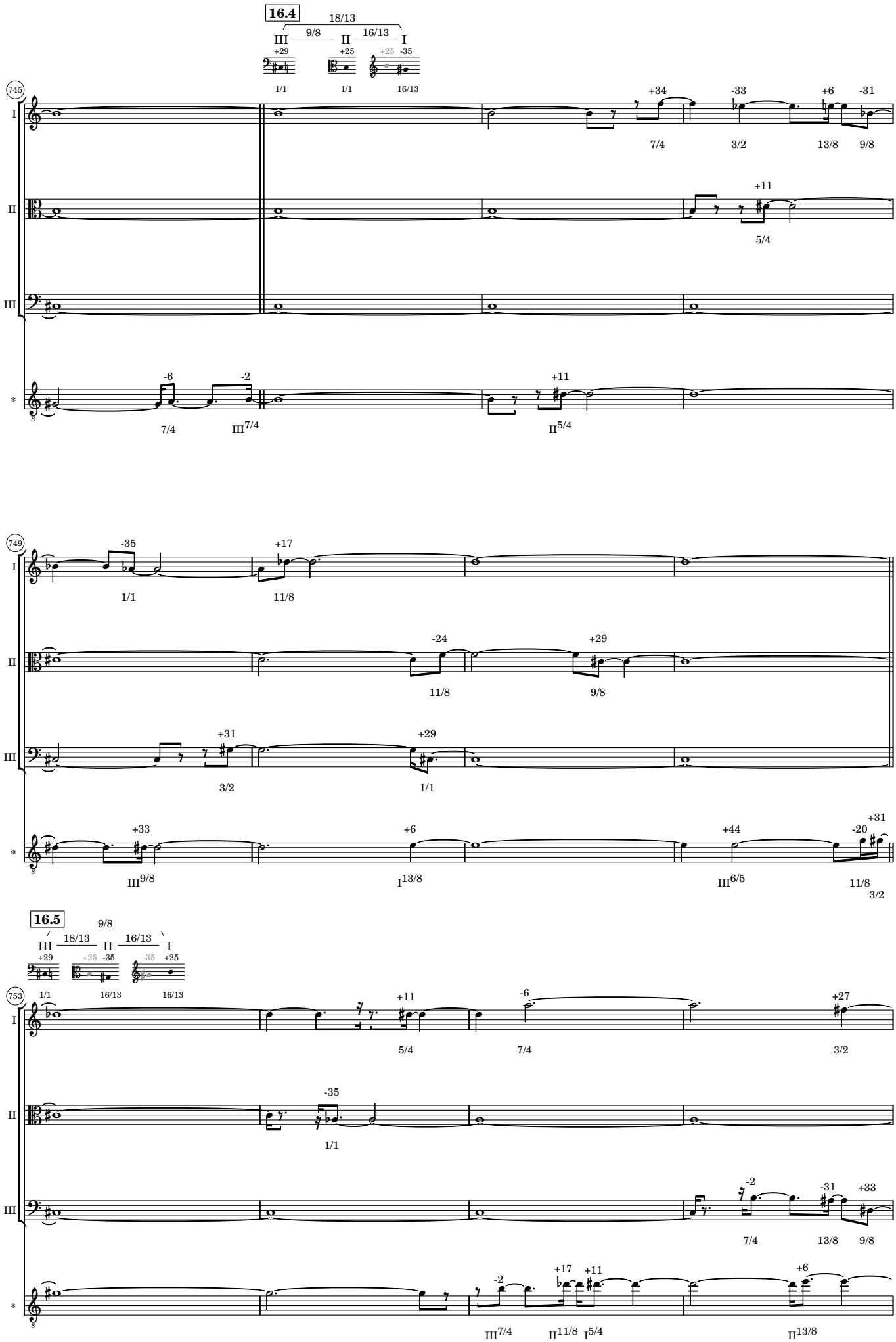
701
I
II
III
*
+9 -24 +48 +23 +7
3/2 7/4 13/8 1/1
6/5
+27 -35 +25 +11 +29
3/2 13/8 1/1 5/4 9/8
-24 -35 +25
11/8 13/8 1/1
-24 +27 -35 -6 +23 +29
I7/4 13/8 7/4 III9/8
III3/2 I6/5

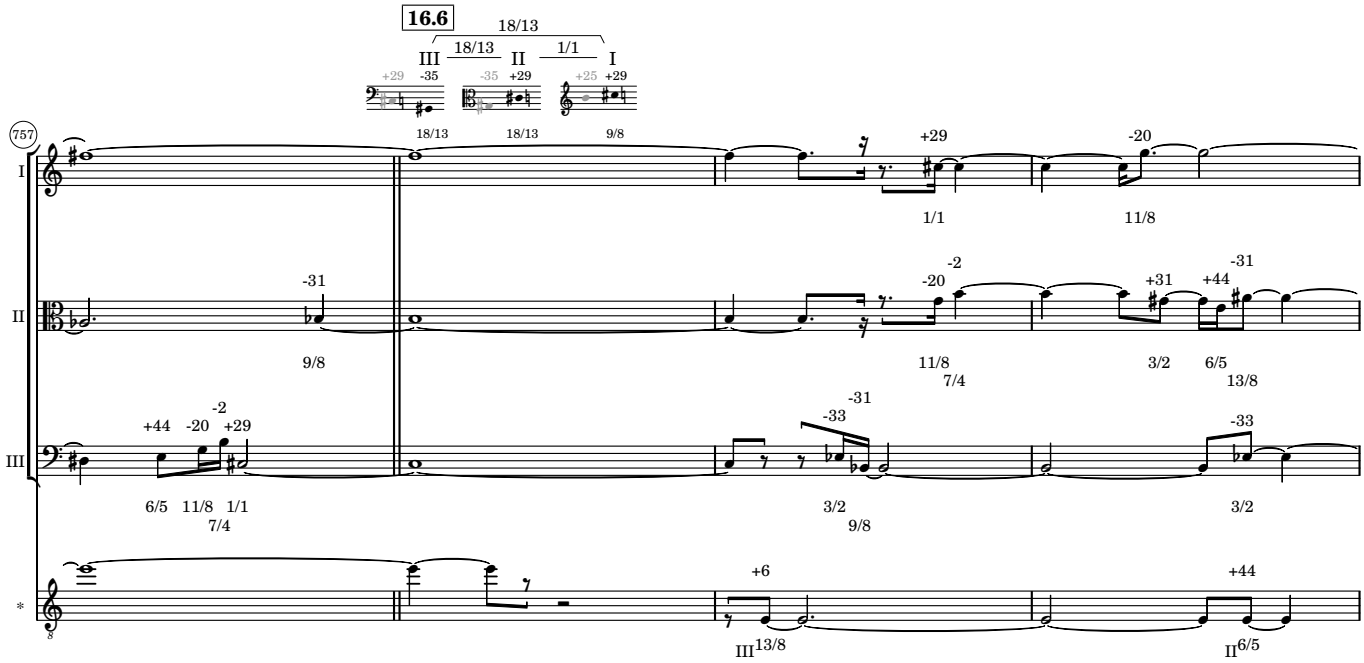

15.6
14/11
III 14/11 II 1/1 I
705
I
II
III
*
+29 +27 -35
9/8 13/8
3/2
-6 -35 -24 -35 +27
7/4 11/8 13/8 13/8 3/2
+9
3/2
+48
I13/8

