Drei kleine Studien nach Morton Feldman (für drei höhere und drei tiefere Instrumentalstimmen)

Markus Lepper, op. el. 20

Oktober 2023

(print version 20240131)

Einleitung

Die Kompositionen “Projection 1” bis “Projection 5” von Morton Feldman aus den Jahren 1950-51 gelten als Meilensteine der Avantgarde des mittleren zwanzigsten Jahrhunderts: Sie sind unter den ersten, die sich einer weitgehend graphischen Notation bedienen und die Wahl der Tonhöhen den Ausführenden freistellen. [3, 1, 2]

Beim genaueren Hinsehen und gar der Modellierung der möglichen Interpretationsweisen als exakte Algorithmen (als Beitrag für die Konferenz “Tenor 24”) stellte sich bald heraus, dass ein Ensemble von Musizierenden sich entscheiden muss zwischen sehr verschiedenen Haltungen gegenüber der geforderten “Improvisation”: Angefangen von einer möglichen Beschränkung des Gesamtvorrates an “erlaubten Tonklassen”, über unterschiedlich stark empfundene und ausgeübte Verantwortlichkeit für den allgemeinen “Ausdruck und Klangcharakter”, über verschiedene Strategien zur “Wiederholungsvermeidung” (nur autark die eigenen Handlungen planend, oder möglichst das Gesamtensemble ablauschend und berücksichtigend) bis hin zum “Vorfabrizieren” von “fest notierten” Interpretationen, wie es in vielen Werken der damaligen und heutigen Avantgarde bei derart freien Notaten ja durchaus erlaubt oder gar vorgeschrieben ist.

Die folgenden drei kleinen Studien versuchen, verschiedene Kombinationen all dieser Aspekte zu realisieren, und damit in erster Linie auch verschiedene “Grade der” oder “Haltungen zur” Improvisation. Sie sind drei Sätze einer zusammenhängenden Komposition und gemeint als didaktisches Experiment für Ausführende wie auch für Zuhörerinnen.

Der grundlegende Klangcharakter ist der der Feldman’schen Vorlagen. In den Ecksätzen wird auch deren Notation direkt den Ausführenden vorgelegt, allerdings mit Ausführungsanweisungen, die über das damals von Feldman Vorgeschriebene hinausgehen. Im Mittelsatz wird die Haltung eingenommen, dass ein graphisch notierter Satz für die Ausführenden vorher “auskomponiert” wird. Im letzten Satz können diese das selbst tun, wenn sie wollen. Im ersten Satz ist das verboten und rein reaktives Spielen wird verlangt.

Zur Notation

Diese Partitur benutzt exakt die von Feldman in “Projections 1–5” eingeführte Notation:

Die Instrumente/Stimmen stehen ganz konventionell untereinander, die Zeit verläuft ganzu konventionell von links nach rechts. Jeder Takt wird durch vertikale Striche begrenzt, er enthält vier Viertel. Jeder Kasten ist ein Ereignis: das Spielen genau einer Tonhöhe. Die vertikale Höhe des Kastens innerhalb des Systems gibt dessen “Register” an: hoch, mittel, tief. Die

Interpetinnen müssen diese Register jeweils für ihr Instrument definieren. Weitergehend als Feldman fordern wir, dass zwischen ihnen eine Lücke von mindestens einer Quarte liegen muss. Die Register brauchen nicht jeweils eine ganze Oktave umfassen, aber mindestens drei Töne des (je Satz anderen) Tonklassenvorrates.

Einsatz- und Endzeitpunkte der Ereignisse liegen immer auf ganzen Vierteln, die aber in der Graphik nicht explizit dargestellt sind, sondern nur per Proportion.[1] Die allgemeine Laustärke ist wie bei den Vorbildern durchgehend piano, das Tempo ungefähr MM 72.

Die hier vorgelegten Graphiken sind durch eine knappe Software aus den entsprechenden tscore-Quellen generiert. (275 lines of code (loc) das Modell, knapp 400 loc das Rendering) [4, 5, 6] Die tscore-Quellen sind weiter unten abgedruckt.

Zur Ausführung

Die Sätze haben sechs Stimmen: drei "eher hohe" und drei "eher tiefe", ungefähr entsprechend den Flöten und Violoncelli der Vorlagen Feldmans. Diese könnnen durch je ein Instrument dargestellt werden, oder je zwei Stimmen durch ein Klavierinstrument.

Die Oktavlagen im festnotierten zweiten Satz können, wenn nötig, verschoben werden, allerdings muss die relative Lage der Ereignisse zueinander gewährleistet bleiben.

Zum ersten Satz

Dieser wird mit (im Vergleich zur historischen Vorlage) mit nur wenig stärkeren Einschränkungen aus einer graphischen Partitur gespielt. Die konkreten Tonhöhen müssen improvisatorisch "in real-time" ausgewählt werden. Festgelegt sind wie in der Vorlage (a) grundsätzlich leise Dynamik, (b) das Tempo MM 72. Dazu kommen (c) die Einschränkung der erlaubten Tonklassen auf des, es, ges, as und b (= "schwarze Klaviertasten"). und (d) die grundlegende Regel:

> Die Wiederholung einer Tonklasse soll so lange vermieden werden, wie möglich. Dazu sollen alle Mitwirkenden die eigenen gespielten Töne und die der anderen memorieren. Sobald eine Wiederholung unausweichlich wird, beginnt ein neuer Zyklus, in dem zunächst wieder beliebig ausgewählt werden darf, aber wieder unter längstmöglicher Vermeidung einer Wiederholung.

[1] = NON-nota.nonuantibus.

Diese Regel wird aber wegen einfacherer Ausführung und besserer Hörbarkeit *innerhalb der Stimmgruppe* angewandt, also unabhängig durch die Spielerinnen von h1, h2 und h3 und denen von t1, t2 und t3.[2] Nach der Folge "des, ges, es, b" kann z.B. ein Instrument an der Reihe sein, dass im vorgeschriebenen Register kein as hat, also z.B. mit "des" eine Tonklasse wiederholt. Ab da beginnt ein neuer Zyklus und es darf dann auch umittelbar "b, as" folgen, was nicht als zu vermeidende Wiederholung gilt.

Der *kompositorische Charakter* dieses Satzs ist aber grundlegend anders als in den Vorbildern: Gemeint ist offensichtlich eine "Klangflächen-Komposition", die sich allmählich aus einer kompakten Mitte in die Extreme des Tonraumes aufspaltet.

Zum zweiten Satz

Der sehr kurze zweite Satz demonstriert in "extrem didaktischer" Weise die wenigen in Feldmans Original "komponierten" Parameter: (D) Dauer, (R) Register und (I) Instrument.[3] Diese werden zunächst dreimal als einzelne vorgetragen, dann in den drei paarweisen Kombination, dann einmal zusammen.

Der Tonklassenvorrat ist maximal beschränkt, auf jeweils eine einzige, ändert sich aber mit den Formteilen. Als Konsequenz daraus böte die graphische Notation keinerlei "Mehr an Freiheit"; der Wechsel zur "ausnotierten konventionellen" Notation ist damit kein konzeptionller, sondern lediglich ein ergonomischer.

Es gibt auch die Möglichkeit, bei einer Erarbeitung des Werkes zwischen den beiden Partiturformen zu wechseln und die Spieler reflektieren und berichten zu lassen, wie sich Spielhaltung, Konzentration, Hören, etc. unterscheiden.

Zum dritten Satz

Das "Finale" ist der komplexeste Satz, gleichsam eine "Schlussfuge".

Er basierte auf der Erkenntnis, dass eine *Hakenfigur* wie die Abfolge "mittel-tief-hoch" = ✓ die einfachst mögliche Kontur ist, welche alle vier "dodekaphonen Modi" (Original, Inverses, Retrograd und Retrograd-Inverses = O, I, R, RI) darzustellen erlaubt.

Stehe der Ausdruck *(All-)Modireihe* für eine Abfolge von Ereignissen, die sich auffassen läßt als eine Abfolge aller vier dodekaphonen Modi einer viermal so kurzen Folge von Ereignissen. (Gegebenenfalls mittels Transposition.)

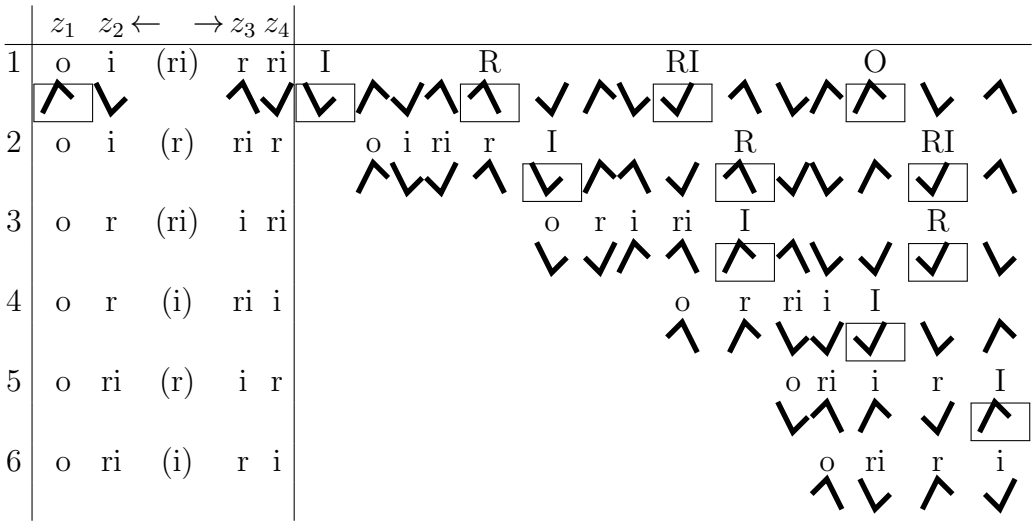
[2] Die digitale Implementierung macht das ebenso: Für diesen einen Namen der Eingabedatei ist der Algorithmus entsprechend modifiziert.

[3] Der eher unsystematisch behandelte Parameter "Spielweise" (arco/pizz/harmonics/sul ponticello) wird hier nicht berücksichtigt.

Ein wichtiges Werk der klassischen Avantgarde bezieht seine kompositorische Substanz aus einer solchen Modireihe: Anton Weberns Konzert op. 24, dessen Reihe aus der Aneinanderreihung aller vier Modi einer dreitönigen Keimzelle besteht. (o – ri – r – i)

Dabei ist sowohl für die Rezeptionsmechanismen als auch für den kompositorischen Umgang konstitutiv, dass die Anwendung der dedekaphonen Ableitungsoperationen auf eine solche Modireihe als Ganzer (= "äußere Anwendung") notwendigerweise eine Permutation der in ihr auftretenden Keimzellen-Modi zur Folge hat.

Tonhöhe: Setzen wir einen beliebigen Haken als das Original, so gibt es sechs verschiedene Permutationen von dessen Modi auf den folgenden Plätzen, also sechs "Typen von Allmodi-Reihen". Diese werden von den nach einander einsetzenden Stimmen des letzten Satzes im Parameter Registerauswahl (/Tonhöhe) so vorgetragen:



Grundsätzlich kann natürlich jeder der vier Haken  ,  ,  und  für das "o" ein gesetzt werden. Wenn z_1 durch die Operation p_2 in z_2 übergeht ($p_2 z_1 = z_2$ mit p_2 aus I, R oder IR), also die Elemente des linken Paares durch p_2 gegenseitig aus einander hervorgehen, dann gilt das auch für die Elemente des rechten Paares, also $z_3 = p_2 z_4$.[4]

Wichtigstes Verhältnis ist aber das von z_2 zu z_3, in obiger Tabelle in der Mitte in Klammern gedruckt. Dieses besteht, aus analogen Gründen, auch zwischen z_1 zu z_4 (siehe also Tabellenspalte z_4).

[4]Denn z_3 und z_4 gehen ja durch zwei *andere und ungleiche* Operationen aus z_1 hervor ($z_3 = p_3 z_1$, $z_4 = p_4 z_1$ und $p_3 \neq p_4$). Zwei der drei Operation I,R und IR verkettet aber ergeben immer die dritte, folglich gilt $p_3 = p_2 \circ p_4$, folglich $z_3 = p_3 z_1 = (p_2 p_4) z_1 = p_2(p_4 z_1) = p_2 z_4$.

Die drei Modusoperationen auf die Reihe als Ganze anzuwenden beeinflusst alle Zellen gleichermaßen, also ändert sich ihr gegenseitiges Verhältnis nicht, also auch nicht der Typ der Gesamtreihe ("1" bis "6"). Das gilt auch für die Operation P, den Rücklauf der Abfolge der vier Zellen, ohne sie intern zu verändern. Bezogen auf ihren Typ sind also alle Allmodireihen gegen alle Operationen symmetrisch = invariant.

Aber die Symmetrie der konkreten Wertefolgen (dargestellt als die Hakenfiguren) gegen die Reihenoperationen wird bestimmt durch z_4: Keine der Operationen I,R oder IR bildet eine einzelne Hakenfigur/Zelle auf sich selber ab. Wird aber die Operation R auf die Gesamtfolge angewendet, dann wird ja auch die Reihenfolge der Zellen ausgetauscht (= Operation P) zu z_4, z_3, z_2, z_1, und $z_1 = Rz_4$ und $z_1 = IRz_4$ sind ja durchaus erfüllbar. Wenn p_4 = R, dann ist also T=RT und IRT=IT, und die Gesamtreihe T hat nur zwei Modi, ist "modus-defekt". Ebenso mit p_4 = IR und T=IRT und RT=IT. Nur bei p_4 = I sind alle vier Modi von T verschieden. (Deshalb wohl auch von Webern gewählt, s.o.)

Dauer: Als Gegengewicht zur sehr bunten Hakenfolge der Tonhöhen wird die Dauer durch das nur halb so schnelle Ablaufen immer derselben Allmodi-Reihe (Typ 3) eher monoton bestimmt. Allemal aber hätten wir immer gleichbleibende Periodendauern pro Hakenablauf:

Dauer high	... mid	... low	Trennpause nach Haken	Summe
8	1+3+1	1	2	16
mit Auslöschungen:				
8		1		9
8	1+3+1			13
	1+3+1	1		6

Überdies sind das die sehr "orthogonalen" (wenn auch belebt nach Fibonacci unterteilten) vier mal vier Viertel. Um dies aufzubrechen werden bei fast allen aneinanderstoßenden Haken die beiden aneinanderstoßenden Ereignisse zu einem einzigen vereinigt, wenn sie in beidenParametern Register und Dauer übereinstimmen — sogennante "**Auslöschung**". Dadurch ergeben sich andere Längen, siehe Tabelle, und die Stimmen beginnen bald munter gegen einander zu schweben. (In der von der Software generierten Partitur sind die Haken an ihren jeweils ersten Ereignissen durchnummeriert. Daran ist das gut zu erkennen.)

