Two algorithms for the instanciation of structures of musical objects

Bernard Bel

▶ **To cite this version:**

Bernard Bel. Two algorithms for the instanciation of structures of musical objects. 2005. halshs-00004504

HAL Id: halshs-00004504
https://shs.hal.science/halshs-00004504

Preprint submitted on 30 Aug 2005

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Two algorithms for the instanciation of structures of musical objects

Bernard Bel

Abstract

This is an extended and revised[1] version of the paper: *Symbolic and Sonic Representations of Sound-Object Structures* published in M. Balaban, K. Ebcioglu & O. Laske (Eds.) "Understanding Music with AI: Perspectives on Music Cognition", AAAI Press (1992, pp.64-109).

A representational model of discrete structures of musical objects at the symbolic and sonological levels is introduced. This model is being used for the design of computer tools for rule-based musical composition in which the low-level musical objects are not notes, but "sound-objects", thereby meaning arbitrary sequences of messages dispatched to a real-time digital sound processor.

"Polymetric expressions" are string representations of concurrent processes that can easily be handled by formal grammars. These expressions may not contain all the information needed for synchronizing the whole structure of sound-objects, i.e. determining their strict ordering on (symbolic) time. In response to this, the notion of "symbolic tempo" is introduced: ordering all objects in a structure is possible once their symbolic tempos are known. Rules for assigning symbolic tempos to objects are therefore proposed. These form the basis of an algorithm interpreting incomplete polymetric expressions. The relevant features of this interpretation are commented.

An example is given to illustrate the advantage of using (incomplete) polymetric representations instead of conventional music notation or event tables when the complete description of the musical piece and/or its variants calls for difficult computations of durations.

Given a strict ordering of sound-objects summarized in a "phase table" representing the complete polymetric expression, the next step is to calculate the dates at which messages should be dispatched. This requires a description of "sound-object prototypes" along with their metric/topological properties, and various parameters related to the musical performance (e.g. "smooth" or "striated" time, tempo, etc.). These properties are discussed in detail and a time-polynomial constraint-satisfaction algorithm for the time-setting of sound-objects in a polymetric structure is introduced. Typical examples computed by this algorithm are shown and discussed.

Keywords

Synchronization, polymetric structures, musical objects, sonology, sound-objects.

[1] Revised on 31/12/94

CONTENTS

Two algorithms for the instantiation of structures of musical objects

Bernard Bel

> […] not only do individuals and groups give different verbal meanings to music; they also conceive its structures in ways that do not permit us to regard musical parameters as objective acoustical facts. In music, thirds, fourths, fifths, and even octaves, are social facts, whose syntactical behaviour can differ as much as that of *si*, *see*, and *sea*, *beau*, *bow*, and *bo*, or *buy*, *bye*, *by* and *bai.*
>
> [Blacking 1984, p.364]

Pierre Schaeffer, the father of *musique concrète* [Schaeffer 1966], introduced a taxonomy of **musical objects** on the basis of their distinctive features. Musical objects may also be called **sonemes**, i.e. equivalence classes on acoustic musical events, each musical event being in turn an acoustic variant of the soneme. The musical object approach certainly contributed to significant developments of electroacoustic music. Nevertheless, composers who had been in search for a higher degree of freedom in sound manipulation felt the need to explore the potential of digital sound synthesis:

> I felt that electronic music yielded dull sounds that could only be made lively through manipulations which, to a large extent, ruined the control the composer could have over them. On the other hand, *musique concrète* did open an infinite world of sounds to music — but the control and manipulation one could exert upon them was rudimentary with respect to the richness of the sounds, which favored an aesthetics of collage.
>
> [Risset 1989:67]

Followers of *musique concrète* have now replaced audio tape with sampling techniques and scissors with various kinds of sequencers, notators or tools for computer-aided composition. As far as digital sound synthesis is concerned, most digital sound processors offer an access to real-time external control through communication devices such as MIDI[2]. Therefore, both "musical object" and "sound model" designers [Borin et al. 1990] are now operating in environments enabling a control on both sound structures and acoustical parameters. Nevertheless, most of the music software available on the market is specialized either for the manipulation of structures (mainly cut-and-paste operations) or sounds (digital filtering or controlling parameters in a sound processor).