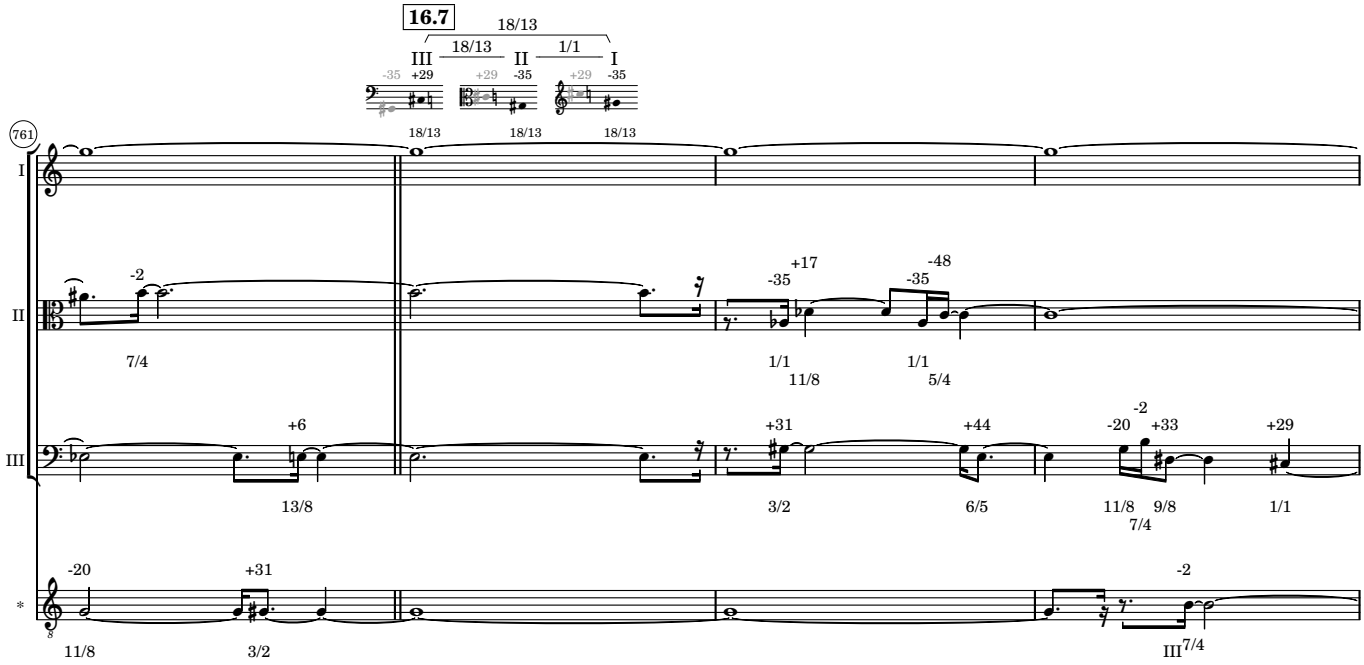

15.7
14/11
III 1/1 II 14/11 I
709
I
II
III
*
713
15.8
1/1
717

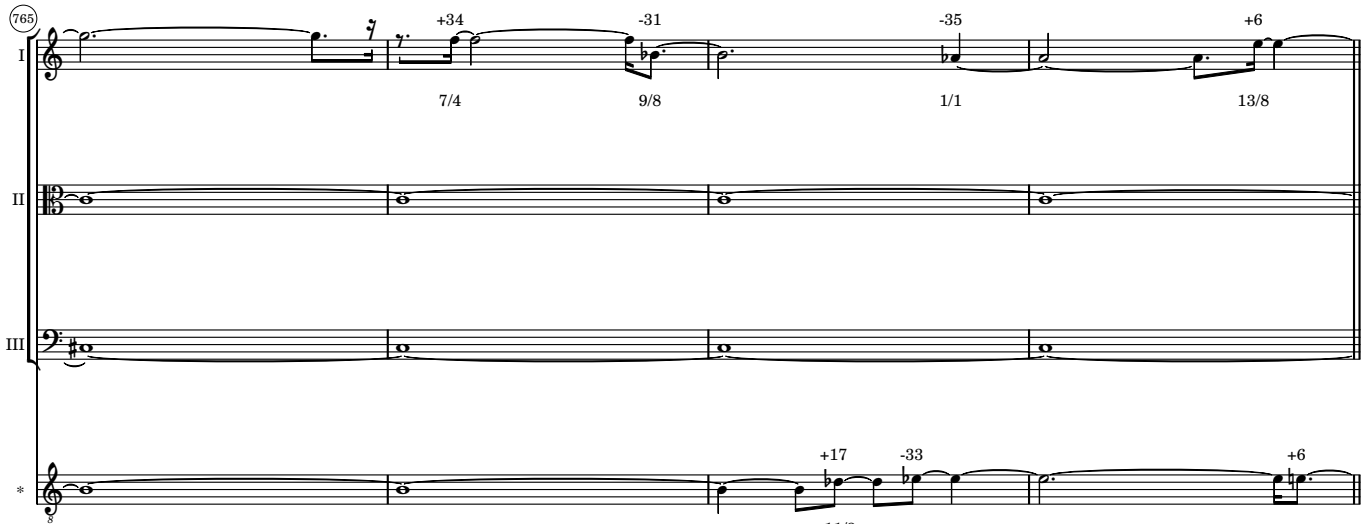

721
I
II
III
*
725
729
16.1
16.2

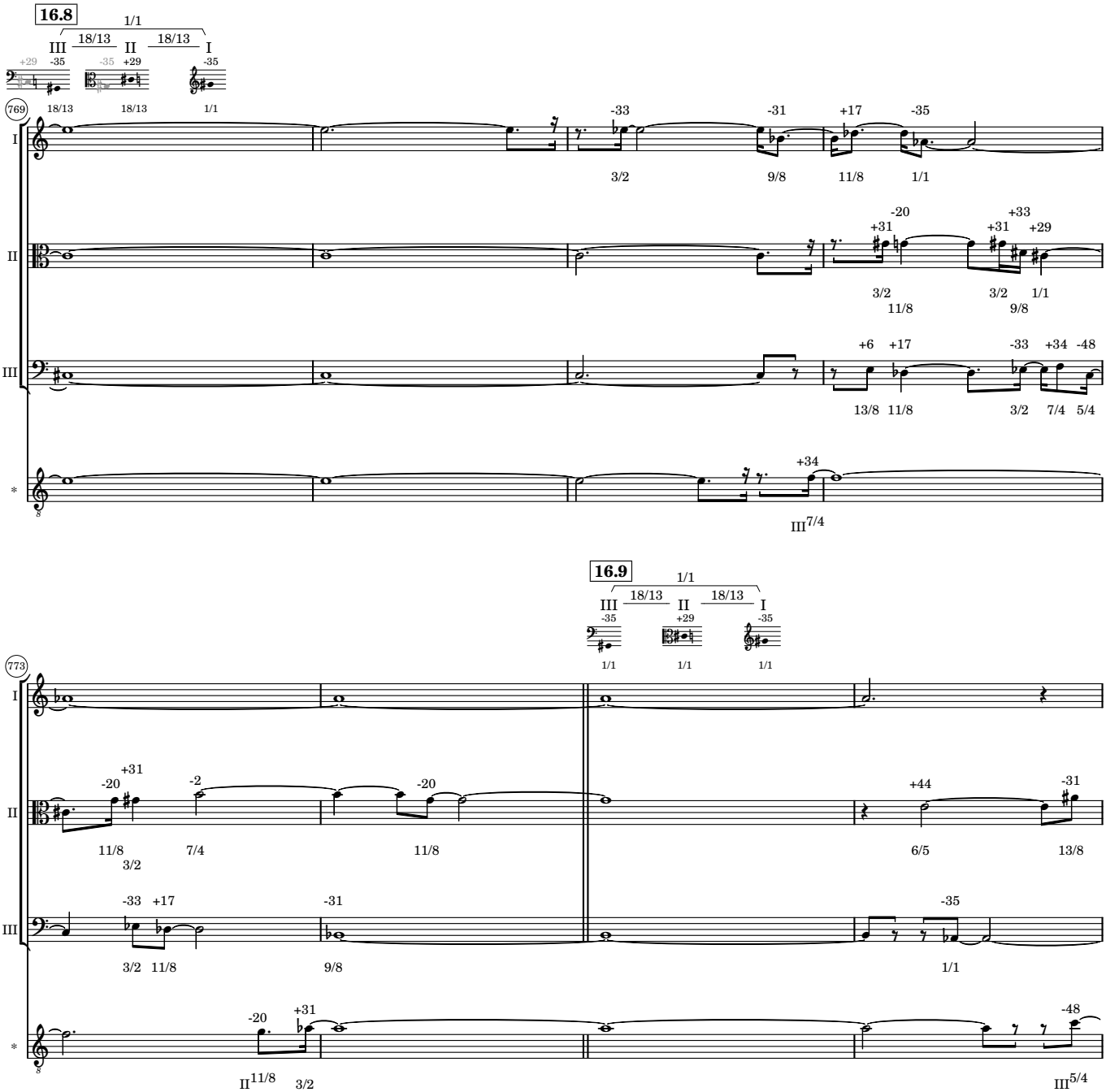

733
I
II
III
*
737
741
16.3
9/8
III 9/8 II 1/1 I

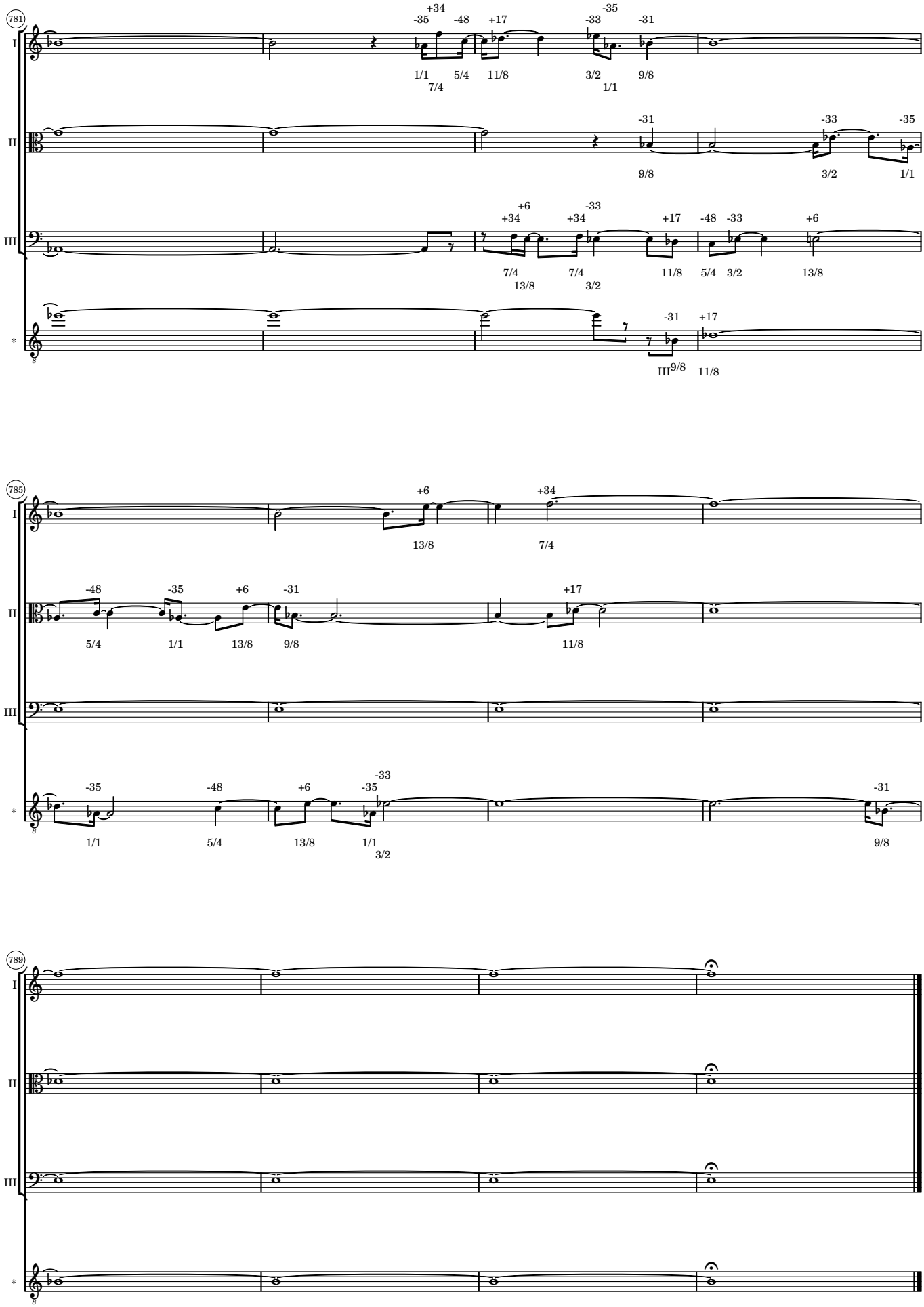

16.4
18/13
III 9/8 II 16/13 I
745
I
II
III
*
749
753
16.5
9/8
18/13
16/13


16.6
18/13
III 18/13 II 1/1 I
757
I
II
III
*


16.7
18/13
III 18/13 II 1/1 I
761
I
II
III
*


765
I
II
III
*


16.8
1/1
III 18/13 II 18/13 I
769
I
II
III
*
III7/4
16.9
1/1
III 18/13 II 18/13 I
773
II11/8
III5/4
16.10
1/1
III 1/1 II 1/1 I
777


781
I
II
III
*
785
789

appendix - SuperCollider code and Lilypond template

tkam_readme.scd

```
/*
----execute
Excecute tkam_main.scd to run.


----transport tab
The play button will always start from the beginning of the current section.

The transport buttons allow you to advance by subsection (<,>) and section (<<,>>).

Tempo change will only go into effect once the enter key or "set tempo" button is pressed.

The default seed given in the application will generate the first version of the music and score (as provided). Changing the seed will generate a new version with that seed once the "generate" button is pressed. After the new version is generated, new Lilypond files can be generated by pressing the "transcribe" button. This will create a tkam_score.ly file in a folder labeled "seed_[number]" which can be rendered by Lilypond. Note that the file must be rendered from that location as it dependes on files in that folder and the "includes" subfolder.


----mixer tab
This allow invidual control of each of the sonic elements. The three parts that can be played on acoustic instruments are automatically muted. The outputs will go out to whatever sound card is being used by the system.
*/
```

tkam_main.scd

```
(
// MAIN LAUNCH (loads necessary files and definitions)

var appEnvironment;

//push new environment
appEnvironment = Environment.make;
appEnvironment.push;

s.waitForBoot({
    var preampBusses, accompBbusses, postampBusses;

    ~hash = Date.getDate.hash.asString;
    ~cRes = 1;

    // load all files
    "tkam_musical_data_generator.scd".loadRelative;
    "tkam_sonifier.scd".loadRelative;
    "tkam_gui.scd".loadRelative;
    "tkam_transcriber.scd".loadRelative;

    # preampBusses, accompBusses, postampBusses = ~allocBusses.value(s);
    ~defineSynths.value(s, preampBusses, accompBusses, postampBusses);

    ~genAll = {arg seed;
        ~dUnit = 8.reciprocal;
        ~musicData = ~genMusicData.value(seed);
        ~scoreData = ~genScoreData.value(~musicData[0]);
        ~sectionData = ~musicData[4];
        ~patterns = ~genPatterns.value(~musicData[0], ~musicData[1], ~musicData[2], ~musicData[3], ~sectionData,
            preampBusses, accompBusses, postampBusses);
        ~sectionNavDict = ~musicData[5];
        ~isPlaying = false;
    };

    ~patternProxy = EventPatternProxy.new;

    ~tempoClock = TempoClock.new(1);
    ~dir = thisProcess.nowExecutingPath.dirname;
    "loading app".postln;
    ~genAll.value(19800725);
    ~play = Synth.new(\masterPlayerControl_ ++ ~hash);
    4.collect({arg p; Synth.new(\clip_ ++ ~hash, [\bin,  accompBusses[p].index, \bus, postampBusses[5].index])});
    ~generateGUI.value(preampBusses, accompBusses, postampBusses);
    "ready".postln;
});
appEnvironment.pop;
)
```