Bei den Grenzen der Reihenabläufe (sie die Großbuchstaben in der Tabelle) finden der Deutlichkeit halber keine Auslöschungen statt.

Die ersten Haken und ersten Reihen für Tonhöhe und Dauer sind so ausgewählt, das in der Mitte der ersten Reihe eine Auslöschung im Mittelregister

stattfindet. Die Operation I aber läßt als einzige das "mid" Ereignis an derselben Stelle, kann also nicht zu einer solchen Auslöschung zwischen z_3 und z_4 führen. Wir müssen als erste Tonhöhenreihe also eine modus-defekte Reihe wählen.

Die erste einsetzende Stimme (names "hoch2") spielt für die Tonhöhen immer wieder die Reihe 1 ab. Dabei beginnt sie mit dem Haken ^. Für die weiteren Reihenabläufe wird immer mit einem anderen Haken begonnen, der durch die Anwendung auch dieser Reihe entsteht, siehe die Großbuchstaben in der Tabelle. Die Reihe wird gleichsam mit sich selbst multipliziert.[5]

Die weiteren Stimmen setzen in immer kürzerem Abstand ein, wie in der Tabelle eingetragen. Sie beginnen in Tonhöhe und Dauer "im Unisono" mit ihrer jüngsten Vorgängerstimme. Sie kopieren also beim Einsatz deren Haken. Auf den Tonhöhen-Haken wenden sie ab dann immer ihre eigene Reihe an: Einsatz Numer n bringt Reihe Nummer n. Nach Ablauf einer solchen Reihe (vier Haken) wenden sie ebenfalls die Reihe 1 auf ihren Anfangshaken an, für den Startpunkt des nächsten Ablaufes.

Wie gesagt wechseln die Dauern langsamer und immer nach derselben Reihe 3. Ihr Anfangshaken wird abr auch nach der Unisono-Regel bestimmt.

Als diese Architektur entschieden war, zeigte sich schnell, dass eine *programmierte Lösung* angemessen ist. Diese erfolgte dann in knappen 176 lines of code.

Für die Ausführung ist neben dem *leisen Grundcharakter* und dem Tempo (analog zu Feldmans Originalen) nicht weiter viel vorgeschrieben.

Für die Tonhöhenauswahl kann gewählt werden zwischen

- Alle Stimmen dieselbe gemeinsame Tonklasse. (Einton-Stück)
- Alle Stimmen immer dieselbe Tonklasse, aber jede eine andere.
- Alle Ausführende dieselbe gemeinsam ausgewählte Zwölftonreihe, immer "in Schleifenstellung" wiederholt.
- Jede Ausführende nur eine einzige Zwölftonreihe, ebenfalls in Schleifenstellung, unabhängig ausgewählt, vielleicht sogar spontan.
- Jede Ausführende nur eine ausgewählte Zwölftonreihe, aber abgesprochen und jede eine andere.
- Jeder Spieler wechselt zwischen ausgewählten Reihen, nach vorher abgesprochener Reihenfolge.

[5] Nur bei I ist die Anwendung auf die Startzelle z_1 gleichbedeutend mit der Anwendung auf die ganze Reihe. Bei R und RI müsste dazu noch die Operation P = Rücklauf der Zellen von z_4 bis z_1 angewandt werden, was hier aber nicht gemeint ist.

- Jeder Spieler kann spontan zwischen ausgewählten Reihen wechseln, immer nach vollständigem Ablauf einer Reihe.
- Jeder Spieler kann spontan zwischen ausgewählten Reihen wechseln, dabei Reihen auch vorzeitig abbrechen.

Die vorgeschlagenen Reihen sind auf Seite 24 als Tabelle abgedruckt. Es sind aber auch beliebig andere erlaubt. Unabhängig von den aufgeuzählten Varianten können für alle oder für einzelne Spielerinnen zusätzlich beliebige Transpositionen und/oder die I-, R- und IR-Transformationen erlaubt werden.

Die Tonhöhengestaltung und damit ein großer Einfluss auf den Charakter des Satzes ist also in weitem Rahmen variabel. Anzustreben ist dabei aber auch, die grundlegenden Einsatzregeln und Reihenabläufe der Hakefiguren deutlich erfahrbar zu machen.

Die momentan existierende Programmierung geht nur bis zur Generierung der Feldman-Partitur und unterstützt (noch?) nicht diese ganze Vielfalt der Tonhöheninterpretation. Darum bleibt bis auf weiteres deren Erforschung einem musizierenden Ensemble anvertraut.

Ungefähre Dauern

Satz 1 — 4 Minuten
Satz 2 — 2 1/2 Minuten
Satz 3 — 4 Minuten

Comprehension

These pieces are inspired by "Projections 1 to 5" (1950-51) by Morton Feldman.

In the notation time runs conventionally from left to right; each measure has four quarters; all events are aligned to quarters. The staves of the instruments/voices are conventionally stacked vertically. Each box means to play one pitch for the indicated duration, in a low, middle, hor high register. Each player must define the registers for their instrument.

Additionally to Feldman we require that there is at least a forth wide gap between the registers.

All movements are at about 72 BPM and in low dynamics.

The pitches of the first movement are restricted to d-flat, e-flat, g-flat, a-flat, and b-flat. The selection of the pitch to play shall be improvised by the players. Re-use of a pitch class in the resulting sound must be delayed as far as possible—each player has to listen to all sounding pitch classes. If a

re-use is unavoidable, a new cycle is started and pitch class counting starts anew.

The second movement allows one pitch class only, which changes partwise. Since there is hardly any freedom, we provide a transcription into normal notation.

The last movement deals with sequences of patterns ("hooks") which are transformed in the spirit of classical dodecaphonic composition. The pitches of the events shall be provided by dodecaphonic series, from which the players can select, see the table on page 24, either bespoken and planned among all, or spontaneously.

Literatur

[1] Brett Boutwell. Morton feldman's graphic notation: Projections and trajectories. *Journal of the Society for American Music*, 6(4):457–82, 2012.

[2] David Cline. *The Graph Music of Morton Feldman*. Cambridge University Press, Cambridge, UK, 2016.

[3] Morton Feldman. *Give My Regards to Eigth Street — Collected Writings*. Exact Change, Cambridge, MA, 2000.

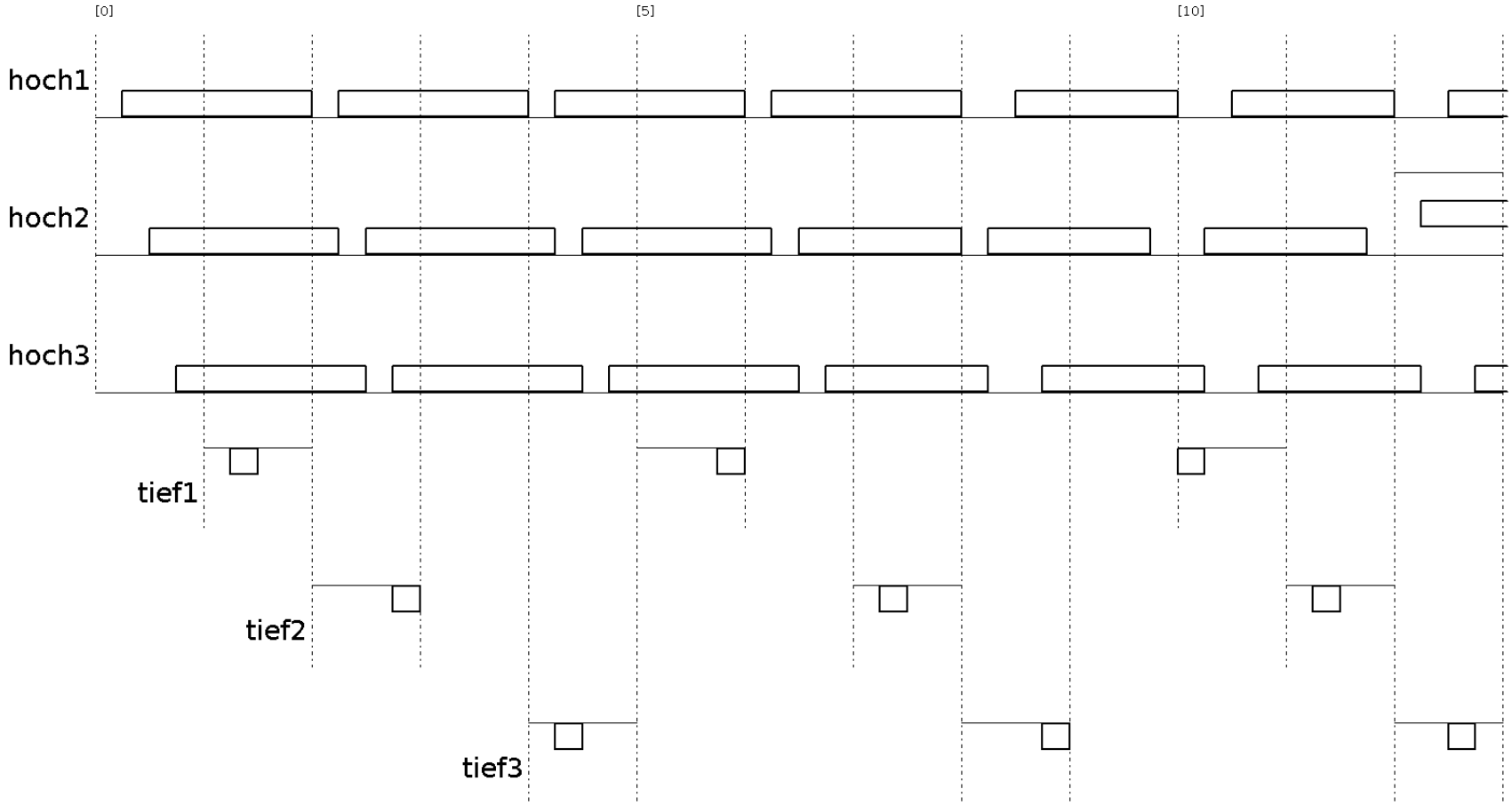
[4] Markus Lepper and Baltasar Trancón y Widemann. *Example Instances of the TScore Projekt Infrastructure*, 2013. `http://markuslepper.eu/sempart/tscoreInstances.html`[20230920].

[5] Markus Lepper and Baltasar Trancón y Widemann. tscore: Makes computers and humans talk about time. In Joaquim Filipe and Jan Dietz, editors, *Proc. KEOD 2013, 5th Intl. Conf. on Knowledge Engineering and Ontology Development*, pages 176–183, Portugal, 2013. instincc, scitePress. `http://markuslepper.eu/papers/tscore2013.pdf`[20231003].

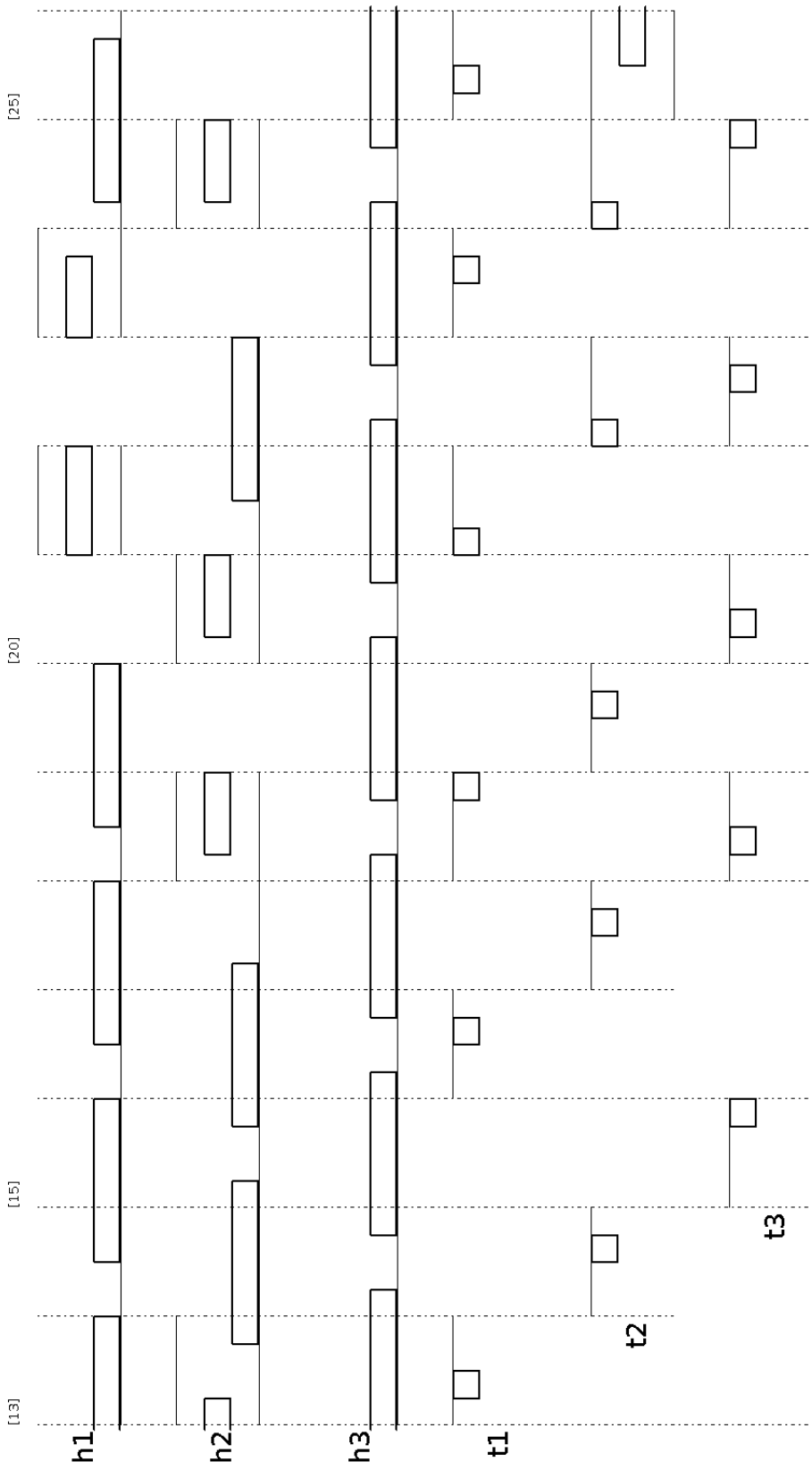
[6] Markus Lepper and Baltasar Trancón y Widemann. Translets —– parsing diagnosis in small dsls, with permutation combinator and epsilon productions. In *Tagungsband des 35ten Jahrestreffens der GI-Fachgruppe "Programmiersprachen und Rechenkonzepte*, volume 482 of *IFI Reports*, pages 114–129. University of Oslo, 2018. `http://urn.nb.no/URN:NBN:no-65294`[20230920].