tkam_musical_data_generator.scd

```
(
// DATA GENEREATOR - this file IS the piece
var frAdd, frDiff, frToFloat, frNearestInList, frCollapse, harmonicDistance,
genMode, hdChoose, wchooseDict, collectRoots,
initModeState, advanceMode,
initTemporalState, genTemporalData,
initPartStates, distributeRoots,
genEnsemblePart, genAccompPart, genBassPart, genAmpCurve, genMusicData, genScoreData, genPatterns;


//~~~~~~~~~~~~FREQUENCY RATIO MATH FUNCTIONS~~~~~~~~~~~
//for frequency ratios in the form [numerator.factors, denominator.factors]
//we use arrays of factors in order to represent very complex ratios

//add
frAdd = {arg fr0, fr1;
    var num, den;
    num = fr0[0] ++ fr1[0];
    den = fr0[1] ++ fr1[1];
    [[1] ++ num.difference(den).sort, [1] ++ den.difference(num).sort]
};

//difference
frDiff = {arg fr0, fr1;
    var res;
    res = frAdd.value(fr0, fr1.reverse);
    if(frToFloat.value(res) < 1, {res = res.reverse});
    res
};

//convert to float
frToFloat = {arg fr; fr[0].asFloat.product / fr[1].asFloat.product};

//find nearest in list (not sharing the same root or itself)
frNearestInList = {arg frComp, frDict;
    var frNearest, diffNearest, sub;
    frNearest = nil;
    diffNearest = 1000;
    frDict.reject({arg item;
```

```
        (item[\root][0] == frComp[1]) ||
        (item[\fr] == frComp[1])
    }).keys.asList.sort({arg a, b; harmonicDistance.value(a) < harmonicDistance.value(b)}).do({arg fr;
        var diff = abs(frToFloat.value(fr) - frToFloat.value(frComp[0]));
        if(diff < diffNearest, {diffNearest = diff; frNearest = fr});
    });
    frNearest
};

//collapse into one octave
frCollapse = {arg fr;
    var res = fr;
    while({frToFloat.value(res) >= 2}, {res = frAdd.value(res, [1, 2])});
    while({frToFloat.value(res) < 1}, {res = frAdd.value(res, [2, 1])});
    res
};

//harmonic distance
harmonicDistance = {arg fr; log2(fr[0].asFloat.product * fr[1].asFloat.product)};


//------------GENERATE MODE------------
genMode = {arg forceHS = false;
    var mode, alternateProb;
    alternateProb = [1, 0].wchoose([if(forceHS, {0}, {1}), 4].normalizeSum);
    mode = [
        [1, 1],
        [9, 8],
        [[5, 4], [6, 5]].wchoose([3, if(forceHS, {0}, {1})].normalizeSum),
        [[4, 3], [11, 8]].wchoose([alternateProb, 1].normalizeSum),
        [3, 2],
        [[8, 5], [13, 8]].wchoose([alternateProb, 1].normalizeSum),
        [[15, 8], [7, 4]].wchoose([alternateProb, 1].normalizeSum)
    ];
    mode.collect({arg fr; [[1] ++ fr[0].factors, [1] ++ fr[1].factors]});
};


//------------CHOOSE AND COLLECT FUNCTIONS------------
hdChoose = {arg mode, exp = 1, weights = [1, 1, 1, 1, 1, 1, 1];
    var probs;
    probs = pow((1 / mode.collect({arg fr;
        harmonicDistance.value(if(fr == [[1], [1]], {[[2], [1]]}, {fr}))})), exp) * weights;
    mode.wchoose(probs.normalizeSum)
};

wchooseDict = {arg dict, exp = 1, limit = 0, isFR = true;
    var keyList, probs;
    keyList = if(isFR, {
        dict.keys.asList.sort({arg a, b; harmonicDistance.value(a) < harmonicDistance.value(b)});
    }, {
        dict.keys.asList.sort({arg a, b; a.convertDigits(2) < b.convertDigits(2)});
    });
    probs = keyList.collect({arg key;
        var count = dict[key][\count];
        if(count < limit, {0}, {count})
    });
    probs = pow(probs, exp);
    keyList.wchoose(probs.normalizeSum)
};

collectRoots = {arg dict; dict.keys.collect({arg fr;
    dict[fr][\root][0]}).asList.sort({arg a, b; harmonicDistance.value(a) < harmonicDistance.value(b)});
};


//------------GENERATE MODE SEQUENCE------------
initModeState = {
    var curModeState, frSet;
    curModeState = Dictionary.new;
    frSet = [[1, 1], [9, 8], [6, 5], [11, 8], [3, 2], [13, 8], [7, 4]].collect({arg fr;
        [[1] ++ fr[0].factors, [1] ++ fr[1].factors]});
    frSet.do({arg fr;
        var mode, count;
        mode = genMode.value;
        count = if(fr == [[1], [1]], {10}, {1});
        curModeState.add(fr->
            Dictionary.with(*[\count->count, \mode->frSet, \root->[[[1], [1]], frSet], \mult->fr, \fr->fr]))
    });
    curModeState;
};

advanceMode = {arg lastModeState, lastCadenceState, forceHS = false;
    var curModeState, curRoots, lastRoots, lastCadenceRoot, changeCount, modSpeed;

    curModeState = lastModeState.deepCopy;
    curRoots = collectRoots.value(curModeState);
    lastRoots = collectRoots.value(lastModeState);
    lastCadenceRoot = collectRoots.value(lastCadenceState).asList[0];
    changeCount = 0;
    modSpeed = if(forceHS, {1}, {[1, 2, 3].wchoose([2, 3, 1].normalizeSum)});

    while({
        ((curRoots == lastRoots) && (changeCount < modSpeed)) ||
        ((changeCount < modSpeed) && (curRoots.size > 1))
    }, {
        var roots, rootSel, mults, multProbs, multSel, new;

        //bump for length of time its been around
        curModeState.keysValuesDo({arg key, val;
            val[\count] = val[\count] + 1;
            if(val[\count] > 100, {val[\count] = 1});
        });

        // max 3 roots that were not the last cadencial root and only in the mode of the last cadence
        roots = curModeState.reject({arg val;
            (val[\fr] == lastCadenceRoot) ||
            lastCadenceState.includesKey(val[\fr]).not || //consider two steps out?
            ((curRoots.size >= 3) && curRoots.includes(val[\fr]).not)
        });

        rootSel = wchooseDict.value(roots, 1, 2);
        mults = curModeState[rootSel][\mode];
        multProbs = mults.collect({arg fr;
            if(curModeState.keys.includes(frCollapse.value(frAdd.value(rootSel, fr))), {1}, {2})});
        multSel = hdChoose.value(mults, 0.5, multProbs);
        new = frCollapse.value(frAdd.value(rootSel, multSel));

        curModeState[rootSel][\count] = curModeState[rootSel][\count] + 2; //bump if gets chosen as a root

        if(curModeState.includesKey(new), {
            //bump if it gets chosen again
            curModeState[new][\count] = curModeState[new][\count] + 1;
            if((curModeState[new][\count] >= 20), {
                curModeState[new][\root] = [rootSel, curModeState[rootSel][\mode]]
```

```
            });
        }, {
            var old;
            //calculate nearest in list that does not share the same root
            old = frNearestInList.value([new, rootSel], curModeState);
            if(curModeState[old][\count] >= 20, {
                var mode, root;
                root = [rootSel, curModeState[rootSel][\mode]];
                curModeState.add(new ->
                    Dictionary.with(*[\count->1,\mode->genMode.value(forceHS), \root->root, \mult->multSel, \fr->new]));
                curModeState.removeAt(old);
                changeCount = changeCount + 1;
            })
        });
        curRoots = collectRoots.value(curModeState);
    });
    curModeState
};


//------------GENERATE TEMPORAL FRAMEWORK------------
initTemporalState = {
    Dictionary.with(*[[0, 1], [0, 1], [0, 1]].allTuples.collect({arg tuple;
        tuple->Dictionary.with(*[\count->1])}))
};

genTemporalData = {arg lastTupleState, modeState, cadenceOverride, noParts = 4;
    var cadence, curTupleState, timeToNextEvent, tuple, temporalData;

    cadence = if(collectRoots.value(modeState).size == 1, {cadenceOverride.not}, {false});
    curTupleState = lastTupleState.deepCopy;
    timeToNextEvent = (64 + 50.rand + if(cadence, {50}, {0})).round(16);
    tuple = wchooseDict.value(curTupleState, isFR: false);
    if(cadence, {tuple = [1, 1, 1]});
    curTupleState = curTupleState.keysValuesDo({arg key, val;
        curTupleState[key][\count] = val[\count] + 1});
    curTupleState[tuple][\count] = 0;
    tuple = if(cadence, {[1, 1, 1, 1]}, {[0] ++ tuple});

    temporalData = noParts.collect({arg p;
        var flourishDensity, genDensity, flourish, beforeLen, before, after, buffer;
        flourishDensity = if(tuple[p] == 1, {0.125 + 0.5.rand}, {3});
        if((p == 0) && cadence.not, {flourishDensity = 3});
        genDensity = if(p == 0, {5}, {20});

        flourish = (if(cadence, {16}, {8}) + 32.rand).collect({[0, 1].wchoose([flourishDensity, 1].normalizeSum)});
        buffer = 16.collect({0});
        beforeLen =((timeToNextEvent - flourish.size - buffer.size) /  if(cadence, {1.25}, {1})).asInteger.rand;
        before = beforeLen.collect({arg i; [0, 1].wchoose([genDensity, 0.25].normalizeSum)});
        after = (timeToNextEvent - before.size - flourish.size - buffer.size).collect({[0, 1].wchoose([genDensity, 1].normalizeSum)});
        flourish = before ++ flourish ++ after;
        if(flourish.sum == 0, {flourish[flourish.size.rand] = 1});
        flourish = buffer ++ flourish;
    });

    [temporalData, curTupleState]
};


//------------GENERATE ENSEMBLE PARTS------------
initPartStates = {
    var allRatios;
    allRatios = [[1, 1], [9, 8], [5, 4], [6, 5], [4, 3], [11, 8], [3, 2], [8, 5], [13, 8], [15, 8], [7, 4]];
    Dictionary.with(*
        4.collect({arg part;
            part->Dictionary.with(*[\multCounts->
                Dictionary.with(*allRatios.collect({arg fr;
                    [[1] ++ fr[0].factors, [1] ++ fr[1].factors]->1})),
                \noteCount->0, \index->part, \lastFreq->0, \lastFreqRatio->[[1], [1]], \lastDur->0
            ])
        })
    )
};

//this is how roots are distributed to the parts
distributeRoots = {arg modeState, lastRoots;
    var roots;
    roots = modeState.keys.asList.collect({arg fr; modeState[fr][\root]});
    roots = roots.asBag.contents.asPairs.reverse.clump(2);
    roots = roots.sort({arg a, b;
        if(a[0] != b[0], {a[0] > b[0]}, {frToFloat.value(a[1][0]) > frToFloat.value(b[1][0])})}).collect({arg item; item[1]}).wrapExtend(4);

    roots = [roots[0]] ++ roots[..2].scramble;
    roots = 4.collect({arg part;
        var root, rootMod, rootFreq, mode;
        root = roots[part];
        rootMod = frDiff.value(root[0], lastRoots[part]);
        rootFreq = 40.midicps * pow(2, [1, 0, 1, 2][part]) * frToFloat.value(root[0]);
        [root[0], root[1], rootMod, rootFreq]
    });
    roots = roots.collect({arg root, r;
        var rootRels;
        rootRels = 4.collect({arg p; frDiff.value(root[0], roots[p][0])});
        rootRels.removeAt(r);
        root.add(rootRels)
    });
    roots
};

genEnsemblePart = {arg partState, modeState, temporalData, roots, part, offset;
    var trans, root, mults, rootMod, amp, firstChange, cadence, lastInsRef, ensData;

    trans = pow(2, [1, 0, 1, 2][partState[\index]]);
    # root, mults, rootMod = roots[part];
    amp =  [0, 1, 2, 3].wchoose([0, 2, 2, 2].normalizeSum);
    firstChange = false;
    cadence = if(collectRoots.value(modeState).size == 1, {true}, {false});
    lastInsRef = nil;

    ensData = [];
    temporalData.do({arg val, ts;
        var timeStamp, comp, change;

        partState[\lastDur] = partState[\lastDur] + 1;
        timeStamp = ts + offset;
        change = [val == 1, (val == 1) && firstChange.not].wchoose([1, 2].normalizeSum);
        if(
            (partState[\index] == 0) &&
            (frToFloat.value(partState[\lastFreqRatio]) >= 4.0) &&
            (partState[\lastDur] < 16) && cadence.not,
            {change = false}
        );

        if(change, {
            var mult, multWeights, freq, rootFreq, insRef;
```

```
290
291             //this weights notes that are richer and mixes with the DCA algorithm
292             multWeights = mults.collect({arg fr;
293                 var comp = frCollapse.value(frAdd.value(root, fr));
294                 if(modeState.keys.includes(comp), {3}, {1}) *  pow(partState[\multCounts][fr], 1);
295             });
296
297             mult = hdChoose.value(mults, 0.5, multWeights);