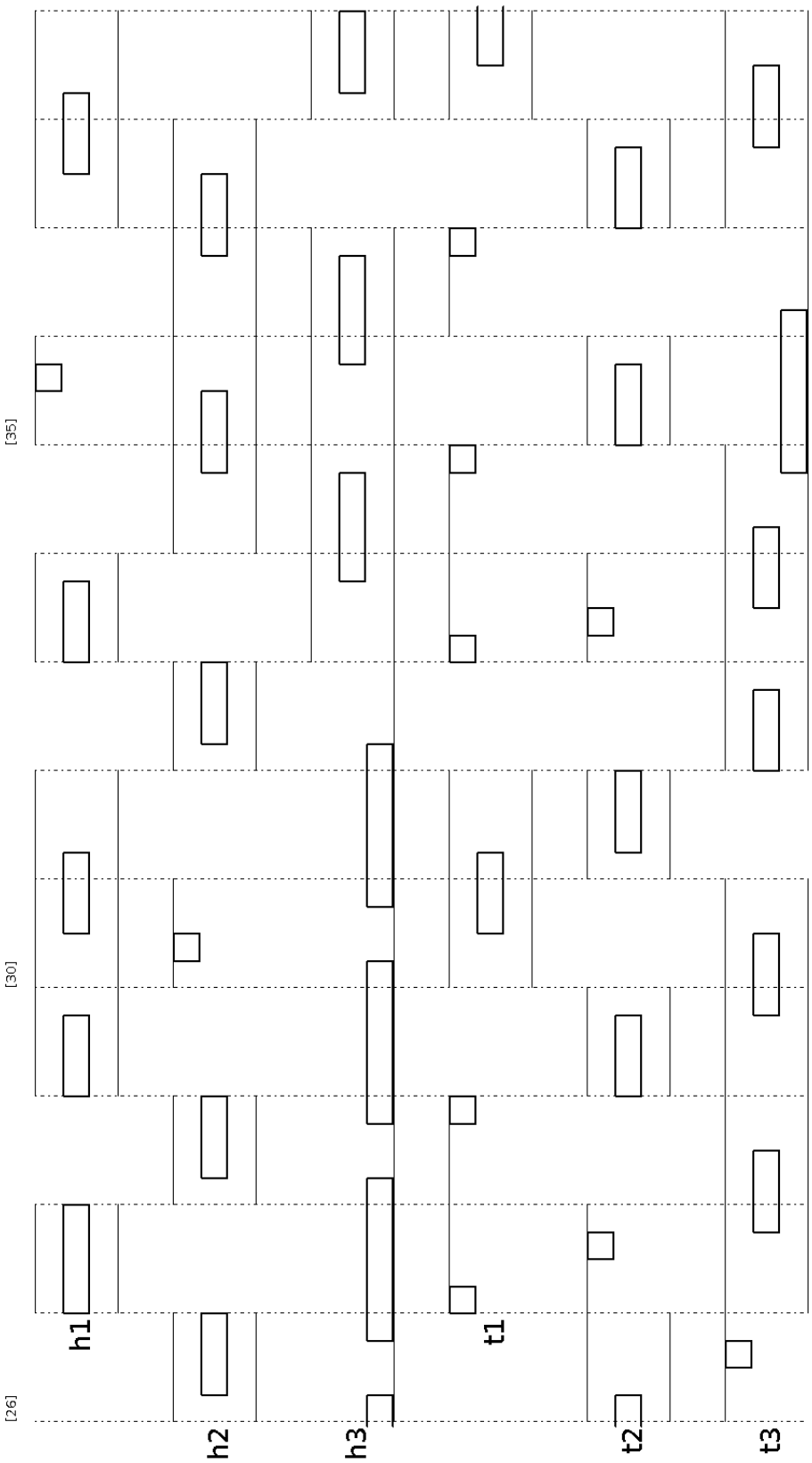
Drei kleine Studien nach Morton Feldman — Satz 1

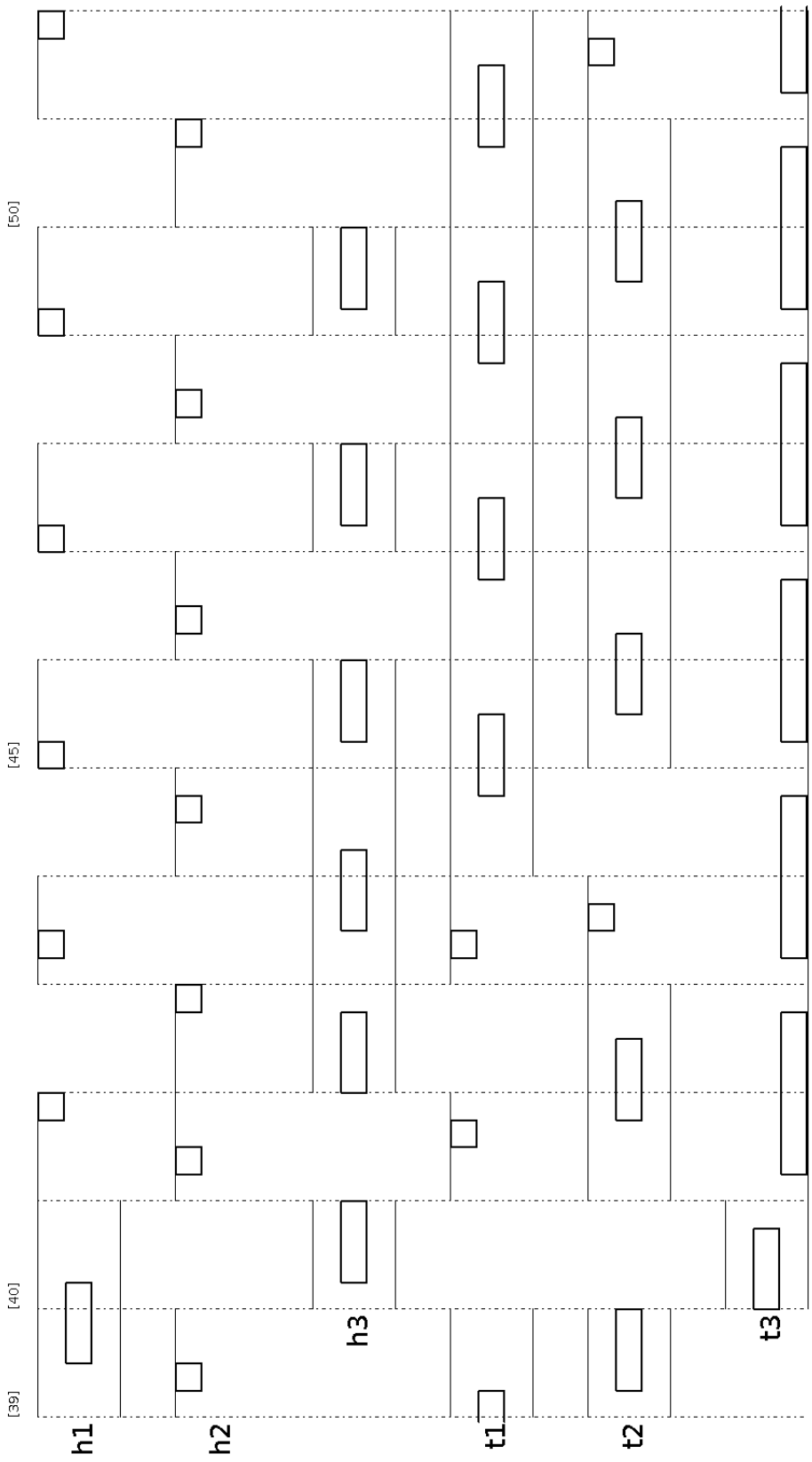
Sempre piano. MM 60-72. Tonhöhen **improvisiert** aus {des, es, ges, as, b}. Möglichst lange wiederholungsfreie Bereiche je hohe/tiefe Stimmen.

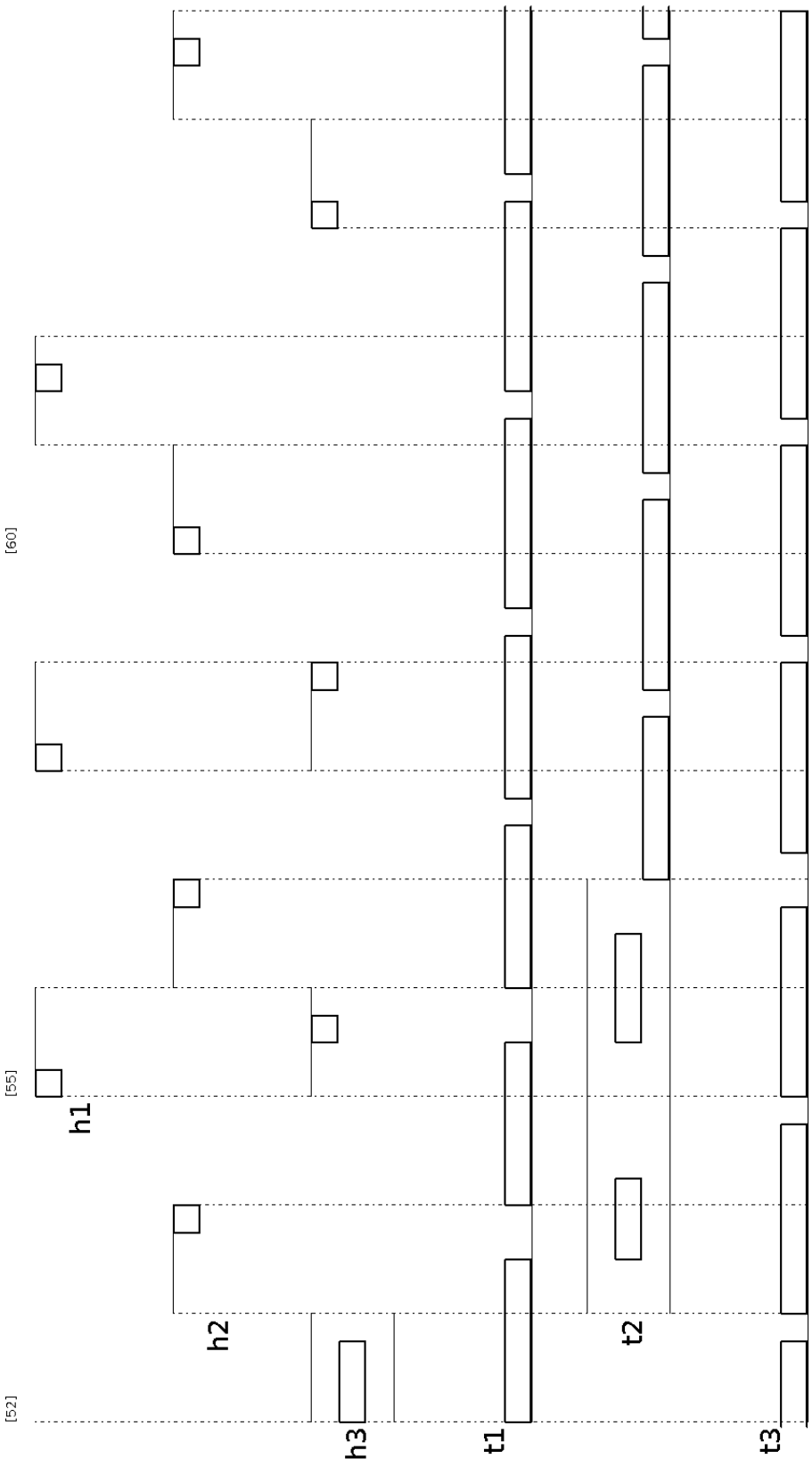


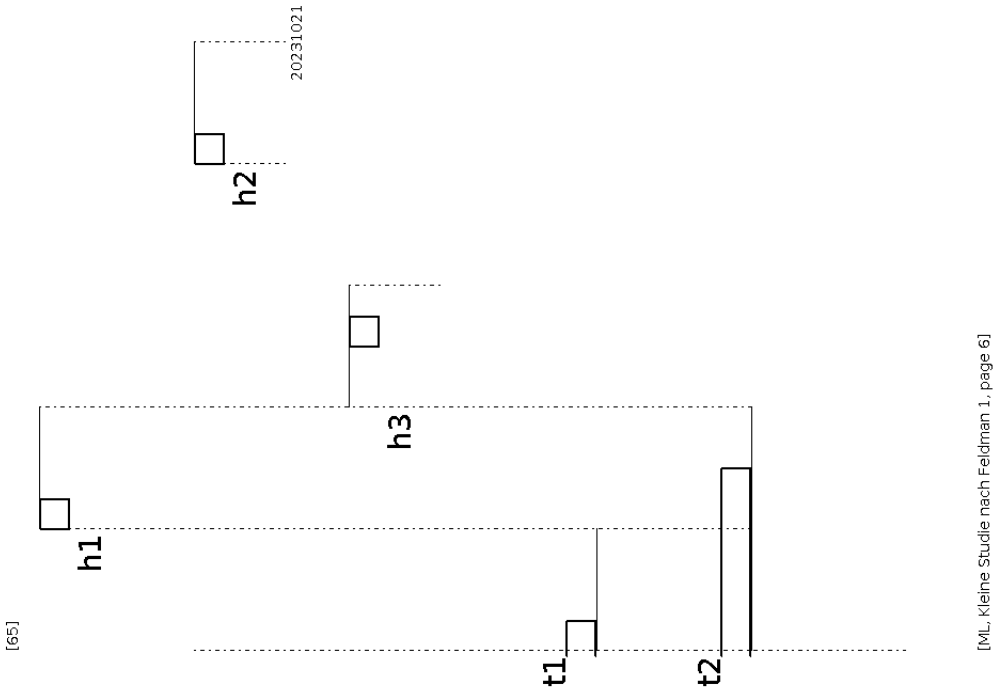
[ML, Kleine Studie nach Feldman 1, page 1]