298             mults.do({arg fr; partState[\multCounts][fr] = partState[\multCounts][fr] + 1});
299             partState[\multCounts][mult] = 0;
300
301             freq = 40.midicps * trans * frToFloat.value(frAdd.value(root, mult));
302
303             //flute special case
304             if((partState[\index] == 0) && cadence.not, {
305                 var mode, continue, freqRatio;
306                 mode = modeState.keys.asList.collect({arg fr;
307                     [
308                         frCollapse.value(frAdd.value(modeState[fr][\root][0], modeState[fr][\mult])),
309                         modeState[fr][\root][0], modeState[fr][\mult]
310                     ]
311                 });
312                 mode = mode.sort({arg a, b;
313                     case
314                     {frToFloat.value(a[0]) != frToFloat.value(b[0])} {frToFloat.value(a[0]) < frToFloat.value(b[0])}
315                     {frToFloat.value(a[1]) != frToFloat.value(b[1])} {frToFloat.value(a[1]) < frToFloat.value(b[1])}
316                     {true} {frToFloat.value(a[2]) < frToFloat.value(b[2])};
317                 });
318                 mode = mode ++ mode.collect({arg fr; [frAdd.value(fr[0], [[2], [1]]), fr[1], fr[2]]});
319                 mode = mode ++ mode.collect({arg fr; [frAdd.value(fr[0], [[4], [1]]), fr[1], fr[2]]});
320                 continue = true;
321                 while({continue}, {
322                     # freqRatio, root, mult = mode[partState[\noteCount] % 15];
323                     freq = 40.midicps * trans * frToFloat.value(frAdd.value([[1], [1]], freqRatio));
324                     continue = (freq <= partState[\lastFreq]) && ((partState[\noteCount] % 15) != 0);
325                     partState[\noteCount] = partState[\noteCount] + 1;
326                 });
327                 partState[\lastFreq] = freq;
328                 partState[\lastFreqRatio] = freqRatio;
329                 insRef = roots.slice(nil, 0).deepCopy.drop(1).indexOfEqual(root) + 1;
330                 insRef = if(lastInsRef != insRef, {lastInsRef = insRef; insRef}, {lastInsRef = insRef; nil});
331             });
332             if((partState[\index] == 0) && cadence, {
333                 insRef = if(firstChange.not, {1}, {nil});
334             });
335
336             rootFreq = 40.midicps * trans * frToFloat.value(root);
337
338             if((partState[\index] == 0) && ((partState[\noteCount] % 15) == 1) && cadence.not, {ensData = ensData.add([0, timeStamp - 8, 0, 0, 0])});
339             ensData = ensData.add([freq, timeStamp, amp, mult, insRef]);
340             firstChange = true;
341             partState[\lastDur] = 0;
342             if((partState[\index] == 0) && cadence, {partState[\lastDur] = 32});
343         });
344     });
345     ensData = [[0, ensData[0][1] - 4, 0, 0, 0]] ++ ensData;
346     [ensData, partState]
347 };
348
349
350 //~~~~~~~~~~~GENERATE ELECTRONIC ACCOMPANIMENT~~~~~~~~~~~
351 genAccompPart = {arg modeState, temporalData, offset, trans, part, register;
352     var firstChange, accompData;
353     firstChange = false;
354     accompData = [];
355     temporalData.do({arg val, tS;
356         var change;
357         change = [val == 1, (val == 1) && firstChange.not].wchoose([1, if(part == 0, {5}, {3})].normalizeSum);
358         if(change, {
359             var sel, freq, amp;
360             sel = wchooseDict.value(modeState, 0.1);
361             freq = 48.midicps * trans * frToFloat.value(sel);
362             amp = [0, 1, 2, 3].wchoose([2, 2, 1, 1].normalizeSum);
363
364             accompData = accompData.add([freq, tS + offset, amp, part]);
365             firstChange = true;
366         });
367     });
368     accompData
369 };
370
371
372 //~~~~~~~~~~~GENERATE ELECTRONIC BASS~~~~~~~~~~~
373 genBassPart = {arg root, ampCurve, hi;
374     var freq;
375     freq = if(hi,
376         {40.midicps * frToFloat.value(frCollapse.value(frAdd.value(root, [[3], [2]])))},
377         {40.midicps * frToFloat.value(root)});
378     ampCurve.collect({arg sec, iter; [freq, sec[1]]})
379 };
380
381
382 //~~~~~~~~~~~GENERATE AMP CURVES~~~~~~~~~~~
383 genAmpCurve = {arg temporalData1, temporalData2, offset1, offset2, type;
384     var firsts1, firsts2, delay, attack, decay, release, min, max, env;
385     firsts1 = temporalData1.collect({arg ptd; ptd.indexOf(1)});
386     firsts2 = temporalData2.collect({arg ptd; ptd.indexOf(1)});
387     delay = switch(type)
388     {0} {0}
389     {1} {0}
390     {2} {firsts1.minItem};
391     attack = switch(type)
392     {0} {offset2 - offset1}
393     {1} {offset2 - temporalData2[0].size + firsts2.minItem - offset1}
394     {2} {firsts1.maxItem - firsts1.minItem};
395     decay = switch(type)
396     {0} {0}
397     {1} {firsts2.maxItem - firsts2.minItem}
398     {2} {temporalData1[0].size - firsts1.maxItem};
399     release = switch(type)
400     {0} {0}
401     {1} {temporalData2[0].size - firsts2.maxItem}
402     {2} {(offset2 - temporalData2[0].size) - offset1} ;
403     min = switch(type)
404     {0} {0.15}
405     {1} {0}
406     {2} {0};
407     max = switch(type)
408     {0} {0.5}
409     {1} {0.65}
410     {2} {1};
411
412     env = Env.dadsr(delay, attack, decay, 0.25, release, curve: \cub).range(min, max);
413     ((delay + attack + decay + release) / 1).asInteger.collect({arg iter; [env.at(iter * 1), offset1 + (iter * 1)]})
414 };
```

```
415
416
417 //------------GENERATE ALL MUSIC DATA------------
418 ~genMusicData = {arg seed;
419     var minTotalDur, minSection1Dur, dUnit,  curLen, cadence,
420     ultimateSubsection, ultimateSection, ultimateCadenceCount,
421     minTotalLen, minSection1Len,
422     modeState, temporalState, partStates,
423     lastCadenceTemporalData, lastCadenceState, lastSectionPoint,
424     ensData, accompData, bassData, ampData,
425     sectionData, sectionNavDict,
426     sectionCount, subsectionCount,
427     lastRoots, roots, ampDataTmp;
428
429     thisThread.randSeed = seed;
430
431     # minTotalDur, minSection1Dur, dUnit, curLen, cadence = [23 * 60, 8 * 60, 8.reciprocal, 0, false];
432     # ultimateSubsection, ultimateSection, ultimateCadenceCount = [false, false, 0];
433     # minTotalLen, minSection1Len = [(minTotalDur / dUnit).round(16), (minSection1Dur / dUnit).round(16)];
434     # modeState, temporalState, partStates = [initModeState.value, initTemporalState.value, initPartStates.value];
435     # lastCadenceTemporalData, lastCadenceState, lastSectionPoint = [nil, modeState.deepCopy, 0];
436     # ensData, accompData, bassData, ampData = [4.collect({[]}), 4.collect({6.collect({[]})}), 2.collect({[]}), 3.collect({[]})];
437     # sectionData, sectionNavDict = [Dictionary.new, Dictionary.new];
438     # sectionCount, subsectionCount = [1, 1];
439
440     while({(curLen < minTotalLen) || ((curLen >= minTotalLen) && cadence.not) || ultimateSection.not}, {
441         var temporalData;
442         # temporalData, temporalState = genTemporalData.value(temporalState, modeState, curLen <= minSection1Len);
443
444         collectRoots.value(modeState).collect({arg fr; [fr[0].asFloat.product, fr[1].asFloat.product]}).postln;
445         //modeState.keys.postln;
446         ("-- " ++ sectionCount ++ "." ++ subsectionCount ++ " -----").postln;
447
448         lastRoots = if(curLen == 0, {4.collect({[[1], [1]]})}, {roots.slice(nil, 0)});
449         roots = distributeRoots.value(modeState, lastRoots);
450
451         sectionData.add((curLen / 4).asInteger->[roots, lastRoots.collect({arg fr, part;
452             [fr, 40.midicps * pow(2, [1, 0, 1, 2][part]) * frToFloat.value(fr)]}), sectionCount, subsectionCount, cadence, ultimateSubsection]);
453         sectionNavDict.add([sectionCount, subsectionCount]->[(curLen / 16 + 1).asInteger]);
454
455         4.do({arg part;
456             var musicData, partState, accompRoutine;
457             # musicData, partState = genEnsemblePart.value(partStates[part], modeState, temporalData[part], roots, part, curLen);
458             ensData[part] = ensData[part] ++ musicData;
459             partStates[part] = partState;
460
461             //use an independent random number generator for the accompaniment
462             accompRoutine = Routine({
463                 thisThread.randSeed = Date.seed;
464                 6.do({arg register;
465                     musicData = genAccompPart.value(modeState, temporalData[part], curLen, pow(2, part + register), part, register);
466                     accompData[part][register] = accompData[part][register] ++ musicData;
467                 });
468             });
469             accompRoutine.value;
470
471         });
472
473         subsectionCount = subsectionCount + 1;
474
475         if(curLen == 0, {
476             lastCadenceTemporalData = temporalData;
477             lastSectionPoint = curLen;
478         });
479
480         curLen = curLen + temporalData[0].size;
481
482         if(curLen > minSection1Len, {
483             if(collectRoots.value(modeState).size == 1, {
484
485                 ampData[0] = ampData[0] ++ genAmpCurve.value(lastCadenceTemporalData, temporalData, lastSectionPoint, curLen, 0);
486                 ampDataTmp = genAmpCurve.value(lastCadenceTemporalData, temporalData, lastSectionPoint, curLen, 1);
487                 ampData[1] = ampData[1] ++ ampDataTmp;
488                 bassData[0] = bassData[0] ++ genBassPart.value(collectRoots.value(modeState).asList[0], ampDataTmp, true);
489
490                 if(sectionCount == 1, {
491                     ampData[2] = ampData[2] ++ ((curLen - temporalData[0].size) / 1).asInteger.collect({arg iter; [0, iter * 1]});
492                     bassData[1] = bassData[1] ++ ((curLen - temporalData[0].size) / 1).asInteger.collect({arg iter; [0, iter * 1]})
493                 }, {
494                     ampDataTmp = genAmpCurve.value(lastCadenceTemporalData, temporalData, lastSectionPoint, curLen, 2);
495                     ampData[2] = ampData[2] ++ ampDataTmp;
496                     bassData[1] = bassData[1] ++ genBassPart.value(collectRoots.value(lastCadenceState).asList[0], ampDataTmp, false);
497                 });
498
499                 sectionData.add(((curLen - temporalData[0].size) / 4).asInteger->
500                     sectionData[((curLen - temporalData[0].size) / 4).asInteger].put(5, true));
501                 (subsectionCount - 1).do({arg subsectionIndex;
502                     sectionNavDict.add([sectionCount, subsectionIndex + 1]->[sectionNavDict[[sectionCount, subsectionIndex + 1]][0], subsectionCount - 1]);
503                 });
504                 # sectionCount, subsectionCount = [sectionCount + 1, 1];
505                 # lastCadenceTemporalData, lastCadenceState, lastSectionPoint = [temporalData, modeState, curLen];
506                 cadence = true;
507
508                 //this should ensure that the final cadence is a HS
509                 if(curLen >= minTotalLen, {ultimateCadenceCount = ultimateCadenceCount + 1});
510                 ultimateSection = ultimateCadenceCount > 1;
511             }, {
512                 cadence = false
513             });
514             modeState = advanceMode.value(modeState, lastCadenceState, curLen >= minTotalLen);
515         });
516     });
517
518     ampDataTmp = genAmpCurve.value(lastCadenceTemporalData, [[1], [1], [1], [1]], lastSectionPoint, lastSectionPoint + 360, 2);
519     ampData[2] = ampData[2] ++ ampDataTmp;
520     bassData[1] = bassData[1] ++ genBassPart.value(collectRoots.value(lastCadenceState).asList[0], ampDataTmp, false);
521
522     [ensData, accompData, bassData, ampData, sectionData, sectionNavDict]
523 };
524 )
```