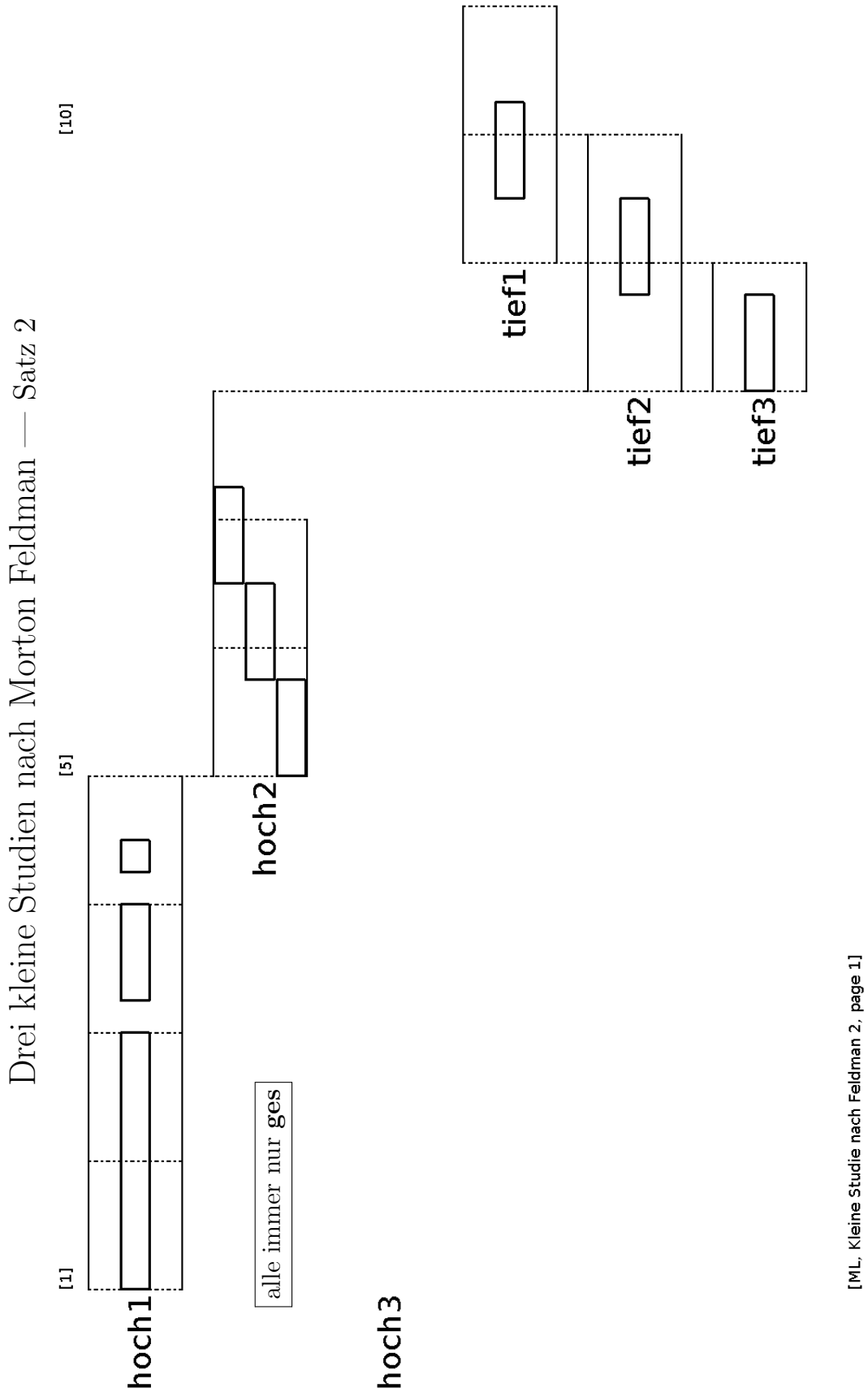

[25]
[20]
[15]
[13]
h1
h2
h3
t1
t2
t3
[ML, kleine Studie nach Feldman 1, page 2]

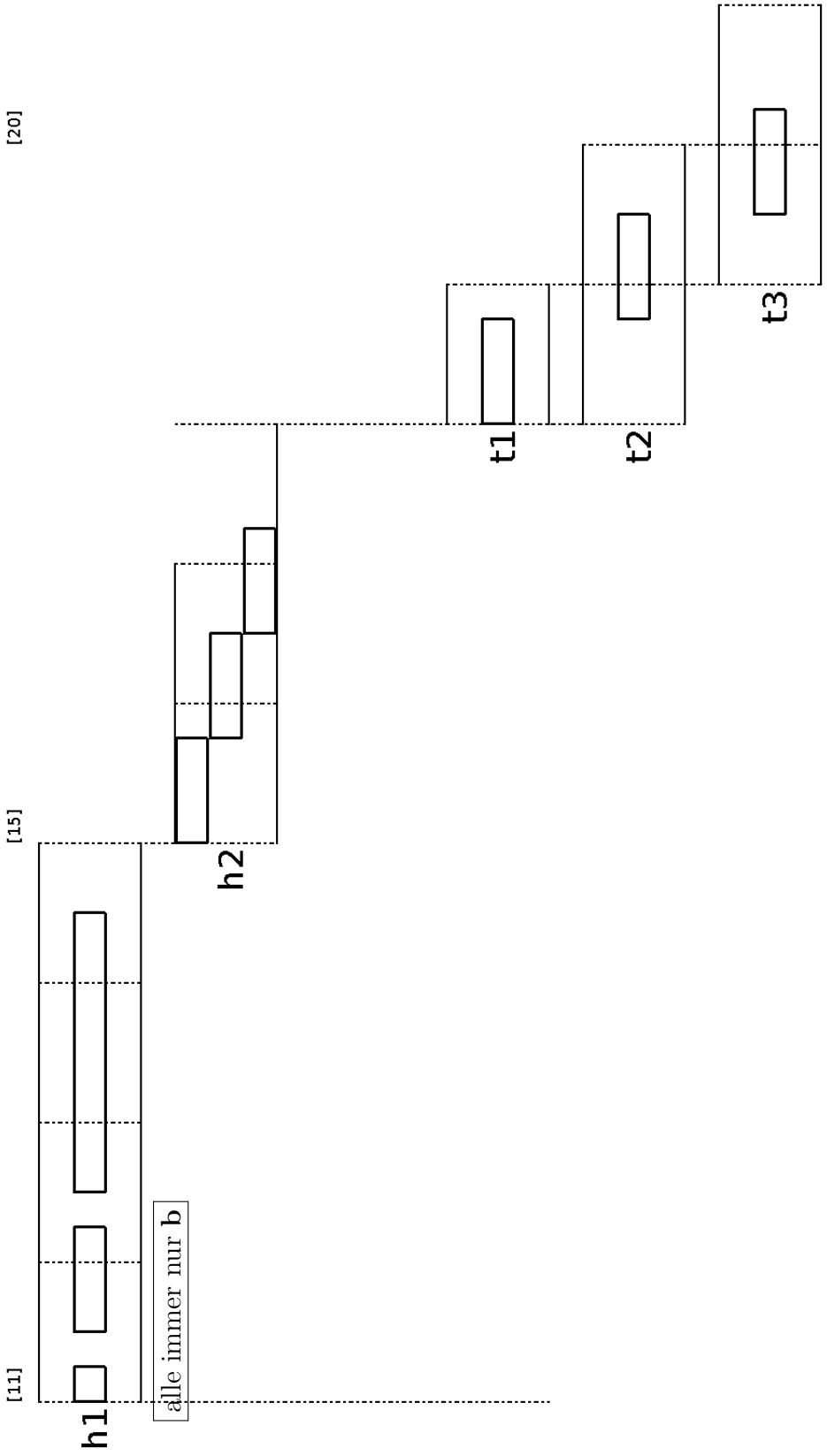

[35]
[30]
[26]
h1
h2
h3
t1
t2
t3
[ML, kleine Studie nach Feldman 1, page 3]

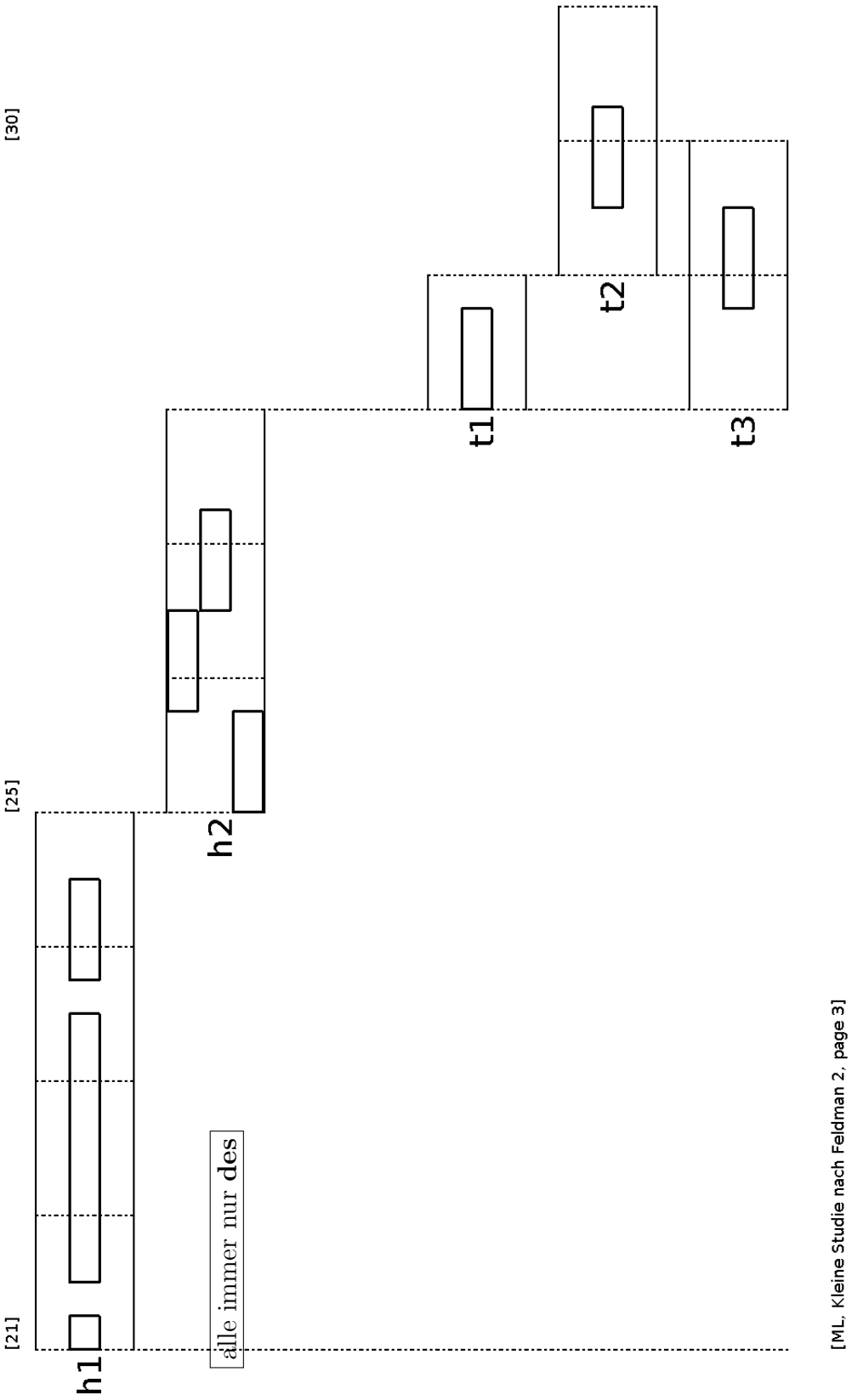

[50]
[45]
[40]
[39]
h1
h2
h3
t1
t2
t3
[ML, kleine Studie nach Feldman 1, page 4]

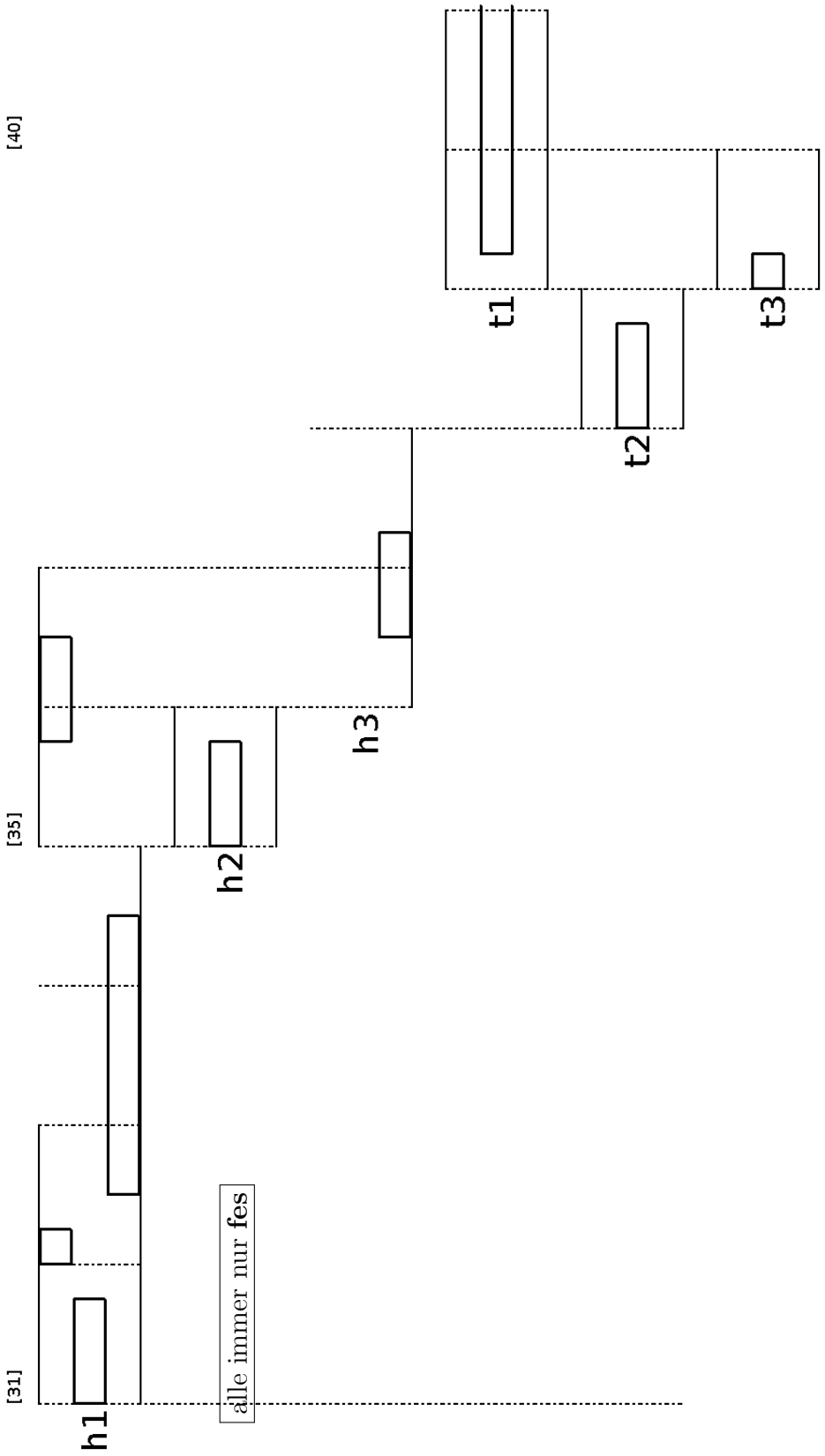

[60]
[55]
[52]
h1
h2
h3
t1
t2
t3
[ML, kleine Studie nach Feldman 1, page 5]