tkam_sonifier.scd

```
1  (
2  var formatPatternData;
3
4  //------------ALLOCATE BUSSES------------
5  ~allocBusses = {arg server;
6      var preampBusses, accompBusses, postampBusses;
7      preampBusses = 3.collect({Bus.audio(server, 1)});
8      accompBusses = 4.collect({Bus.audio(server, 1)});
9      postampBusses = 7.collect({Bus.audio(server, 1)});
10     [preampBusses, accompBusses, postampBusses];
```

```
};

//------------DEFINE SYNTHS------------
~defineSynths = {arg server, preampBusses, accompBusses, postampBusses;
    var sdPlayer, sdTransport, sdClick, sdAmpCurve, sdEns, sdAccomp, sdClip, sdBass, sdDiskOut, allSds;

    sdPlayer = SynthDef(\masterPlayerControl_ ++ ~hash, {
        var router, sigs, sigsPanned, masterSig, imp;

        sigs = postampBusses.collect({arg bus, i; In.ar(bus) * NamedControl.kr(\vol_ ++ i, 1, 0.1) * NamedControl.kr(\mute_ ++ i, 1, 0.1)});
        router = sigs.collect({arg sig, i; NamedControl.kr(\out_ ++ i, 0, 0)});
        sigs.collect({arg sig, i; Out.ar(router[i] - 1, sig * router[i].sign)});

        sigsPanned = sigs.collect({arg sig, i; Pan2.ar(sig, NamedControl.kr(\pan_ ++ i, 0, 0.1))});
        masterSig = Mix.ar(sigsPanned.collect({arg sig, i; sig * abs(router[i].sign - 1)}));
        masterSig = masterSig * NamedControl.kr(\masterVol, 1, 0.1) * NamedControl.kr(\masterMute, 1, 0.1);

        Out.ar(NamedControl.kr(\masterOut, 0, 0), masterSig);

        imp = Impulse.kr(10);
        SendReply.kr(imp, '/masterLevels_' ++ ~hash, values: [Amplitude.kr(masterSig)]);
        sigs.collect({arg sig, i; SendReply.kr(imp, '/trackLevel_' ++ i ++ "-" ++ ~hash, values: [Amplitude.kr(sig)])});
    });

    sdTransport = SynthDef(\transport_ ++ ~hash, {arg measure = 0, beat = 0, section = 0, subsection = 0, gate = 1, dur = 1;
        SendReply.kr(Impulse.kr(0) * (measure > 0) * (beat > 0),'/measureClock_' ++ ~hash, values: [measure, beat, section, subsection]);
        EnvGen.kr(Env.sine(dur), gate, doneAction: 2);
    });

    sdClick = SynthDef(\click_ ++ ~hash, {arg beat = 0, gate = 1, dur = 1;
        Out.ar(postampBusses[6], 10 * BPF.ar(WhiteNoise.ar * EnvGen.kr(Env.perc(0.01, 0.1), gate), 440 * ((beat <= 1) + 1), 0.02));
        EnvGen.kr(Env.sine(dur), gate, doneAction: 2);
    });

    sdAmpCurve = SynthDef(\amp_curve_ ++ ~hash, {arg amp = 1, dur = 0.1, bus = 0;
        Out.kr(bus, amp.lag)
    });

    sdEns = SynthDef(\ens_ ++ ~hash, {arg freq = 440, amp = 1, dur = 1, gate = 1, bus = 0, ampBus = 0, rel = 0.1;
        Out.ar(bus, SinOsc.ar(freq, 2pi.rand, 0.1) * amp * Latch.kr(In.kr(ampBus), Impulse.kr(0)) * EnvGen.kr(Env.asr(0.1, 1, rel), gate, doneAction: 2))
    });

    sdAccomp = SynthDef(\accomp_ ++ ~hash, {arg freq = 440, amp = 1, sustain = 1, dur = 1, gate = 1, bout = 0, ampBus = 0, rel = 0.01;
        Out.ar(bout, SinOsc.ar(freq, 2pi.rand, 1) * 0.01 * amp * Latch.kr(In.kr(ampBus), Impulse.kr(0)) * EnvGen.kr(Env.asr(sustain, 1, rel), gate, doneAction: 2))
    });

    sdClip = SynthDef(\clip_ ++ ~hash, {arg dur = 1, gate = 1, bin = 0, bus = 0;
        Out.ar(bus, (In.ar(bin)).clip(0, 1) * 50)
    });

    sdBass = SynthDef(\bass_mono_ ++ ~hash, {arg freq = 440, ampBus = 0, bus = 0;
        Out.ar(bus, (SinOsc.ar(freq) * 0.5 * In.kr(ampBus)))
    });

    sdDiskOut = SynthDef(\disk_out_ ++ ~hash, {arg bufnum, inbus;
        DiskOut.ar(bufnum, In.ar(inbus));
    });

    allSds = [sdPlayer, sdTransport, sdClick, sdAmpCurve, sdEns, sdAccomp, sdClip, sdBass, sdDiskOut];
    allSds.do({arg sd; sd.add});
    allSds
};


// group data by measures for navigation
formatPatternData = {arg musData, measureLen, rel, print = false;
    var dataLen;
    dataLen = musData[0][0].size + 1;
    musData.collect({arg partData;
        var res;
        res = partData;
        res = res.collect({arg mData, index; mData.add(if(index != (res.size - 1), {rel}, {5.rand + 5}))});
        res = res.flop;
        res = res.add(res[1]);
        res[1] = (res[1].differentiate.drop(1) ++ [10]);
        res = res.flop ++ measureLen.collect({arg measure; dataLen.collect({0}) ++ [measure * 16]});
        res = res.sort({arg a, b; a.last < b.last}).flop;
        res = res.insert(1, (res.last.differentiate.drop(1) ++ [10])).flop;
        res = res.separate({arg a, b; (a.last / 16).trunc != (b.last / 16).trunc});
        res.collect({arg measureData; measureData.flop})
    }).flop
};


//------------GENERATE PATTERNS------------
//this generates patterns grouped by measures except for bass data and amp curve data which are much higher resolution
//these are used to make playable patterns
~genPatterns = {arg ensData, accompData, bassData, ampData, sectionData, preampBusses, accompBusses, postampBusses;
    var measureLen, ensDataFormatted, accompDataFormatted, bassDataFormatted, ampDataFormatted,
    dUnit, section, subsection, patterns;

    measureLen = ((
        ensData.collect({arg partData; partData.last[1]}) ++
        accompData.flatten.collect({arg partData; partData.last[1]})
    ).maxItem.ceil(16) / 16).asInteger + 1;

    ensDataFormatted = formatPatternData.value(ensData, measureLen, 0.1, true);
    accompDataFormatted = formatPatternData.value(accompData.flatten, measureLen, 0.01);
    dUnit = 8.reciprocal;

    patterns = measureLen.collect({arg measure;
        if(sectionData[measure * 4] != nil, {
            section = sectionData[measure * 4][2];
            subsection = sectionData[measure * 4][3];
        });
        Ppar(
            //check how amplitude is being handled
            ensDataFormatted[measure].collect({arg musData, p;
                Pbind(
                    \instrument, \ens_ ++ ~hash,
                    \freq, Pseq(musData[0].replace(0, Rest(0))),
                    \dur, Pseq(musData[1] * dUnit),
                    \sustain, Pseq(musData[2] * dUnit),
                    \amp, [1, 0.7, 0.5, 0.3][p],
                    \ampBus, preampBusses[0].index,
                    \bus, postampBusses[p].index,
                    \rel, Pseq(musData[6])
                )
            }) ++
            //check how amplitude and attack are being handled
            accompDataFormatted[measure].collect({arg musData;
                Pbind(
                    \instrument, \accomp_ ++ ~hash,
                    \freq, Pseq(musData[0].replace(0, Rest(0))),
                    \dur, Pseq(musData[1] * dUnit),
```

```
                    \sustain, Pseq(musData[2] * dUnit),
                    \amp, Pseq(musData[3].collect({arg item; [0, 2, 4, 8][item]}) * 0.0125 * 1),
                    \ampBus, preampBusses[0].index,
                    \bout, Pseq(musData[4].collect({arg index; accompBusses[index].index})),
                    \rel, Pseq(musData[5])
                )
            }) ++
            [
                Pbind(
                    \instrument, \transport_ ++ ~hash,
                    \measure, measure + 1,
                    \beat, Pseq([1, 2]),
                    \section, section,
                    \subsection, subsection,
                    \dur, 8 * dUnit
                ),
                Pbind(
                    \instrument, \click_ ++ ~hash,
                    \beat, Pseq([1, 2]),
                    \dur, 8 * dUnit
                )
            ]
        )
    });
    [patterns, bassData, ampData]
};


//this is a playable pattern based on start measure
~genPlayablePatterns = {arg startMeasure, patterns, preampBusses, accompBusses, postampBusses;
    Ppar(
        [Pseq(patterns[0][startMeasure..], 1)] ++

        patterns[2].collect({arg pattern, p;
            Pmono(\amp_curve_ ++ ~hash,
                \amp, Pseg(Pseq(pattern.slice(nil, 0)[(startMeasure * 16)..], 1), 1 * ~dUnit, \cub),
                \dur, 1 * ~dUnit,
                \bus, preampBusses[p].index
            )
        }) ++
        patterns[1].collect({arg pattern, p;
            Pmono(\bass_mono_ ++ ~hash,
                \freq, Pseq(pattern.slice(nil, 0)[(startMeasure * 16)..], 1),
                \dur, 1 * ~dUnit,
                \ampBus, preampBusses[p + 1].index,
                \bus, postampBusses[4].index
            )
        })
    );
};


//~~~~~~~~~~~~BOUNCE AUDIO~~~~~~~~~~~~
//this bounces the audio for use in another DAW or for practice
~bounceAudio = {arg seed;
    var trackNames, basePath, server, buffers, recDur,
    preampBusses, nextNode, accompBusses, postampBusses,
    synths, prePatterns, playablePatterns, score;

    trackNames = ["part_star", "part_III", "part_II", "part_I", "accomp_II", "accomp_I", "click"];

    basePath = ~dir +/+ ".." +/+ "audio" +/+ "seed_" ++ seed;
    basePath.mkdir;

    server = Server(\nrt_ ++ ~hash,
        options: ServerOptions.new
        .numOutputBusChannels_(7)
        .numInputBusChannels_(0)
    );

    # preampBusses, accompBusses, postampBusses = ~allocBusses.value(s);
    postampBusses = 7.collect({arg index; Bus.new(rate: 'audio', index: index, numChannels: 1, server: server)});
    synths = ~defineSynths.value(s, preampBusses, accompBusses, postampBusses);

    prePatterns = ~genPatterns.value(~musicData[0], ~musicData[1], ~musicData[2], ~musicData[3], ~sectionData,
            preampBusses, accompBusses, postampBusses);

    playablePatterns = ~genPlayablePatterns.value(0, prePatterns, preampBusses, accompBusses, postampBusses);

    recDur = (prePatterns[2][0].size / 8) + 45;
    score = playablePatterns.asScore(duration: recDur, timeOffset: 0.001);
    nextNode = score.score.slice(nil, 1).select({arg msg; msg[0] == 9}).slice(nil, 2).maxItem + 1;

    synths.do({arg synth; score.add([0.0, [\d_recv, synth.asBytes]])});

    4.collect({arg p;
        score.add([0.0, [\s_new, \clip_ ++ ~hash, nextNode, 1, 1, \bin,  accompBusses[p].index, \bus, postampBusses[5].index]]);
        nextNode = nextNode + 1;
    });

    buffers = 7.do({arg track;
        score.add([0.0, [\b_alloc, track, 65536, 1]]);
        score.add([0.0, [\b_write, track, basePath +/+ "tkam_" ++ trackNames[track] ++ ".wav".standardizePath, "WAV", "int16", 0, 0, 1]]);
        score.add([0.0, [\s_new, \disk_out_ ++ ~hash, nextNode, 1, 1, \bufnum, track, \inbus, track]]);
        score.add([recDur, [\n_free, nextNode]]);
        score.add([recDur, [\b_close, track]]);
        score.add([recDur, [\b_free, track]]);
        nextNode = nextNode + 1;
    });

    score.sort;

    score.recordNRT(
        outputFilePath: basePath +/+ "tkam_all" ++ ".wav".standardizePath,
        sampleRate: 44100,
        headerFormat: "WAV",
        sampleFormat: "int16",
        options: server.options,
        duration: recDur
    );

    server.remove;
}
)
```

tkam_transcriber.scd

```
(
var formatMusicData, spellingDict, lyNoteNameStr, lyOctStr, lyFinalizeMusic, lyMeasureDef,
lyRelMark, lyRelMarkNote, lyHBracket, lyStaffDef, lyTie,
lyNoteName, lyCentDev, lyFreqRatio, lyDur, lyNote, lyBeamOpen, lyBeamClosed,
consolidateNotes, consolidateRests;

```

```
// formats the data for the transcriber
formatMusicData = {arg rawMusicData;
    var maxSize, musicData;
    maxSize = 0;
    musicData = rawMusicData.collect({arg partData, p;
        var res;
        res = partData.collect({arg item, i;
            var freq, dur, amp, mult, insRef, sus, note, rest;
            # freq, dur, amp, mult, insRef = item;
            sus = dur * sign(amp);
            note = sus.collect({[freq, mult, insRef, i]});
            rest = if(p < rawMusicData.size, {(dur - sus).collect({[-1, -1, -1, i]})}, {[]});
            note ++ rest
        }).flatten;
        if(res.size > maxSize, {maxSize = res.size});
        res
    });

    //make them all the same length
    maxSize = maxSize.trunc(64) + 64;
    musicData = musicData.collect({arg partData, p; partData.extend(maxSize, partData.last)});
    musicData
};

// constants (spelling dictionary note names and octaves)
spellingDict = Dictionary.with(*
    [
        \major -> Dictionary.with(*
            [0, 7, 2, 9, 4, 11].collect({arg pc; pc->\sharps}) ++
            [5, 10, 3, 8, 1, 6].collect({arg pc; pc->\flats})
        ),
        \minor -> Dictionary.with(*
            [9, 4, 11, 6, 1, 8].collect({arg pc; pc->\sharps}) ++
            [2, 7, 0, 5, 10, 3].collect({arg pc; pc->\flats})
        )
    ]
);

lyNoteNameStr = Dictionary.with(*
    [
        \sharps -> ["c", "cis", "d", "dis","e", "f", "fis", "g", "gis", "a", "ais", "b"],
        \flats -> ["c", "des", "d", "ees","e", "f", "ges", "g", "aes", "a", "bes", "b"],
    ]
);

lyOctStr = [",,", ",", "", "'", "''", "'''", "''''"];

//define staff
lyStaffDef = {arg name, nameShort, nameMidi;
    "\\new Staff = \"" ++ name ++ "\" \\with { \n" ++
    "instrumentName = \"" ++ name ++ "\" \n" ++
    "shortInstrumentName = \"" ++ nameShort ++ "\" \n" ++
    "midiInstrument = #\"" ++ nameMidi ++ "\" \n" ++
    "\n}\n"
};

// add music preamble
lyFinalizeMusic = {arg lyStr, part, name, nameShort, nameMidi, clef;
    "\\new StaffGroup  \\with {\\remove \"System_start_delimiter_engraver\"}\n<<\n" ++
    lyStaffDef.value(name, nameShort, nameMidi)  ++
    "<<\n\n{ " +
    //"\n\\set Score.markFormatter = #format-mark-box-numbers " +
    "\\tempo 2 = 60\n" +
    "\\numericTimeSignature \\time 2/2\n" +
    "\\clef " ++ clef ++ "\n" ++ lyStr + "\\fermata" +
    " }>> \\bar \"|.\" \n} \n\n>>" ++
    "\n>>"
};

lyRelMarkNote = {arg root, lastRoot, part, clef;
    if(root[part][2] != [[1], [1]], {
        "\\stopStaff s8. \\startStaff \\clef" + clef + "s16 \n" ++
        "\\once \\override TextScript.color = #(rgb-color 0.6 0.6 0.6) \n " ++
        "\\tweak Accidental.color #(rgb-color 0.6 0.6 0.6) \n " ++
        "\\tweak NoteHead.color #(rgb-color 0.6 0.6 0.6) \n " ++
        lyNote.value(lastRoot[part][1], 1, lastRoot[part][0], nil, \sharps, true, true, false) +
        "\\hide c" ++ [nil, "", "'", "''"][part] ++ "8 \n "
    }, {
        "\\stopStaff s4. \\startStaff \\clef" + clef + "s16 \n"
    }) ++
    lyNote.value(root[part][3], 1, root[part][2], nil, \sharps, true, false, true)
};

lyHBracket = {arg fr, yOffset, sPair1, sPair2, edgeH1, edgeH2;
    "-\\tweak HorizontalBracket.Y-offset #" ++ yOffset ++ "\n " ++
    "-\\tweak HorizontalBracket.shorten-pair #'(" ++ sPair1 + "." + sPair2 ++") \n " ++
    "-\\tweak HorizontalBracket.edge-height #'("  ++ edgeH1 + "." + edgeH2 ++ ") \n " ++
    "-\\tweak HorizontalBracketText.text" + fr + "\\startGroup \n "
};

lyRelMark = {arg root, lastRoot, section, subsection;
    var sectionMark;
    sectionMark = "\\mark \\markup { \\bold \\override #'(box-padding . 0.5) \\box " ++ section ++ "." ++ subsection ++ " } \n";
    if((section == 1) && (subsection > 1),
        {
            "\\once \\override Score.RehearsalMark.self-alignment-X = #0 \n " ++
            "\\once \\override Score.RehearsalMark.Y-offset = #5 \n " ++
            "\\once \\override Score.RehearsalMark.X-offset = #1 \n " ++
            sectionMark
        }, {
            "\\mark \\markup { \n" ++
            "\\halign #-1 \n " ++
            "\\relMark ##{ { \n " ++
            "\\time 15/8 \n " ++
            "\\once \\override Staff.Clef.stencil = ##f \n " ++
            sectionMark ++

            lyRelMarkNote.value(root, lastRoot, 1, "bass") ++ "^\\markup{\\large \\raise #2 \"III\"}" ++

            lyHBracket.value(lyFreqRatio.value(root[1][4][2], nil, true, 0, false), 8.5, 1, 2, 1, 1) ++
            lyHBracket.value(lyFreqRatio.value(root[2][4][1], nil, true, 0, false), 5.5, 3, 3, 0, 0) ++

            "\\hide c16 \n " ++

            lyRelMarkNote.value(root, lastRoot, 2, "alto") ++ "^\\markup{\\large \\raise #2 \"II\"}" +
            "\\stopGroup \\hide c'16 \n " ++

            lyHBracket.value(lyFreqRatio.value(root[2][4][2], nil, true, 0, false), 5.5, 1, 3, 0, 0) ++

            lyRelMarkNote.value(root, lastRoot, 3, "treble") ++ "^\\markup{\\large \\raise #2 \"I\"}" +
            "\\stopGroup \\stopGroup \n " ++
            "\\hide c''16 \n " ++
            "}#}}"
    });
};
```

```
132
133 // barline and ossia definition
134 lyMeasureDef = {arg sectionData, insName, part, beat;
135     var ossia = "", barline = "|", break = "";
136     if(sectionData != nil, {
137         var root, lastRoot, section, subsection;
138         # root, lastRoot, section, subsection = sectionData;
139         ossia = lyRelMark.value(root, lastRoot, section, subsection);
140         barline = "\\bar \"||\"";
141         if(sectionData[4], {barline = "\\bar \"|.|\""});
142         if(sectionData[5], {barline = "\\bar \".|\""});
143     });
144     if((beat % 16) == 0, {break = "\\break \\noPageBreak"});
145     //for full score
146     //if((beat % (16 * 3)) == 0, {break = "\\pageBreak"});
147     //for parts
148     if((beat % (16 * 8)) == 0, {break = "\\pageBreak"});
149     if(beat != 0, {"}\n>>\n" + barline + break}, {""}) + "\n<<\n" ++ ossia + "{"
150 };
151
152 // add tie
153 lyTie = {"~"};
154
155 lyNoteName = {arg freq, spellingPref = \sharps;
156     if(freq != -1, {
157         lyNoteNameStr[spellingPref][((freq.cpsmidi).round(1) % 12)] ++
158         lyOctStr[((((freq).cpsmidi).round(1) / 12).asInteger - 2];
159     },{"r"});
160 };
161
162 lyCentDev = {arg freq, padding = true;
163     var centDev;
164     centDev = ((freq.cpsmidi - (freq.cpsmidi).round(1)) * 100).round(1).asInteger;
165     "^\\markup { " ++ if(padding, {"\\pad-markup #0.2 \""}, {"\""}) ++
166     if(centDev >= 0, {"+"}, {""}) ++ centDev.asString ++ "\"}"
167 };
168
169 lyFreqRatio = {arg freqRatioMult, ref, padding = true, lower = 3, attachedToNote = true;
170     var res, ratio;
171     res = "\\markup {" + if(attachedToNote, {""}, {"\\normalsize"}) +
172     "\\lower #" ++ lower + if(padding, {"\\pad-markup #0.2 "}, {" "});
173     ratio = "\"" ++ freqRatioMult[0].product.asInteger ++ "/" ++ freqRatioMult[1].product.asInteger ++ "\" }";
174     res = if(ref != nil,
175         {
176             res ++ "\\concat{ \"" ++ [nil, "III", "II", "I"][ref] ++ "\"\\normal-size-super " ++ ratio ++ "}"
177         }, {
178             res ++ ratio
179         }
180     );
181     if(attachedToNote, {"-" ++ res}, {res})
182 };
183
184
185 lyNote = {arg freq, noteLength, freqRatioMult, ref, spellingPref = \sharps, addMarkup = true, frHide = false, padding = true;
186     lyNoteName.value(freq, spellingPref) ++
187     lyDur.value(noteLength) ++
188     if(addMarkup, {
189         "<MARKUP" ++
190         lyCentDev.value(freq, padding) ++
191         if(frHide, {""}, {lyFreqRatio.value(freqRatioMult, ref, padding)}) ++
192         "MARKUP>"
193     }, {""})
194 };
195
196 lyDur = {arg noteLength;
197     switch(noteLength, 1, {"16"}, 2, {"8"}, 3, {"8."}, 4, {"4"});
198 };
199
200 lyBeamOpen = {"["};
201
202 lyBeamClosed = {"]"};
203
204 consolidateNotes = {arg lyStr, part;
205     var noteRegex, markupRegex, fullNoteRegex, restRegex, fullRestRegex, res;
206     noteRegex = "(?<n>[a-g](?:es|is)?(?:[,']*?)?4)";
207     markupRegex = if(part != 0, {"(<MARKUP.{75,85}MARKUP>)?"}, {"(<MARKUP.{75,115}MARKUP>)?"});
208     fullNoteRegex = noteRegex ++ markupRegex ++ "(?:\\h+~\\h+\\k<n>)";
209     restRegex = "(?<r>r4)";
210     fullRestRegex = "(?<r>r4)(?:(\\h+)\\k<r>)";
211     res = lyStr;
212     [6, 4, 3, 2].do({arg len;
213         [fullNoteRegex, fullRestRegex].do({arg regex;
214             res.findRegexp(regex ++ "{" ++ (len-1) ++ "}").clump(3).do({arg match;
215                 var word, note, markup, lyDur;
216                 word = match[0][1];
217                 note = match[1][1];
218                 markup = match[2][1];
219                 lyDur = switch(len, 6, {"1."}, 4, {"1"}, 3, {"2."}, 2, {"2"});
220                 res = res.replace(word, note.replace("4", lyDur) ++ markup)});
221         });
222     });
223     res.replace("<MARKUP", "").replace("MARKUP>", "");
224 };
225
226 ~transcribe = {arg rawMusicData, sectionData, seed;
227     var basePath, scoreFile, musicData, insData, insNames, insNamesShort, insMidi, insClef;
228
229     basePath = ~dir +/+ ".." +/+ "lilypond" +/+ "seed_" ++ seed;
230     basePath.mkdir;
231     (basePath +/+ "includes").mkdir;
232
233     scoreFile = File(basePath +/+ "tkam_score.ly".standardizePath,"w");
234     scoreFile.write(File.readAllString(basePath +/+ ".." +/+ "template" +/+ "tkam_score_template.ly").replace("seed: xxx", "seed: " ++ seed));
235     scoreFile.close;
236
237     musicData = formatMusicData.value(rawMusicData);
238
239     insData = [
240         ["*", "*", "clarinet", "\"treble_8\""],
241         ["III", "III", "clarinet", "bass"],
242         ["II", "II", "clarinet", "alto"],
243         ["I", "I", "clarinet", "treble"]
244     ];
245
246     insNames = insData.slice(nil, 0);
247     insNamesShort = insData.slice(nil, 1);
248     insMidi = insData.slice(nil, 2);
249     insClef = insData.slice(nil, 3);
250
251     musicData.do({arg part, p;
252         var lyFile, lyStr, lastMusAtom, measureCount, spellingPref,
253         tmpSectionData, pcRoot, partLookup, quality;
254
255         //create file
256         //for full score
```

```
        //lyFile = File(basePath +/+ "includes" +/+ "part_" ++ ["star", "III", "II", "I"][p] ++ ".ly".standardizePath,"w");
        //for parts
        lyFile = File(basePath +/+ "includes" +/+ "part_" ++ ["star", "III", "II", "I"][p] ++ "_8systemsperpage.ly".standardizePath,"w");

        //start lypond directives
        lyStr = "";
        lastMusAtom = nil;
        measureCount = 0;
        spellingPref = \sharps;
        tmpSectionData = nil;
        part.clump(4).do({arg beat, i;
            var gSum;
            gSum = 0;
            beat.separate({arg a, b;
                ((a[0] != -1) || (b[0] != -1)) && (a != b)}).do({arg group, g;
                var noteLength, curMusAtom, freq, freqRatioMult, ref, isSame, isRest, isFirst, isLast,
                isTied, isMeasureBound, isBeamStart, isBeamEnd;

                noteLength = group.size;
                gSum = gSum + noteLength;
                curMusAtom = group[0];
                freq =  curMusAtom[0];
                freqRatioMult = curMusAtom[1];
                ref = curMusAtom[2];
                # isSame, isRest, isFirst, isLast = [curMusAtom == lastMusAtom, freq == -1, g == 0, gSum == 4];
                # isTied, isMeasureBound = [isSame && isRest.not, isFirst && ((i % 4) == 0)];
                # isBeamStart, isBeamEnd = [(noteLength != 4) && isFirst, (noteLength != 4) && isLast];

                //add ties
                if(isTied, {lyStr = lyStr + lyTie.value});

                //add barline and ossia definition
                if(isMeasureBound, {lyStr = lyStr + lyMeasureDef.value(sectionData[i], insNames[p], p, i)});

                //add note data
                if(sectionData[i] != nil, {
                    tmpSectionData = sectionData[i];
                });
                if(isTied.not, {
                    partLookup = if((p != 0) || [1, 2, 3].includes(ref).not , {p}, {ref});
                    pcRoot = ((tmpSectionData[0][partLookup][3].cpsmidi).round(1) % 12).asInteger;
                    quality = if(tmpSectionData[0][partLookup][1][2] == [[ 1, 5 ], [ 1, 2, 2 ]], {\major}, {\minor});
                    spellingPref = spellingDict[quality][pcRoot];
                    if(p == 0, {[(i / 4).asInteger, partLookup, pcRoot, quality]});
                });

                lyStr = lyStr + lyNote.value(freq, noteLength, freqRatioMult, ref, spellingPref, isSame.not && isRest.not);

                //beam group
                if(isBeamStart, {lyStr = lyStr ++ lyBeamOpen.value});
                if(isBeamEnd, {lyStr = lyStr ++ lyBeamClosed.value});

                lastMusAtom = curMusAtom;
            });
        });

        //wrap music and add staff definitions
        lyStr = lyFinalizeMusic.value(lyStr, p, insNames[p], insNamesShort[p], insMidi[p], insClef[p]);

        //consolidate notes and rests
        lyStr = consolidateNotes.value(lyStr, p);

        //write file
        lyFile.write(lyStr);
        lyFile.close;
    });
};


//------------GENERATE SCORE DATA------------
~genScoreData = {arg ensData;
    var res;
    res = ensData.collect({arg partData;
        partData.flop.collect({arg data, d; if(d == 1, {data.differentiate ++ [10]}, {[0] ++ data})})
    });
    res.collect({arg part; part.flop})
};
)
```