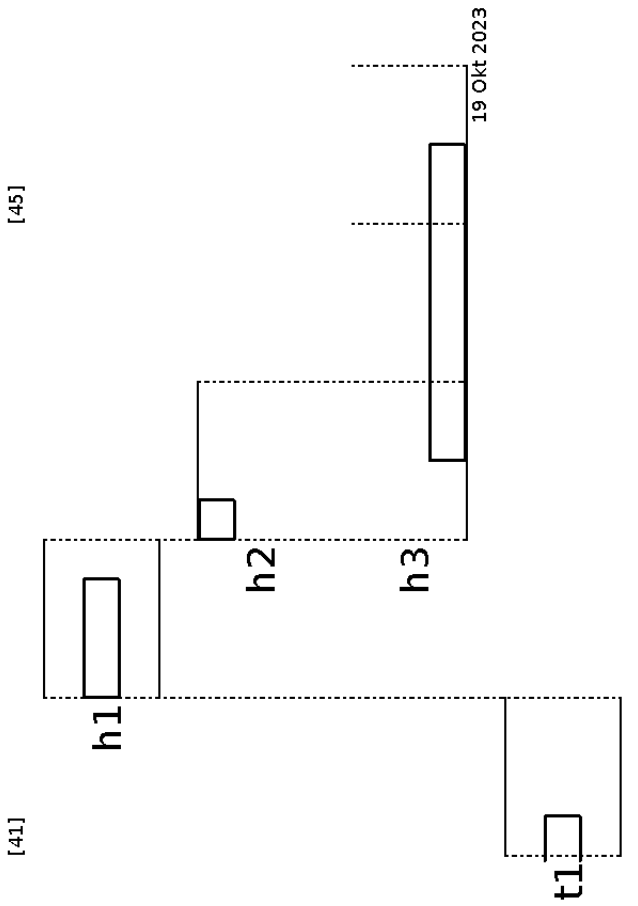

20231021
h2
h3
h1
t1
t2
[65]
[ML, kleine Studie nach Feldman 1, page 6]


Drei kleine Studien nach Morton Feldman — Satz 2
[1]
[5]
[10]
hoch1
hoch2
hoch3
alle immer nur ges
tief1
tief2
tief3
[ML, Kleine Studie nach Feldman 2, page 1]


[20]
[15]
[11]
h1
alle immer nur b
h2
t1
t2
t3
[ML, Kleine Studie nach Feldman 2, page 2]


[30]
[25]
[21]
h1
h2
t1
t2
t3
alle immer nur des
[ML, Kleine Studie nach Feldman 2, page 3]


[40]
[35]
[31]
h1
h2
h3
t1
t2
t3
alle immer nur fes
[ML, Kleine Studie nach Feldman 2, page 4]



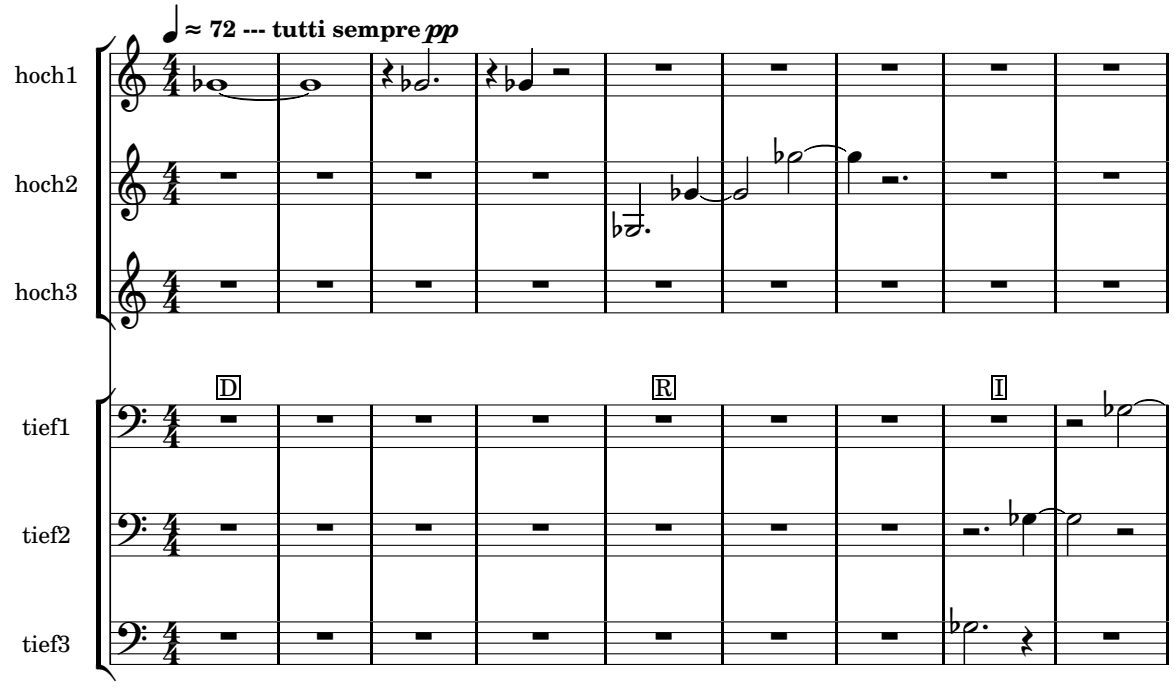
alle immer nur **as**

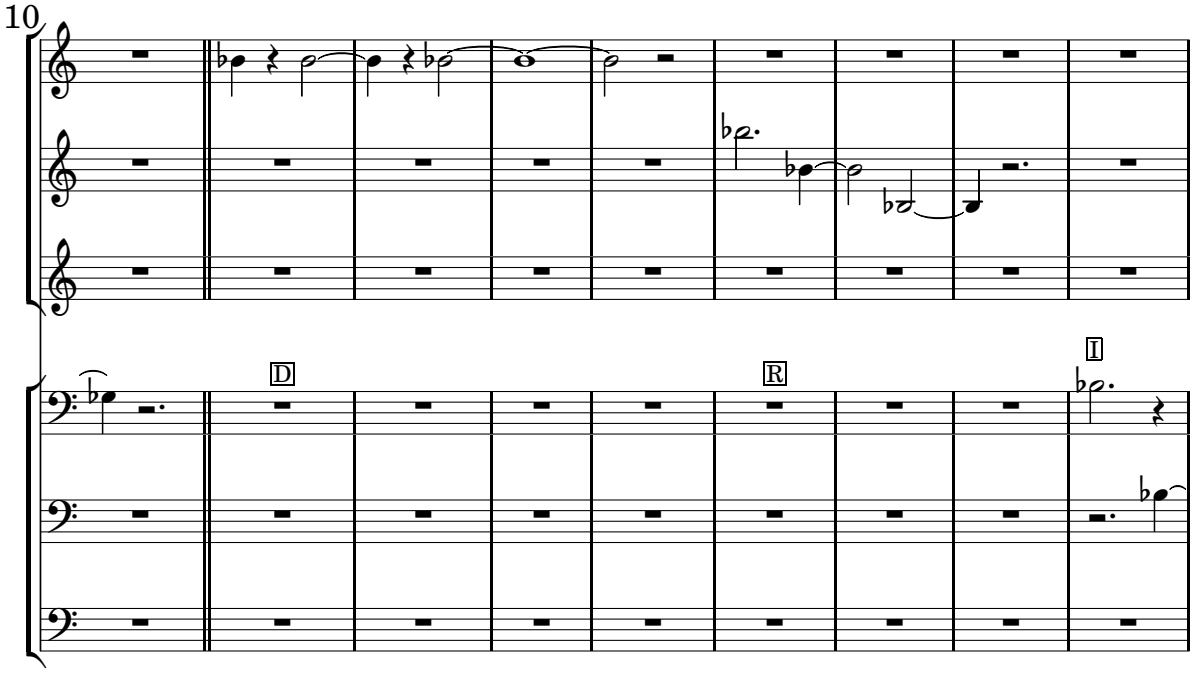
[ML, Kleine Studie nach Feldman 2, page 5]

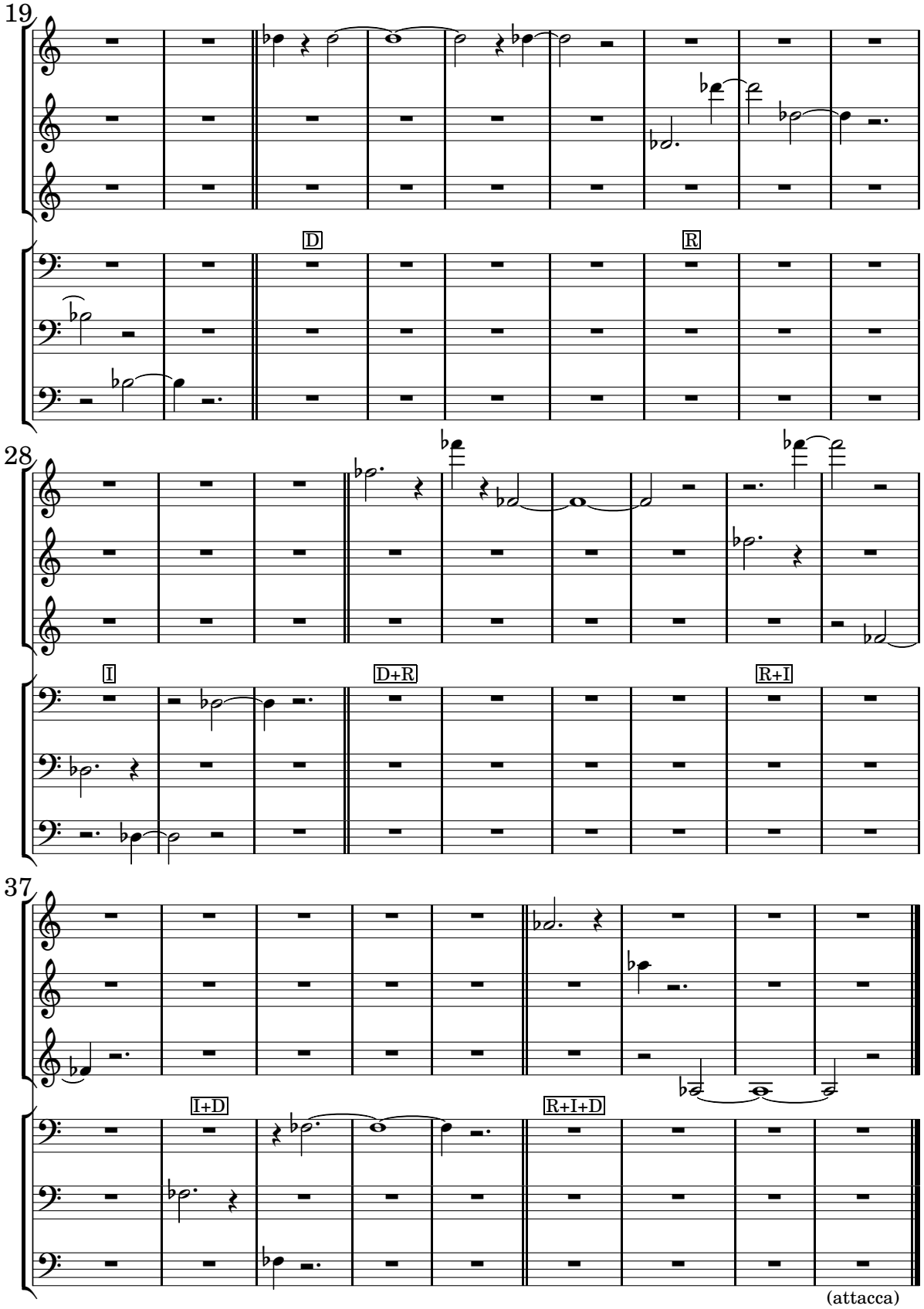
Drei kleine Studien nach Morton Feldman

(Satz 2, ausnotiert)

ML 20231019
op.el.20/2

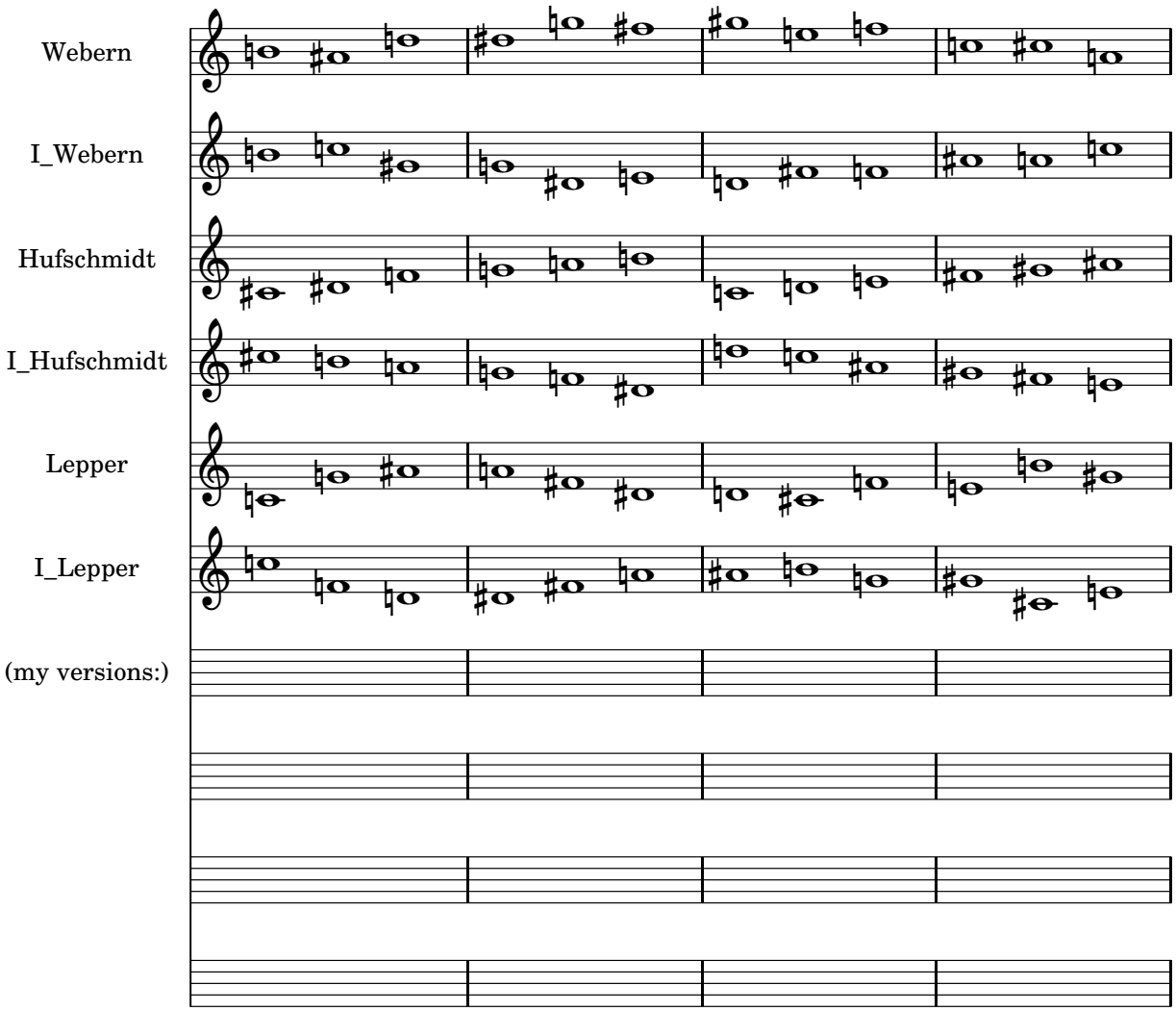




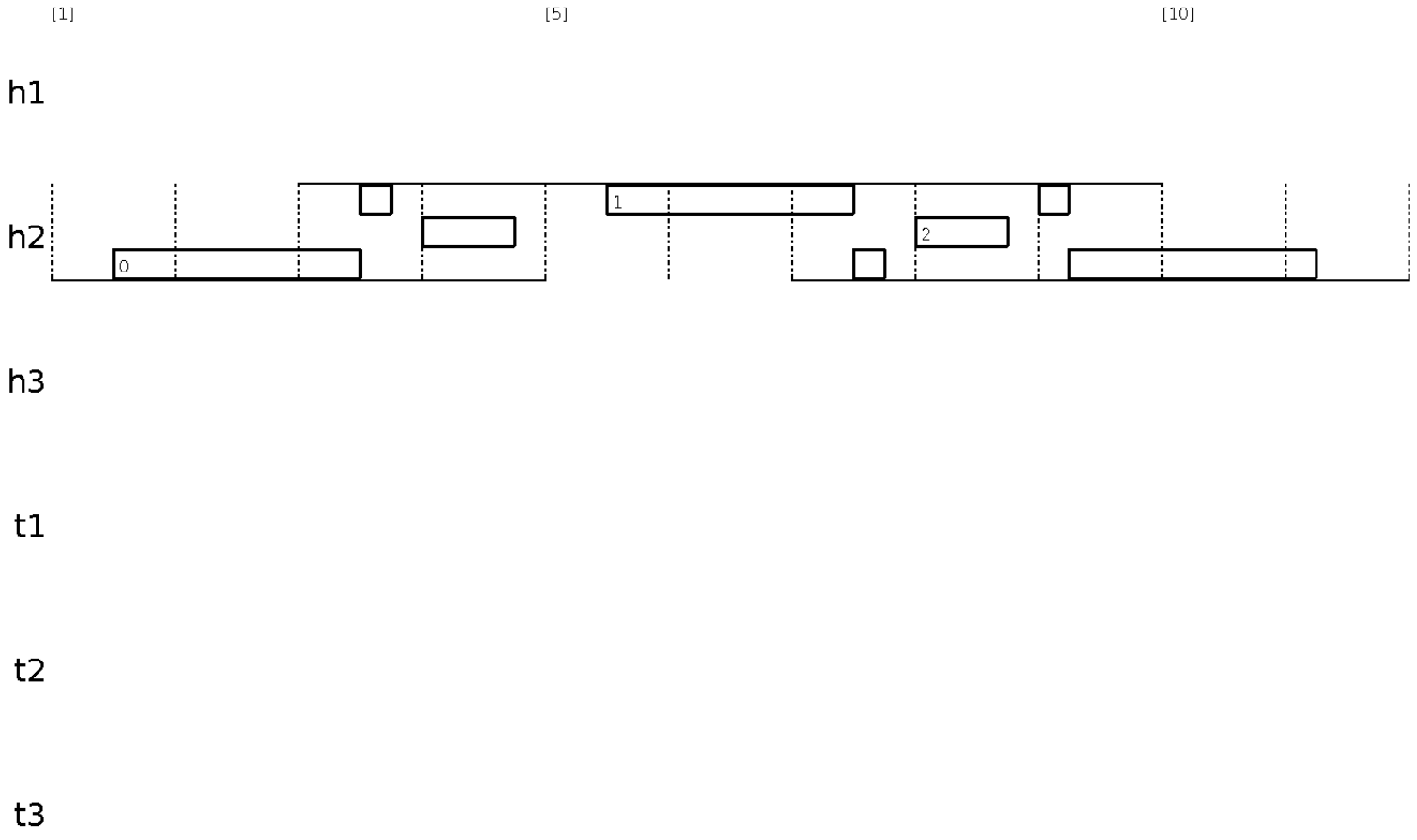


Music engraving by LilyPond 2.16.0—www.lilypond.org

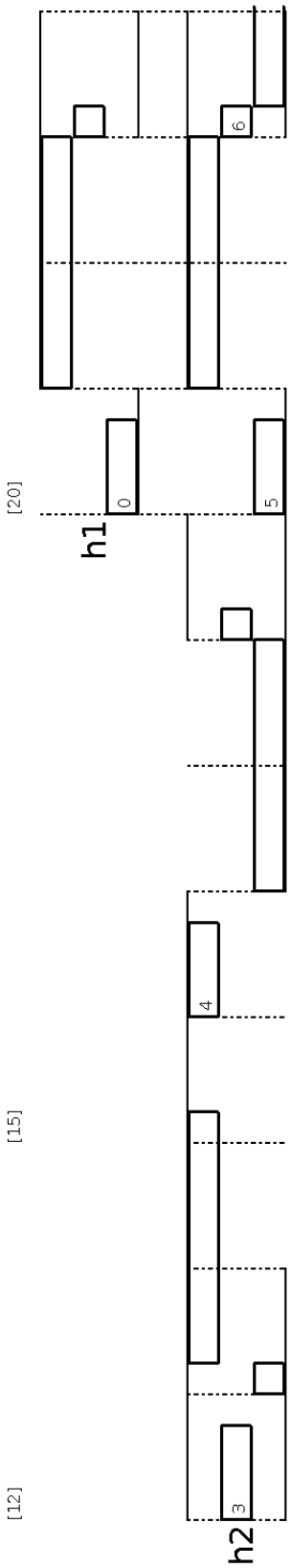
Satz 3 --- Reihen zur Auswahl:



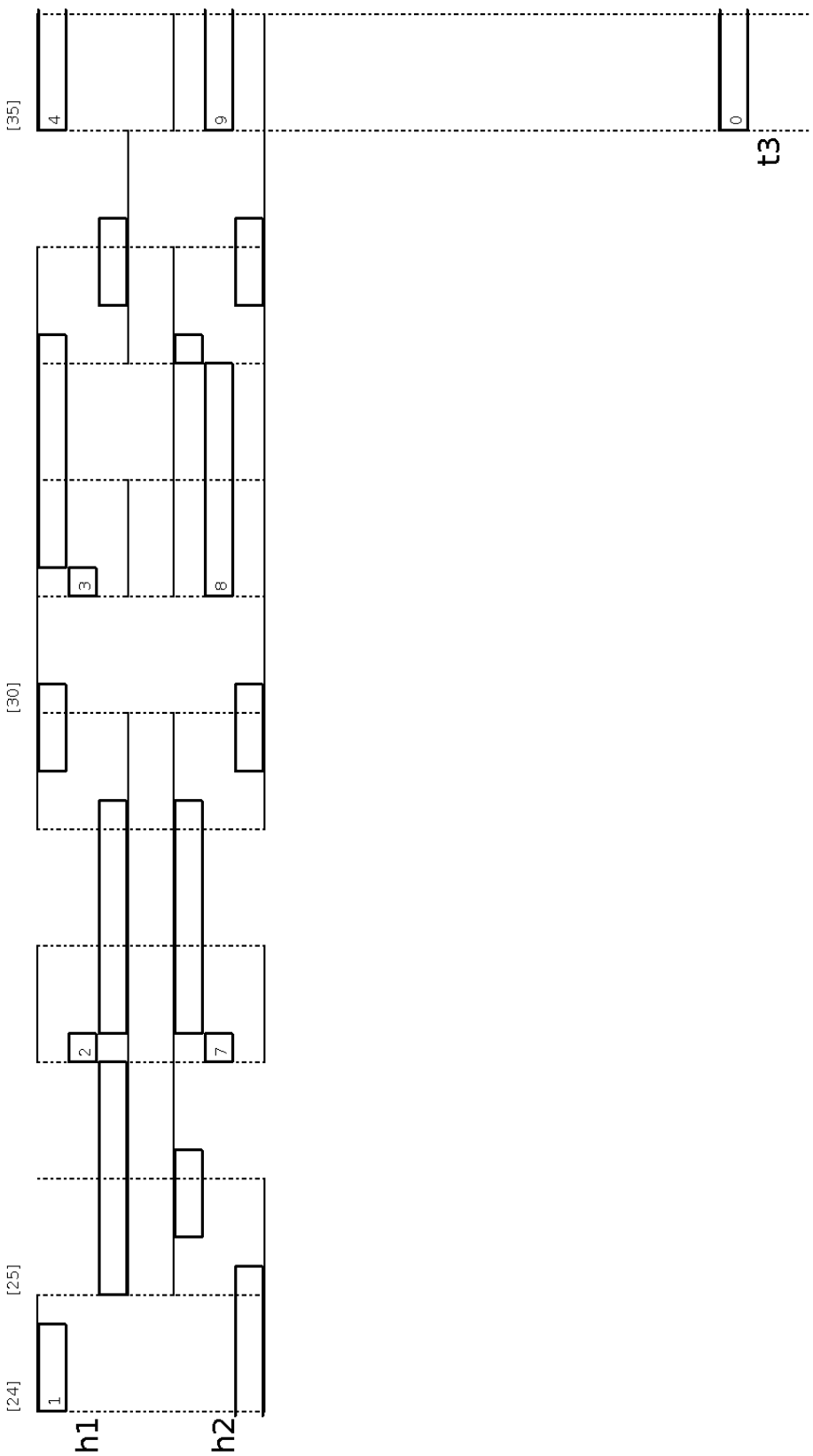
Drei kleine Studien nach Morton Feldman — Satz 3



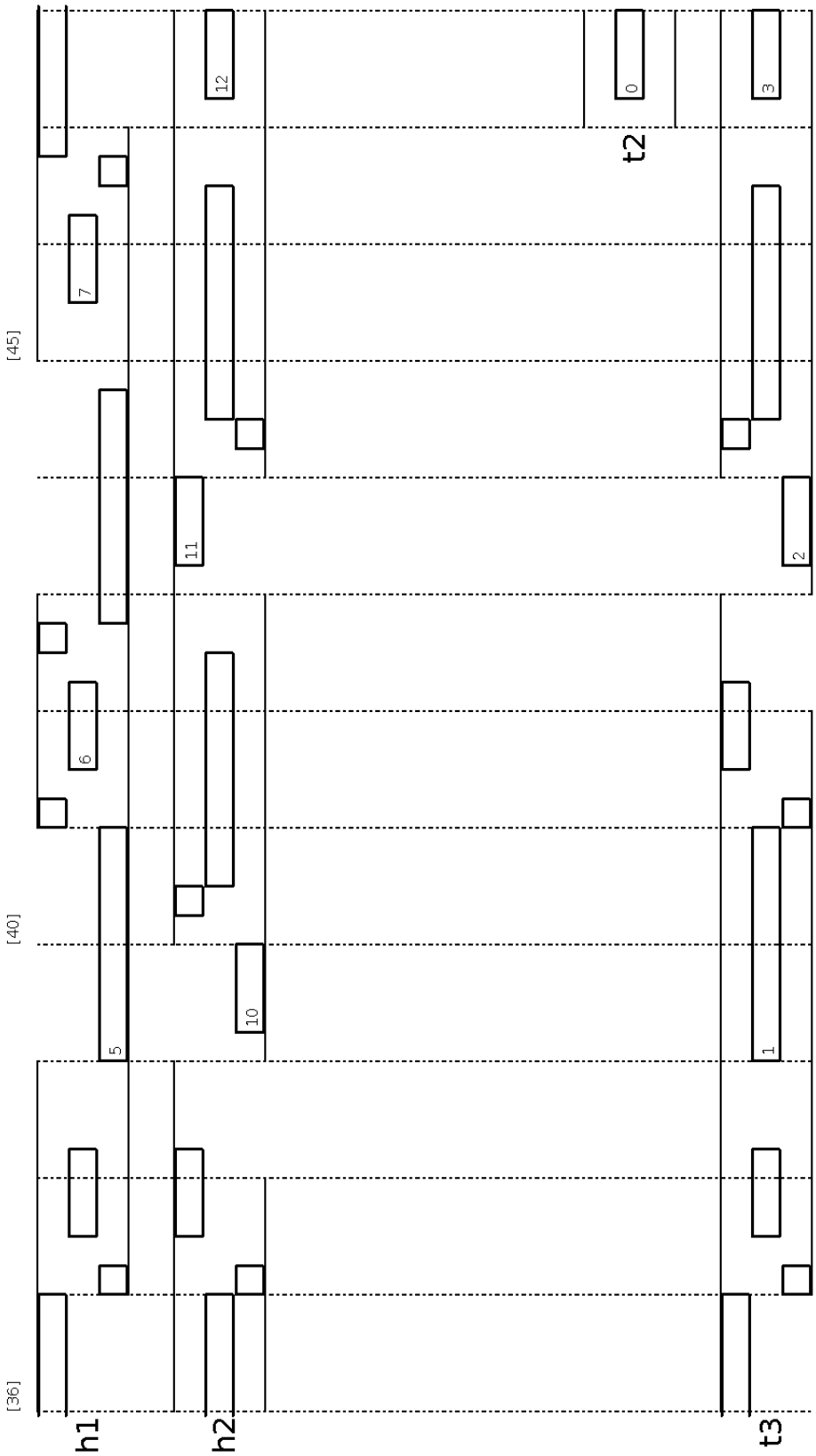
[ML, Kleine Studie nach Feldman 3, page 1]