tkam_gui.scd

```
(
var clockStringFunc, metronomeStringFunc, metronomeColorFunc, updateTransport, updateSection, updateSubsection,
buildGenerator, buildMetronome, buildTransport, buildTempoControl, buildMasterFader, buildTrackFader,
buildMasterView, buildFaderView, buildHelpView, currentSection = 1, currentSubsection = 1;

// these funcs update the elements of the transport panel
clockStringFunc = {
    arg measure, beat;
    var measureString, beatString, leadSpace;
    measureString = measure.asInteger.asString;
    beatString = beat.asInteger.asString;
    leadSpace = (3 - measureString.size).collect({" "}).join;
    leadSpace ++ measureString ++ "." ++ beatString
};

// [-30, -105, -104] and [-30, -105, -113] are unicode inverse bullet and normal bullet, respectively
metronomeStringFunc = { arg beat; if(beat == 1,
    {[-30, -105, -104].collect({arg int; int.asAscii}).as(String)},
    {[-30, -105, -113].collect({arg int; int.asAscii}).as(String)})};
metronomeColorFunc = { arg beat; if(beat == 1, {Color.red},{Color.black})};

updateTransport = {arg clock, metronome, sectionDisplay, measure, beat, section, subsection;
    sectionDisplay.string = "section: " ++ section.asInteger ++ "." ++ subsection.asInteger;
    clock.string = clockStringFunc.value(measure, beat);
    metronome.stringColor = metronomeColorFunc.value(beat);
    metronome.string = metronomeStringFunc.value(beat);
    {0.75.wait; {metronome.string = ""}.defer}.fork(~tempoClock, quant: 0);
}.inEnvir;

buildGenerator = {arg view;
    var ranSeed;
    HLayout(
        ranSeed = TextField(view).string_("19800725"),
        Button(view).states_([["reset seed"]]).action_({ ranSeed.string = "19800725"}.inEnvir),
        Button(view).states_([["random seed"]]).action_({ ranSeed.string = 50000000.rand.asString}.inEnvir),
        Button(view).states_([["generate"]]).action_({
            {~genAll.value(ranSeed.string.asInteger); ~appStatus.string = "status: ready"}.fork(AppClock);
            ~appStatus.string = "status: generating"}.inEnvir),
        [~appStatus = StaticText(view).string_("status: ready"), stretch: 1],
        Button(view).states_([["transcribe"]]).action_({
            {~transcribe.value(~scoreData, ~sectionData, ranSeed.value); ~appStatus.string = "status: ready"}.fork(AppClock);
            ~appStatus.string = "status: transcribing"}.inEnvir),
```

```
        Button(view).states_([["bounce audio"]]).action_({
            {~bounceAudio.value(ranSeed.value); ~appStatus.string = "status: ready"}.fork(AppClock);
            ~appStatus.string = "status: bouncing audio"}.inEnvir),
        nil)
};

buildMetronome = {arg win;
    var clock, metronome, layout;

    clock = StaticText(win).string_("  1.1").font_(Font("Liberation Mono", 200));
    metronome = StaticText(win).string_([-30, -105, -104].collect({arg int; int.asAscii})
            .as(String)).font_(Font("Liberation Mono", 300)).stringColor_(Color.red);

    layout = HLayout(
        clock,
        StaticText(win).string_("|").font_(Font("Liberation Mono", 200)),
        metronome
    );

    [clock, metronome, layout]
};

updateSection = {arg mod, clock, metronome, sectionDisplay, refresh = true, indirect = false;
    var changeSection;
    case
    {(currentSubsection > 1) && (mod < 0)} {
        currentSubsection = 1;
    }
    {(currentSubsection <= 1) && (mod < 0) && (currentSection > 1)} {
        currentSection = currentSection + mod;
        if(indirect, {
            currentSubsection = ~sectionNavDict[[currentSection, 1]][1]
        }, {
            currentSubsection = 1;
        })
    }
    {(mod > 0) && (~sectionNavDict[[currentSection + mod, 1]] != nil)} {
        currentSection = currentSection + mod;
        currentSubsection = 1;
    };

    if(refresh, {
        updateTransport.value(clock, metronome, sectionDisplay,
            ~sectionNavDict[[currentSection, currentSubsection]][0], 1,
            currentSection, currentSubsection
        );
    });
};

updateSubsection = {arg mod, clock, metronome, sectionDisplay, refresh = true;
    if(~sectionNavDict[[currentSection, currentSubsection + mod]] != nil, {
        currentSubsection = currentSubsection + mod;
        if(refresh, {
            updateTransport.value(clock, metronome, sectionDisplay,
                ~sectionNavDict[[currentSection, currentSubsection]][0], 1,
                currentSection, currentSubsection
            );
        });
    }, {
        updateSection.value(mod, clock, metronome, sectionDisplay, refresh, true)
    })
};

buildTransport = {arg win, view, clock, metronome, preampBusses, accompBusses, postampBusses;
    var sec, subsec, sectionDisplay, layout, player;

    sectionDisplay = StaticText(win).string_("section: 1.1").font_(Font("Liberation Mono", 70));

    OSCFunc({ arg msg, time;
        {
            var measure, beat, section, subsection;
            # measure, beat, section, subsection = msg[3..];
            currentSection = sec = section.asInteger;
            currentSubsection = subsec = subsection.asInteger;
            updateTransport.value(clock, metronome, sectionDisplay, measure, beat, section, subsection);
        }.inEnvir.defer;
    },'/measureClock_' ++ ~hash, s.addr);

    layout = HLayout(
        Button(view).states_([["<<", Color.black]]).action_({arg pState; updateSection.value(-1, clock, metronome, sectionDisplay)}.inEnvir),
        Button(view).states_([["<", Color.black]]).action_({arg pState; updateSubsection.value(-1, clock, metronome, sectionDisplay)}.inEnvir),
        Button(view).states_([["play", Color.black], ["stop", Color.black, Color.grey]]).action_({arg pState;
            if(pState.value == 1, {
                player = {
                    var startMeasure = ~sectionNavDict[[currentSection, currentSubsection]][0] - 1;
                    ~patternProxy.source = ~genPlayablePatterns.value(startMeasure, ~patterns, preampBusses, accompBusses, postampBusses);
                    Pbind(\instrument, \click_ ++ ~hash, \beat, Pseq([1, 2, 1, 2]), \dur, 1).play(~tempoClock, quant: 0);
                    [1, 2, 1, 2].do({arg beat;
                        {
                            metronome.stringColor = metronomeColorFunc.value(beat);
                            metronome.string = metronomeStringFunc.value(beat);
                        }.defer;
                        0.75.wait;
                        {metronome.string = ""}.defer;
                        0.25.wait;
                    });
                    ~patternProxy.play(~tempoClock, quant: 0)
                }.fork(~tempoClock, quant: 0)
            }, {
                ~patternProxy.pause;
                //player.stop;
                updateTransport.value(clock, metronome, sectionDisplay,
                    ~sectionNavDict[[currentSection, currentSubsection]][0], 1,
                    currentSection, currentSubsection);
            });
        }.inEnvir),
        Button(view).states_([[">", Color.black]]).action_({arg pState; updateSubsection.value(1, clock, metronome, sectionDisplay)}.inEnvir),
        Button(view).states_([[">>", Color.black]]).action_({arg pState; updateSection.value(1, clock, metronome, sectionDisplay)}.inEnvir), nil,
        sectionDisplay, nil);
    [sectionDisplay, layout]
};

buildTempoControl = {arg view;
    var layout, tempoField, address, updateSection;
    layout = HLayout(
        tempoField = TextField(view).string_("60").action_({arg v;
            var tempo = v.value.asInteger; ~tempoClock.tempo = tempo / 60}.inEnvir),
        Button(view).states_([["set tempo"]]).action_({arg v; ~tempoClock.tempo = tempoField.string.asInteger / 60}.inEnvir),
        [StaticText(view).string_("  "), stretch: 1]);
    [layout, tempoField]
};

buildMasterFader = {arg view;
    var trackIndicators, layout, volSlider, muteButton, outMenu;