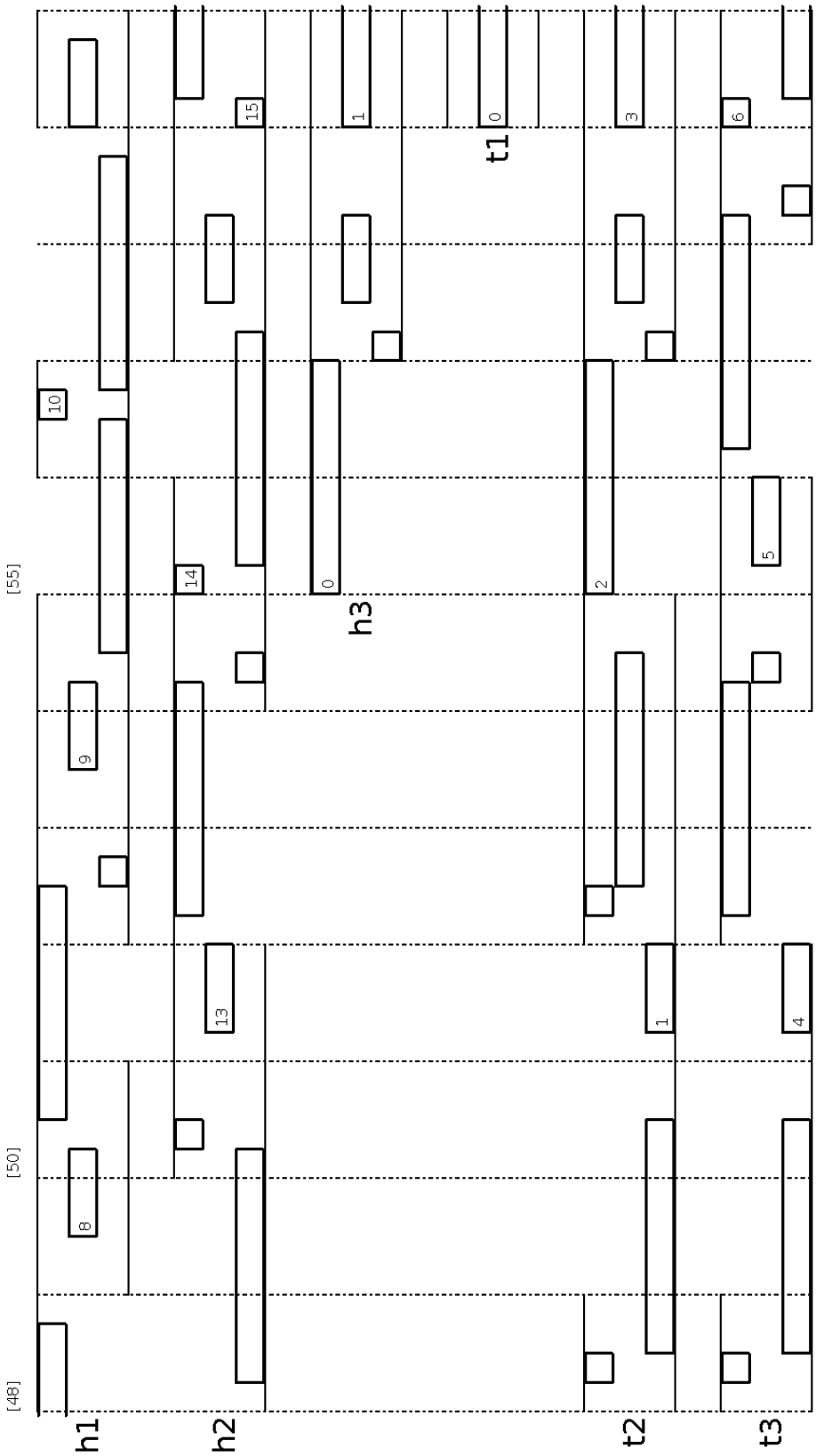
[ML, Kleine Studie nach Feldman 3, page 2]



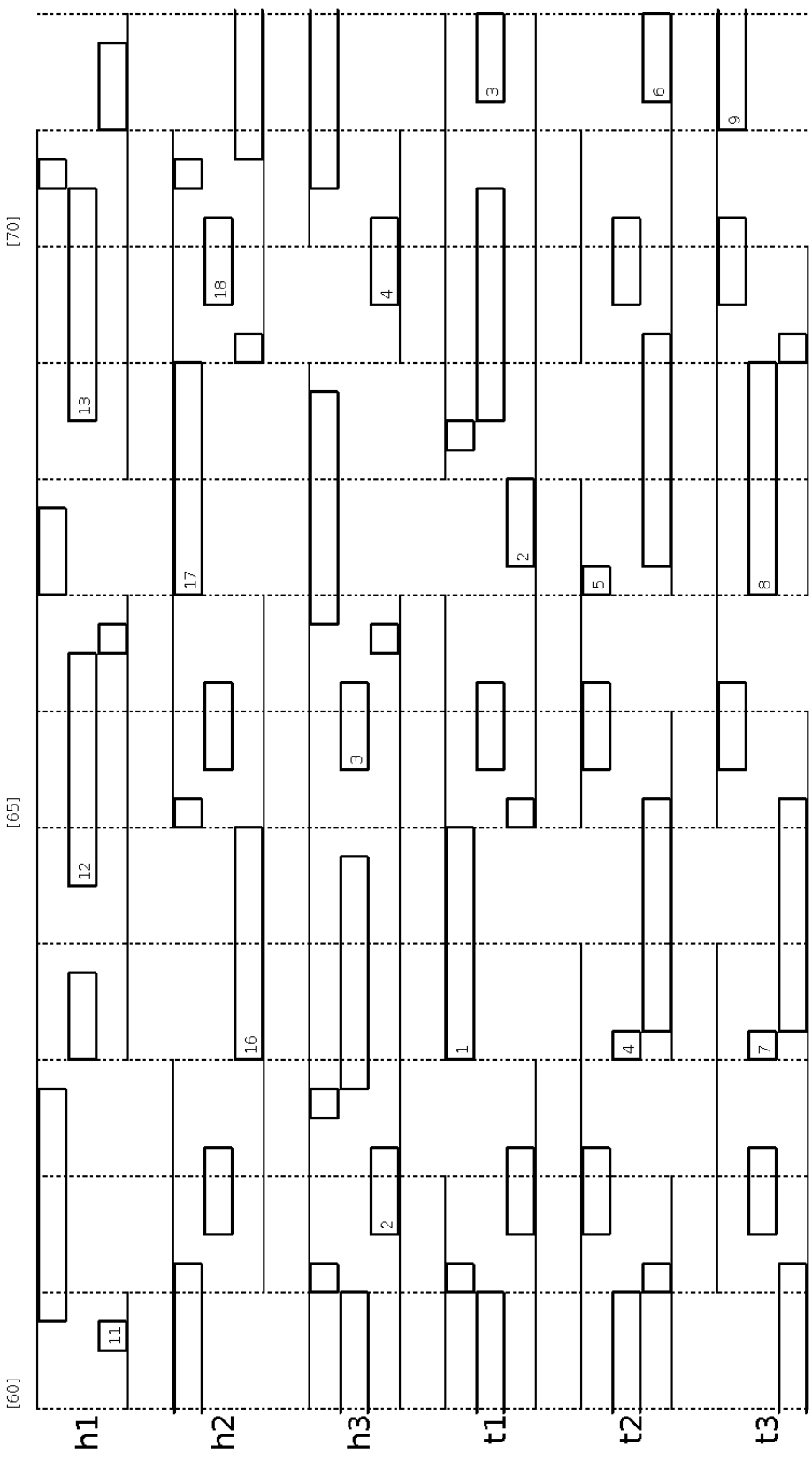
[ML, Kleine Studie nach Feldman 3, page 3]



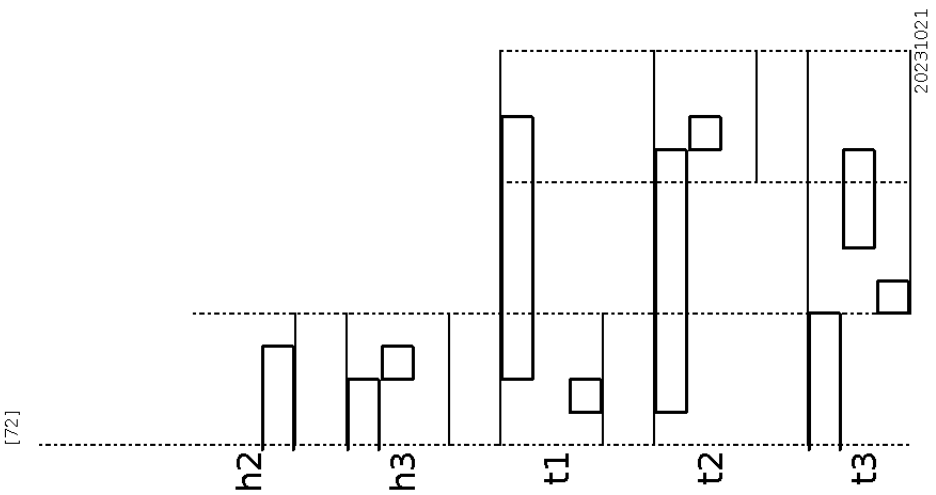
[ML, Kleine Studie nach Feldman 3, page 4]



[ML, Kleine Studie nach Feldman 3, page 5]



[ML, Kleine Studie nach Feldman 3, page 6]


[72]
h2
h3
t1
t2
t3
20231021
[ML, Kleine Studie nach Feldman 3, page 7]

Quelltexte

```
1: // ML_studie_1_meta
2:
3: PARS sola
4: VOX h1                nomen.longum = hoch1    nomen.breve = h1
5: VOX h2                nomen.longum = hoch2    nomen.breve = h2
6: VOX h3                nomen.longum = hoch3    nomen.breve = h3
7: VOX t1                nomen.longum = tief1    nomen.breve = t1
8: VOX t2                nomen.longum = tief2    nomen.breve = t2
9: VOX t3                nomen.longum = tief3    nomen.breve = t3
10:
11: T       0    10      20          30      40      50      60      65
12: VOX h1  0/0  10/30                  /50     50/     80/     100/100 100/100
13: VOX h2  0/0  10/40                          50/70           100/100 100/100
14: VOX h3  0/0  10/20   /30            20/             40/60   100/100 100/100
15:
16: VOX t1  100/100             50/90           50/60           50/100          0/0     0/0
17: VOX t2  100/100             60/90           30/60                   40/60   0/0     0/0
18: VOX t3  100/100             60/80           30/60           20/40           0/0     0/0
19:
20: // eof
```

```
// ML Studie nach Feldman 2
CONFORM eu.bandm.music.applications.feldproj.FeldmanProjection_n

PARS sola
titulum = "Kleine Studie nach Feldman 2"
auctor = "ML"
visuum.tactiInPagina = 10
visuum.interVoces = 1
signumFinis  = "19 Okt 2023"

VOX h1                  nomen.longum = hoch1    nomen.breve = h1
VOX h2                  nomen.longum = hoch2    nomen.breve = h2
VOX h3                  nomen.longum = hoch3    nomen.breve = h3
VOX t1                  nomen.longum = tief1    nomen.breve = t1
VOX t2                  nomen.longum = tief2    nomen.breve = t2
VOX t3                  nomen.longum = tief3    nomen.breve = t3


T         1            2         3           4           5         6        7        8          9       10        11
VOX h1    m            -         % .m        % m  %  %
VOX h2                                                   l.     m  -    h   - .%     %

VOX t1                                                                                          %    m  -  .%
VOX t2                                                                               %.      m  -    %
VOX t3                                                                               m.      %

//VOX th    ges


T         11           12        13          14          15        16       17       18         19      20        21
VOX h1    m  %  m -    - %  m -  -           -     %     %
VOX h2                                                   h.     m  -    l   - .%     %

VOX t1                                                                               m.      %
VOX t2                                                                               %.      m  -    %
VOX t3                                                                                          %    m  -  .%

//VOX th    b