```

```
    trackIndicators = {LevelIndicator()} ! 2;

    OSCFunc.new({arg msg;
        {trackIndicators[0].value = msg[3].ampdb.linlin(-50, 0, 0, 1)}.defer;
        {trackIndicators[1].value = msg[4].ampdb.linlin(-50, 0, 0, 1)}.defer
    }, '/masterLevels_' ++ ~hash, s.addr);

    layout = HLayout([
        VLayout(
            HLayout(
                volSlider = Slider(view).value_(0.8).action_(
                    {arg v; var masterVol = v.value * 1.25; ~play.set(\masterVol, masterVol)}.inEnvir),
                trackIndicators[0],
                trackIndicators[1]),
            muteButton = Button(view).states_([["mute", Color.black], ["mute", Color.black, Color.grey]]).action_(
                {arg v; var masterMute = (1 - v.value).abs; ~play.set(\masterMute, masterMute)}.inEnvir),
            StaticText(view).string_("out").align_(\center),
            outMenu = PopUpMenu(view).items_((1..16).collect({arg o; o + "-" + (o + 1)})).action_(
                {arg v; var out = v.value.postln; ~play.set(\masterOut, out)}.inEnvir),
            StaticText(view).string_("master").align_(\center)
    ), stretch: 2], nil);
    [layout, volSlider, muteButton, outMenu]
};

buildTrackFader = {arg view, name, index;
    var trackIndicator, netAddr, layout, volSlider, soloButton, muteButton, panKnob, outMenu;

    netAddr = NetAddr("127.0.0.1", NetAddr.langPort);
    trackIndicator = LevelIndicator();

    OSCFunc.new({arg msg; {trackIndicator.value = msg[3].ampdb.linlin(-50, 0, 0, 1)}.defer},
        '/trackLevel_' ++ index ++ "_" ++ ~hash, s.addr);

    layout = HLayout(
        VLayout(
            HLayout(
                volSlider = Slider(view).value_(0.8).action_(
                    {arg v; var vol = v.value * 1.25; ~play.set(\vol_ ++ index, vol)}.inEnvir),
                trackIndicator),
            soloButton = Button(view).states_([["solo", Color.black], ["solo", Color.black, Color.grey]]).action_(
                {netAddr.sendMsg("/soloer_" ++ ~hash, index)}.inEnvir).value_(0),
            muteButton = Button(view).states_([["mute", Color.black], ["mute", Color.black, Color.grey]]).action_(
                {arg v; var mute = (1 - v.value).abs;
                    ~play.set(\mute_ ++ index, mute)}.inEnvir).valueAction_(if(index < 4, {1}, {0})),
            VLayout(
                StaticText(view).string_("pan").align_(\center),
                panKnob = Knob(view).action_({arg v; var pan = v.value * 2 - 1; ~play.set(\pan_ ++ index, pan)}.inEnvir).valueAction_(0.5)
            ),
            StaticText(view).string_("out").align_(\center),
            outMenu = PopUpMenu(view).items_(["master"] ++ (1..16)).action_(
                {arg v; var out = v.value; ~play.set(\out_ ++ index, out)}.inEnvir).valueAction_(if(index < 6, {0}, {3})),
            StaticText(view).string_(name).align_(\center)
            //StaticText(view).string_("output").align_(\center),
        ),
        nil);
    [layout, volSlider, soloButton, muteButton, panKnob, outMenu]
};

buildMasterView = {arg win, preampBusses, accompBusses, postampBusses;
    var view, generatorLayout, clock, metronome, metronomeLayout, transportLayout,
    tempoContol, auxControlsLayout, countOff, ranSeed, order, tempo, sectionDisplay, address;

    view = View(win);
    generatorLayout = buildGenerator.value(view);
    # clock, metronome, metronomeLayout = buildMetronome.value(win);
    # sectionDisplay, transportLayout = buildTransport.value(win, view, clock, metronome, preampBusses, accompBusses, postampBusses);
    tempoContol = buildTempoControl.value(view);
    auxControlsLayout = tempoContol[0];

    view.layout_(
        HLayout(
            [
                VLayout(
                    metronomeLayout,
                    [StaticText(view).string_("  "), stretch: 1],
                    transportLayout,
                    [StaticText(view).string_("  "), stretch: 1],
                    auxControlsLayout,
                    [StaticText(view).string_("  "), stretch: 1],
                    generatorLayout),
                alignment: \top
            ]
        )
    );
    [view, tempoContol[1]]
};

buildFaderView = {arg win, tempoField;
    var view, masterIndicators, trackIndicators, master, tracks, openButton, basePath, saveButton;
    var partAbbr = ["*", "III", "II", "I", "accomp_II", "accomp_I", "click"];
    var trackNames = ["*", "III", "II", "I", "accomp_II", "accomp_I", "click"];
    var partVols, partMutes, partPans;
    var masterMute, masterVol;
    var netAddr = NetAddr("127.0.0.1", NetAddr.langPort);
    var player = ~play;

    // set initial mixer values
    partVols = [1, 1, 1, 1, 1, 1];
    partMutes = [0, 1, 1, 1, 1, 0];
    partPans = [0, 0, 0, 0, 0, 0];
    masterMute = 1;
    masterVol = 1;

    view = View(win);
    masterIndicators = {LevelIndicator()} ! 2;
    trackIndicators = {LevelIndicator()} ! 6;

    master = buildMasterFader.value(view);
    tracks = {arg part;
        buildTrackFader.value(view, trackNames[part], part);
    } ! 7;

    OSCFunc.new({arg msg; {
        tracks.slice(nil, 3).do({arg mute, m;
            if(tracks[msg[1]][2].value == 1, {
                mute.valueAction = if(msg[1] == m, {0}, {1});
                tracks[m][2].value = if(msg[1] != m, {0}, {1})
            }, {
                mute.valueAction = 0
            });
        });
    }.defer}, '/soloer_' ++ ~hash, netAddr);

    basePath = ~dir +/+ ".." +/+ "mixer_settings";
```

```
    openButton = Button(view).states_([["open", Color.black]]).action_({
        Dialog.openPanel({ arg path;
            var settings;
            settings = File.readAllString(path).parseJSON;
            tempoField.valueAction = settings["tempo"];
            master[1].valueAction = settings["master_volume"];
            master[2].valueAction = settings["master_pan"];
            master[3].valueAction = settings["master_out"];
            settings["track_volumes"].do({arg val, v; tracks[v][1].valueAction = val});
            settings["track_solos"].do({arg val, v; tracks[v][2].valueAction = val});
            settings["track_mutes"].do({arg val, v; tracks[v][3].valueAction = val});
            settings["track_pans"].do({arg val, v; tracks[v][4].valueAction = val});
            settings["track_outs"].do({arg val, v; tracks[v][5].valueAction = val});
        },{}, false, basePath);
    });

    saveButton = Button(view).states_([["save", Color.black]]).action_({
        Dialog.savePanel({ arg path;
            var settings, file;
            settings = "{\n";
            settings = settings ++ "\"tempo\": " ++ tempoField.string ++ ",\n";
            settings = settings ++ "\"master_volume\": " ++ master[1].value ++ ",\n";
            settings = settings ++ "\"master_mute\": " ++ master[2].value ++ ",\n";
            settings = settings ++ "\"master_out\": " ++ master[3].value ++ ",\n";
            settings = settings ++ "\"track_volumes\": [" ++ tracks.collect({arg track; track[1].value}).join(",") ++ "],\n";
            settings = settings ++ "\"track_solos\": [" ++ tracks.collect({arg track; track[2].value}).join(",") ++ "],\n";
            settings = settings ++ "\"track_mutes\": [" ++ tracks.collect({arg track; track[3].value}).join(",") ++ "],\n";
            settings = settings ++ "\"track_pans\": [" ++ tracks.collect({arg track; track[4].value}).join(",") ++ "],\n";
            settings = settings ++ "\"track_outs\": [" ++ tracks.collect({arg track; track[5].value}).join(",") ++ "]\n";
            settings = settings ++ "}";
            file = File(path, "w");
            file.write(settings);
            file.close;
        },{}, basePath);
    });

    view.layout_(HLayout(HLayout(master[0], nil, *tracks.slice(nil, 0)), VLayout(nil, saveButton, openButton)))
};

buildHelpView = {arg win;
    TextView(win).string_(File.readAllString(~dir +/+ "tkam_readme.scd")).editable_(false);
};

~generateGUI = {arg preampBusses, accompBusses, postampBusses;
    var win, tabButtonReset, transportButton, mixerButton, helpButton, masterConrol, tempoControl, masterView, faderView, helpView, tabs;
    win = Window("to kill a monarch", Rect(500, 500, 1100, 575), false).front;
    tabButtonReset = {transportButton.value = 1; mixerButton.value = 1; helpButton.value = 1};
    masterConrol = buildMasterView.value(win, preampBusses, accompBusses, postampBusses);
    masterView = masterConrol[0];
    tempoControl = masterConrol[1];
    faderView = buildFaderView.value(win, tempoControl);
    helpView = buildHelpView.value(win);

    win.layout = VLayout(
        HLayout(
            HLayout(
                [
                    transportButton = Button().states_([["transport", Color.white, Color.grey], ["transport", Color.black]]).action_(
                        {tabButtonReset.value; transportButton.value = 0; tabs.index = 0 }.inEnvir).value_(0), stretch: 1
                ], [
                    mixerButton = Button().states_([["mixer", Color.white, Color.grey], ["mixer", Color.black]]).action_(
                        {tabButtonReset.value; mixerButton.value = 0; tabs.index = 1 }.inEnvir).value_(1), stretch: 1
                ]
            ),
            helpButton = Button().states_([["help", Color.white, Color.grey], ["help", Color.black]]).action_(
                {tabButtonReset.value; helpButton.value = 0; tabs.index = 2 }.inEnvir).value_(1)
        ),
        tabs = StackLayout(masterView, faderView, helpView));
};
)
```

tkam_score_template.ly

```
%\version "2.19.83"
\version "2.24.1"

#(define (override-color-for-all-grobs color)
  (lambda (context)
   (let loop ((x all-grob-descriptions))
    (if (not (null? x))
     (let ((grob-name (caar x)))
      (ly:context-pushpop-property context grob-name 'color color)
      (loop (cdr x)))))))

 #(define-markup-command (relMark layout props mus) (ly:music?)
    #:properties ((size -2))
    (interpret-markup layout props
      #{
        \markup {
          \score {
            \new Staff { $mus }
            \layout {
              \context {
                \Staff
                \remove Time_signature_engraver
                fontSize = #-2
                \hide Stem
                \override TextScript.outside-staff-priority = ##f
                \override StaffSymbol.staff-space = #(magstep -2)
                \override StaffSymbol.thickness = #(magstep -2)
                \override TextScript.self-alignment-X = #-0.4
                \override TextScript.staff-padding = #1
              }
              \context {
                \Score
                proportionalNotationDuration = #(ly:make-moment 1/16)
                \remove "Separating_line_group_engraver"
                \override SpacingSpanner.strict-note-spacing = ##t
                \override RehearsalMark.self-alignment-X = #-1
                \override RehearsalMark.Y-offset = #10
                \override RehearsalMark.X-offset = #10
              }
              \context {
                \Voice
                \consists "Horizontal_bracket_engraver"
                \override HorizontalBracket.direction = #UP
              }
              indent = 0
              line-width = 4\cm
            }
          }
        }
      #}))
```

```
51
52
53  \paper {
54    #(set-paper-size "a4" 'portrait)
55    top-margin = 1 \cm
56    bottom-margin = 1 \cm
57    left-margin = 2 \cm
58    ragged-bottom = ##t
59
60    top-system-spacing =
61    #'((basic-distance . 15 )
62    (minimum-distance . 15 )
63    (padding . 0 )
64    (stretchability . 0))
65
66    system-system-spacing =
67    #'((basic-distance . 35 )
68    (minimum-distance . 35 )
69    (padding . 0 )
70    (stretchability . 0))
71
72    last-bottom-spacing =
73    #'((basic-distance . 10 )
74    (minimum-distance . 10 )
75    (padding . 0 )
76    (stretchability . 0))
77
78    %systems-per-page = 3
79    first-page-number = 1
80    print-first-page-number = ##t
81
82    % for lilypond version 2.19.83
83    %print-page-number = ##t
84    %oddHeaderMarkup = \markup { \fill-line { \line { \on-the-fly #not-first-page {\pad-markup #2 { \concat {\italic {"to kill a monarch "} (seed: xxx)}}}}}}
85    %evenHeaderMarkup = \markup { \fill-line { \line { \on-the-fly #not-first-page {\pad-markup #2 { \concat {\italic {"to kill a monarch "} (seed: xxx)}}}}}}
86    %oddFooterMarkup = \markup { \fill-line {
87    %  \concat {
88    %    "-"
89    %    \fontsize #1.5
90    %    \on-the-fly #print-page-number-check-first
91    %    \fromproperty #'page:page-number-string
92    %    "-"}}}
93    %evenFooterMarkup = \markup { \fill-line {
94    %  \concat {
95    %    "-"
96    %    \fontsize #1.5
97    %    \on-the-fly #print-page-number-check-first
98    %    \fromproperty #'page:page-number-string
99    %    "-"}}}
100 %}
101
102   % for lilypond version 2.24.1
103   print-page-number = ##t
104   oddHeaderMarkup = \markup { \fill-line { \line { \unless \on-first-page {\pad-markup #2 { \concat {\italic {"to kill a monarch "} (seed: xxx)}}}}}}
105   evenHeaderMarkup = \markup { \fill-line { \line { \unless \on-first-page {\pad-markup #2 { \concat {\italic {"to kill a monarch "} (seed: xxx)}}}}}}
106   oddFooterMarkup = \markup { \fill-line {
107     \concat {
108       "-"
109       \fontsize #1.5
110       \fromproperty #'page:page-number-string
111       "-"}}}
112   evenFooterMarkup = \markup { \fill-line {
113     \concat {
114       "-"
115       \fontsize #1.5
116       \fromproperty #'page:page-number-string
117       "-"}}}
118 }
119
120 \header {
121   title = \markup { \italic {to kill a monarch}}
122   composer = \markup \right-column {"michael winter" "(berlin, germany; 2021)"}
123   poet = "seed: 19800725"
124   tagline = ""
125 }
126
127 \header {
128   title = \markup { \italic {to kill a monarch}}
129   composer = \markup \right-column {"michael winter" "(berlin, germany; 2021)"}
130   poet = "seed: xxx"
131   tagline = ""
132 }
133
134 #(set-global-staff-size 11)
135
136 \layout {
137   indent = 0.0\cm
138   line-width = 17.5\cm
139   ragged-last = ##f
140   ragged-right = ##f
141
142   \context {
143     \Score
144       \override BarNumber.stencil = #(make-stencil-circler 0.1 0.25 ly:text-interface::print)
145       \override Stem.stemlet-length = #0.75
146       proportionalNotationDuration = #(ly:make-moment 1/16)
147       \remove "Separating_line_group_engraver"
148       \override RehearsalMark.self-alignment-X = #-1
149       \override RehearsalMark.Y-offset = #10
150       \override RehearsalMark.X-offset = #-8
151       %\override RehearsalMark.outside-staff-priority = #0
152       %added for new lilypond
153       rehearsalMarkFormatter = #format-mark-box-numbers
154   }
155   \context {
156     \Staff
157
158     \override VerticalAxisGroup.staff-staff-spacing =
159       #'((basic-distance . 20 )
160       (minimum-distance . 20 )
161       (padding . 0 )
162       (stretchability . 0))
163
164       \override VerticalAxisGroup.default-staff-staff-spacing =
165       #'((basic-distance . 20 )
166       (minimum-distance . 20 )
167       (padding . 0 )
168       (stretchability . 0))
169       \override TextScript.staff-padding = #2
170       \override TextScript.self-alignment-X = #0
171   }
172   \context {
173     \StaffGroup
174     \name "SemiStaffGroup"
175     \consists "Span_bar_engraver"
```

```
    \override SpanBar.stencil =
      #(lambda (grob)
        (if (string=? (ly:grob-property grob 'glyph-name) "|")
            (set! (ly:grob-property grob 'glyph-name) ""))
        (ly:span-bar::print grob))
  }
  \context {
    \Score
    \accepts SemiStaffGroup
  }
}

\midi { }


\score{
\new Score
 <<
    \new SemiStaffGroup {
   <<
    \include "includes/part_I.ly"
    \include "includes/part_II.ly"
    \include "includes/part_III.ly"
   >>
    }
    \include "includes/part_star.ly"
 >>

  \layout{}
  \midi{}
}
```