```

```
T         21           22        23          24          25        26       27       28         29      30        31
VOX h1    m  %  m -    -         -  -   %  m -     %
VOX h2                                                   l.     h  -    m   - .%     %

VOX t1                                                                                     m.      %
VOX t2                                                                                                %    m  -  .%
VOX t3                                                                                     %.      m  -    %

//VOX th    des

//        D+R                                            R+I                         I+D
T         31           32        33          34          35        36       37       38        39       40     41      42
VOX h1    m.       %   h % l  - -           -     %     %.     h  -    %
VOX h2                                                   m.     %
VOX h3                                                             %    l   - .%

VOX t1                                                                                                 %  .m    -     -  .%
VOX t2                                                                                     m.      %
VOX t3                                                                                                 m .%

//VOX th    fes

//        D+R
T         42           43        44          45       46
VOX h1    m.       %
VOX h2                 h .%
VOX h3                 %     l   -           -    %

VOX t1
VOX t2
VOX t3

//VOX th    as

// eof
```

```
 1: package eu.bandm.music.applications.feldproj ;
 2:
 3: import java.util.List ;
 4: import java.util.ArrayList ;
 5: import java.util.Map ;
 6: import java.util.TreeMap ;
 7: import java.util.Set ;
 8:
 9: import eu.bandm.tools.location.Location;
10: import static eu.bandm.tools.location.Location.noLocation ;
11:
12: import eu.bandm.tools.message.MessageReceiver ;
13: import eu.bandm.tools.message.SimpleMessage ;
14:
15: import eu.bandm.tools.util.xml.XMLDocumentIdentifier ;
16:
17: import eu.bandm.tools.ops.Rational ;
18:
19: import static eu.bandm.tools.util.java.Collections.literalSet ;
20:
21: import eu.bandm.tscore.base.Timelesses ;
22:
23: // FIXME RAUS
24: import eu.bandm.tscore.model.TimeScape ;
25:
26: import eu.bandm.tscore.model.Tp ;
27: import eu.bandm.tscore.model.TpTop ;
28: import eu.bandm.tscore.model.Part ;
29: import eu.bandm.tscore.model.Vox ;
30: import eu.bandm.tscore.model.Event ;
31: import eu.bandm.tscore.model.Textvalue ;
32:
33: import eu.bandm.music.entities.SeriesTransformation ;
34: import static eu.bandm.music.entities.SeriesTransformation.I ;
35: import static eu.bandm.music.entities.SeriesTransformation.R ;
36: import static eu.bandm.music.entities.SeriesTransformation.IR ;
37: import static eu.bandm.music.entities.SeriesTransformation.O ;
38:
39: import eu.bandm.music.haken.Haken ;
40: import static eu.bandm.music.haken.Haken.OM ;
41: import static eu.bandm.music.haken.Haken.UM ;
42: import static eu.bandm.music.haken.Haken.MO ;
43: import static eu.bandm.music.haken.Haken.MU ;
44:
45: import eu.bandm.music.haken.RelWert ;
46: import static eu.bandm.music.haken.RelWert.M ;
47: import static eu.bandm.music.haken.RelWert.U ;
48:
49: /** Generate a FeldmanProjection_n score where dodecaphonic-like serieses
50:  *  are placed by multiplying them with themselves.
51:  */
52: public class Generate_3 {
53:
54:   /** The nameof the tscore part to generate*/
55:   public static final String partName = "generated_3" ;
56:
57:   /** Evident*/
58:   final MessageReceiver<SimpleMessage<XMLDocumentIdentifier>> msgr ;
59:
60:   /** Only constructor, does nothing*/
61:   public Generate_3 (final MessageReceiver<SimpleMessage<XMLDocumentIdentifier>> msgr){
62:     this.msgr = msgr ;
63:   }
64:
65:   /** The serieses of dodecaphonic modes used for the voices in this order.*/
66:   final SeriesTransformation[][] reihen = {{O,  I,  R, IR},
67:                                            {O,  I, IR,  R},
68:                                            {O,  R,  I, IR},
69:                                            {O,  R, IR,  I},
70:                                            {O, IR,  I,  R},
71:                                            {O, IR,  R,  I}};
72:
73:   /** The generators for the voices, in the order of their acoustic appearance.*/
74:   final List<Agent> agents = new ArrayList<>();
75:
76:   /** Permutation fo the voice names: The agents start sounding in their numerical order,
```

```
77:     *  but the generated voices do not.
78:     */
79:    final String[] voiceNames = new String[]{"h2", "h1", "t3", "t2", "h3", "t1"};
80:
81:
82:    /** Needed only to make the write-out to the FeldmanProjection_n score in the
83:     *  alphabetic order of voice names.
84:     */
85:    final Map<String, Agent> agentsByName = new TreeMap<>();
86:
87:    /** The generated result.*/
88:    FeldmanProjection_n score ;
89:
90:    /** Cache for the generated time points, maintained by {@linl #write_to_score()} .*/
91:    final Map<Integer, Tp> viertel2tp = new TreeMap<>();
92:
93:    /** Aux function for {@link Agent#Agent} because "this() must be first statement".*/
94:    Haken next_th(){
95:      final int predecs = agents.size();
96:      final Agent predec = agents.get(predecs-1);
97:      return predec.hakenFolge_th.get(6-predecs);
98:    }
99:
100:   /** Aux function for {@link Agent#Agent} because "this() must be first statement".*/
101:   Haken next_dt(){
102:     final int predecs = agents.size();
103:     final Agent predec = agents.get(predecs-1);
104:     return predec.hakenFolge_dt.get(6-predecs);
105:   }
106:
107:
108:   /** Generator for one voice.*/
109:   class Agent {
110:
111:     /** Position in the sequence of canonically starting voices.*/
112:     final int number ;
113:
114:     /** The list of dodecaphonic modes (taken from {@link #reihen} for the pitch parameter. */
115:     final int reihe_th;
116:
117:     /** Very first haken form.*/
118:     final Haken startHaken_th ;
119:
120:     /** Very first haken form.*/
121:     final Haken startHaken_dt ;
122:
123:     /** List of all hakens for this voice, filled by {@link #generate_haken(int)}.*/
124:     List<Haken> hakenFolge_dt = new ArrayList<>();
125:
126:     /** List of all hakens for this voice, filled by {@link #generate_haken(int)}.*/
127:     List<Haken> hakenFolge_th = new ArrayList<>();
128:
129:     /** The resulting voice, contained in the resulting {@link #score}.*/
130:     Vox voice ;
131:
132:     /** Constructor with explicit haken forms. Called from outside only for
133:      *  the very first agent.
134:      */
135:     Agent (final Haken startHaken_th,
136:            final Haken startHaken_dt){
137:       this.number = agents.size();
138:       agentsByName.put(voiceNames[number], this);
139:       agents.add(this);
140:       this.reihe_th = number ;
141:       // duration follows always reihe 3
142:       this.startHaken_th = startHaken_th ;
143:       this.startHaken_dt = startHaken_dt ;
144:     }
145:
146:     /** Constructor called from outside for all agents except the first.
147:      *  Every voice (except the first) starts in unison with its
148:      *  predecessor w r t the haken-form for parameters "th" and "dt".
149:      *  The second voice has one predecessor and follows 5 hakens later,
150:      *  the third voice has two predecessors and follows 4 hakens later, etc.
151:      */
152:     Agent(){
```

```
153:       this(next_th(), next_dt());
154:     }
155:
156:
157:     /** Fill the sequences of hakens {@link #hakenFolge_th} and {@link #hakenFolge_dt}
158:      *  starting with the start haken in {@link #startHaken_th}  and
159:      *  {@link #startHaken_dt}. For the former the own dodecaphonic series (="Reihe")
160:      *  as given by {@link #reihe_th} is used, multiplied by the 0th reihe.
161:      *  For dt, the haken form changes only every second haken and always reihe #3 is used.
162:      */
163:     void generate_haken(final int count){
164:       int index = 0 ;
165:       while(index<count){
166:         hakenFolge_dt.add(startHaken_dt.apply(reihen[3][(index/2) % 4])) ;
167:         hakenFolge_th.add(startHaken_th.apply(reihen[0][(index/4) % 4])
168:                           .apply(reihen[reihe_th][index % 4])) ;
169:         index++;
170:       }
171:     }
172:
173:     /** Keeps track which haken (identified by its index in the hakenfolge)
174:      *  starts at which time position (in 1/4 notes). Needed only to realize the
175:      *  canonical start of the voices.
176:      */
177:     List<Integer> haken2start = new ArrayList<>();
178:
179:     /** Keeps track which event in the generated output realizes the start of which
180:      *  haken (relative to the haken list of the same voice)
181:      */
182:     Map<Integer,Integer> event2haken = new TreeMap<>();
183:
184:     /** Pre-Feldman event data: start point measured in 1/4.*/
185:     List<Integer> event2start = new ArrayList<>();
186:
187:     /** Pre-Feldman event data: duration measured in 1/4.*/
188:     List<Integer> event2duration = new ArrayList<>();
189:
190:     /** Pre-Feldman event data: register as a RelWert O/M/U.*/
191:     List<RelWert> event2wert = new ArrayList<>();
192:
193:     /** Pause between two haken realizations.*/
194:     public static final int haken_additionalPause =2 ;
195:
196:     /** Number of haken modulo 4 the start of which may be subject to an overlap. */
197:     Set<Integer> overlap_positions = literalSet(1,2,3);
198:     // literalSet(2) ?? AUSTESTEN
199:     // 1 and 3 can never overlap because the duration haken is the same !
200:
201:     /** Global counter for generating the pre-Feldman event data
202:      *  in {@link #event2start}, {@link #event2duration}, and {@link #event2wert}.
203:      *  Must be global because updated by both,
204:      * {@link generate_eventData()} and {@link generate_eventData(RelWert,RelWert)}.
205:      */
206:     int viertel ;
207:
208:     /** Step through the seres of haken in {@link #hakenFolge_dt} and
209:      *  {@link #hakenFolge_th} and translate each haken into three pre-Feldman events
210:      *  in {@link #event2start}, {@link #event2duration}, and {@link #event2wert}-
211:      *  Two adjacent hakens may share the middle event, if it has the same parameters.
212:      *  Otherwise the additional pause {@link #haken_additionalPause} is inserted.
213:      */
214:     void generate_eventData(){
215:       // the second voice has one predec and follows 5 hakens later:
216:       viertel = (number == 0) ? 0
217:         : agents.get(number-1).haken2start.get(6-number) - haken_additionalPause ;
218:       final int hakens = hakenFolge_dt.size();
219:       RelWert last_th = null ;
220:       RelWert last_dt = null ;
221:       for (int i = 0 ; i<hakens; i++){
222:         final Haken haken_th = hakenFolge_th.get(i);
223:         final Haken haken_dt = hakenFolge_dt.get(i);
224:         boolean skip = false ;
225:         if (overlap_positions.contains(i%4))
226:           skip = haken_th.getFirst()==last_th &&  haken_dt.getFirst()==last_dt ;
227:         // && last_dt==RelWert.M  ?? = anderes Konzept: auslöschung nur in der Mitte ! FIXME
228:         event2haken.put(event2start.size()  - (skip ? 1 : 0), i);
```

```
229:         if (skip)
230:           msgr.receive(SimpleMessage.log("in voice %d skipped the first of haken %d",
231:                                          number, i));
232:         if (!skip)
233:           viertel += haken_additionalPause ;
234:         haken2start.add(viertel);
235:         if (!skip)
236:           generate_eventData(haken_th.getFirst(), haken_dt.getFirst());
237:         generate_eventData(haken_th.getMiddle(), haken_dt.getMiddle());
238:         generate_eventData(last_th=haken_th.getLast(), last_dt=haken_dt.getLast());
239:       }
240:     }
241:
242:     // FIXME Relwert ist noch keine enum !?!?
243:     /** Translate twp relative values O/M/U  for duration and pitch into pre-Feldman events
244:      *  in {@link #event2start}, {@link #event2duration}, and {@link #event2wert}.
245:      *  Start point is at {@link viertel}, which is advanced according to duration and gap,
246:      *  which both depend on the duration haken.
247:      */
248:     void generate_eventData(final RelWert th, final RelWert dt
249:                             /* GLO IN OUT viertel, event2start/diration/wert */ ){
250:       if(dt==RelWert.O){
251:         event2start.add(viertel);
252:         event2duration.add(8);
253:         viertel+=8;
254:       }
255:       else if (dt==RelWert.M){
256:         event2start.add(viertel+1);
257:         event2duration.add(3);
258:         viertel+=5; }
259:       else {
260:         event2start.add(viertel);
261:         event2duration.add(1);
262:         viertel+=1;
263:       }
264:       event2wert.add(th);
265:     }
266:
267:     /** Generate a new score and transform the pre-Feldman data
268:      *  from {@link #event2start}, {@link #event2duration}, and {@link #event2wert} to
269:      *  tscore events. Therefore Tps must be synthesized whenever necessary.
270:      */
271:     void write_to_score(){
272:       final String voicename = voiceNames[number] ;
273:       voice = new Vox(score.part, voicename);
274:       score.part.get_voicesBySource().add(voice);
275:       score.part.get_voices().put(voicename, voice);
276:       score.normalizedEvents.put(voice, new ArrayList<>());
277:       Timelesses.addTl(voice, FeldmanProjection_n.timelessTag_shortName, voicename);
278:       Timelesses.addTl(voice, FeldmanProjection_n.timelessTag_longName, voicename);
279:       final int events = event2start.size();
280:       for (int i = 0 ; i<events; i++){
281:         final Tp start = get_tp(event2start.get(i));
282:         final Event event = voice.newEvent(noLocation(), start);
283:         if (event2haken.containsKey(i))
284:           score.adjunctLabel.put(event,String.valueOf(event2haken.get(i)));
285:         score.normalizedEvents.get(voice).add(event);
286:         final RelWert wert = event2wert.get(i);
287:         if (wert==RelWert.O)
288:           score.registerH.put(event,"H");
289:         else if (wert==RelWert.M)
290:           score.registerM.put(event,"M");
291:         else
292:           score.registerL.put(event,"L");
293:         final Tp end = get_tp(event2start.get(i)+event2duration.get(i));
294:         score.event2endTp.put(event, end);
295:       }
296:     }
297:   }//class Agent
298:
299:
300:   /** Generate a tp obejct in the output score.
301:    *  Viertel is a zero-based position. Thus related to rational time points only by a factor.
302:    */
303:   Tp get_tp (final int viertel){
304:     if (viertel2tp.containsKey(viertel))
```

```
305:       return viertel2tp.get(viertel);
306:     final int rest = viertel % 4 ;
307:     if (rest==0)
308:       return get_top(viertel/4);
309:     final Tp left = get_top(viertel/4);
310:     final Tp tp =  left.makeDivision(get_top(viertel/4 +1), 4, true).getTp(rest);
311:     viertel2tp.put(viertel, tp);
312:     final Rational v = Rational.valueOf(viertel, 4);
313:     score.rat2tp.add(v, tp);
314:     score.tp2rat.put(tp, v);
315:     return tp ;
316:   }
317:
318:   /** Generate a tp obejct in the output score.
319:    *  Ganze is a zero-based position. Thus related to rational time points only by a factor.
320:    */
321:   TpTop get_top (final int ganze){
322:     if (score.numericName2tpTop.containsKey(ganze+1))
323:       return score.numericName2tpTop.get(ganze+1);
324:     final TpTop tp = new TpTop(String.valueOf(ganze+1));
325:     score.numericName2tpTop.put(ganze+1, tp);
326:     score.tpTop2numericName.put(tp, ganze+1);
327:     viertel2tp.put(ganze*4,tp);
328:     final Rational v= Rational.valueOf(ganze);
329:     score.rat2tp.add(v, tp);
330:     score.tp2rat.put(tp, v);
331:     return tp ;
332:   }
333:
334:   /** Generate the FeldmanProjection_n score object ex ovo and determinstically.
335:    *  First construct the agents and generate their series of hakens by
336:    *  {@link Agent#generate_haken(int)}..
337:    *  (For starting agent n+1, the series of agent n is needed.)
338:    *  <br/>
339:    *  In a second step translate the haken series into event series by
340:    *  {@link Agent#generate_eventData()}.
341:    *  This is independent for each agent/voice.
342:    *  <br/>
343:    *  Finall call for each agent {@link Agent#write_to_score()} and
344:    *  add some timeless parameters to the result.
345:    */
346:   FeldmanProjection_n generate(){
347:     final Agent agent0 = new Agent(UM,OM);
348:     agent0.generate_haken(5+4+3+2+1+4); // = 19 = 76 events - auslöschungen
349:
350:     final Agent agent1 = new Agent();
351:     agent1.generate_haken(4+3+2+1+4);
352:     final Agent agent2 = new Agent();
353:     agent2.generate_haken(3+2+1+4);
354:     final Agent agent3 = new Agent();
355:     agent3.generate_haken(2+1+4);
356:     final Agent agent4 = new Agent();
357:     agent4.generate_haken(1+4);
358:     final Agent agent5 = new Agent();
359:     agent5.generate_haken(4);
360:
361:     for (Agent agent : agents)
362:       agent.generate_eventData();
363:
364:     score  = new FeldmanProjection_n(new Part(new TimeScape(),  // FIXME RAUS !?!?!
365:                                               noLocation(), partName),
366:                                      msgr);
367:     for (Agent agent : agentsByName.values())
368:       agent.write_to_score();
369:
370:
371:     Timelesses.addTl(score.part, FeldmanProjection_n.timelessTag_interStaffGap,
372:                      "1.5"); // FIMXE default?
373:     Timelesses.addTl(score.part, FeldmanProjection_n.timelessTag_endMark,
374:                      "20231021");
375:     Timelesses.addTl(score.part,
376:                      FeldmanProjection_n.timelessTag_author, "ML");
377:     Timelesses.addTl(score.part,
378:                      FeldmanProjection_n.timelessTag_opusName,
379:                      "Kleine Studie nach Feldman 3");
380:     score.maxMeasuresPerPage = 12 ;  // FIXME default ??
```

```
381:     score.firstTp = (TpTop) score.numericName2tpTop.get(score.numericName2tpTop.firstKey());
382:     score.lastTp = (TpTop) score.numericName2tpTop.get(score.numericName2tpTop.lastKey());
383:     final int lastMeasureNumber = score.tpTop2numericName.get(score.lastTp);
384:     for (int i = 0 ; i<= lastMeasureNumber ; i += score.maxMeasuresPerPage)
385:       score.pageStarts.add(i);
386:
387:     // add missing tpTops = make dense : FIXME abstrahieren !?!?! :
388:     final int lastInt = score.tpTop2numericName.get(score.lastTp);
389:     for (int i = 1 ; i<lastInt; i++){
390:       if (score.numericName2tpTop.containsKey(i))
391:         continue ;
392:       final TpTop tptop = new TpTop ("synth"+i);
393:       score.numericName2tpTop.put(i,tptop);
394:       score.tpTop2numericName.put(tptop,i);
395:       Rational rat = Rational.valueOf(i-1);
396:       score.tp2rat.put(tptop,rat);
397:       score.rat2tp.add(rat,tptop);
398:
399:     }
400:     return score ;
401:   }
402: }
403: //eof
```