The present study is an attempt to deal with musical objects that may be handled both at the symbolic level and at the level of "elementary actions". The nature of these actions is not specified because it depends on the hardware/software configuration of the sound processor and, to a lower extent, on the communication device used for the real-time control of the processor. The motivation of our project, therefore, has been the

[2] *Musical Instrument Digital Interface*.

implementation of efficient procedures handling musical objects as "lists of messages" (at a macro or micro-level) unrelated to music notation conventions. Some information about the task environment of this work is given in §2.

The representation of time and musical structures is discussed in §3.

In §5 and §6.1 we introduce **polymetric expressions**, string representations of universes of concurrent processes, along with an algorithm that infers the missing time information in incomplete expressions. An application to music in conventional notation is discussed in §9.

In §10-ff we take into account that the terminal symbols of a polymetric expression may be arbitrary labels assigned to **sound-objects**. A sound-object may be viewed as an instance of some predefined **sound-object prototype**. Each prototype contains (1) a sequence of (time-stamped) messages destined to a sound processor and (2) a list of (inheritable) **sonic properties**. These properties are defined in §12-13.

In §10 a **time-setting algorithm** is informally introduced. This algorithm calculates the accurate positioning of all sound-objects in a sound-object structure, given the definitions of their prototypes. Its main operation is the resolution of a system of constraints resulting from the sonic properties of sound-objects, the **nature of time** (**smooth** or **striated**) and its **structure** (e.g. a metronomic or irregular beat). The time-setting algorithm is analyzed in §16 and examples of its output are commented in §18.

1. Related work

In the same way computational linguists, e.g. [Jakobson 1963; Chomsky & Halle 1968], have attempted to deal with phonological models of natural language, the work presented here is part of the design of a **sonological** component of musical (formal) grammars:

> Supposed sonological segments of music are composed of sound-objects realizing certain syntactical structures.
>
> It is reasonable to assume that no unambiguous one-to-one correlation of such segments with acoustical (i.e., physical) sound properties exists. On the level of the *sonic* representation — mediating between the acoustical and the sonological levels — one therefore expects to find a representation of properties which are acousmatic[3] (independent of sound sources in the physical sense) as well as asyntactic (independent of syntactical classes of musical formatives).
>
> [Laske 1972:30]

The aim of this work is to introduce "performance parameters" into musical pieces generated by computations on abstract symbols. Our basic assumption is that these parameters should be determined altogether by (1) the musical structure, (2) interpretation rules, and (3) the properties of sound-objects.

The notion of symbolic time introduced in part A is close to Jaffe's [1985] **basic time** and to **virtual time** in the *Formula* musical programming environment [Anderson et Kuivila 1989:11-23].

---

[3] The term **acousmatic** refers to sounds whose physical source is either unknown or is intentionally neglected; the term is taken over from Pierre Schaeffer's *Traité des Objets Musicaux*, […] 1966:61. [Original footnote]

The **sonological interpretation** [Laske 1972:24] of a musical item may first be handled by a rewriting system whose terminal alphabet is a finite set of sound-object labels.[4] Rules in this system reflect the **sonological properties** of sound-objects. The second part of this paper deals with a second step of the sonological interpretation: the inference of missing precedence/simultaneity relations in structures of sound-objects. The intuitive interpretation of superimpositions (see §5.1), leading to the concept of **symbolic tempo** (see §6.1), may be viewed as a kind of **semantic interpretation** of time structures.

Sonic properties[5] of sound-objects are features unrelated with the syntactic structure of musical pieces in which they appear. The only properties considered in this paper are the ones controlling the time-scaling of sound-objects (i.e. **metrical** properties) and acceptable mutual relations of their time-span intervals (i.e. **topological** properties). Starting from a complete representation of a sound-object structure, these sonic properties are taken into account for determining the accurate timing of sound-objects, as shown in §12-13-16.

Our approach may be viewed as a compromise between **top-down** (goal-driven) and **bottom-up** (data-driven) compositional strategies in music. In the former, rules and procedures are used to compute the final sound output on the basis of information entirely imbedded in its symbolic description. In the latter, a structure may "emerge" from some specific arrangement of elementary acoustic events. The top-down strategy is the main one available in conventional score editors and MIDI composition tools. A bottom-up strategy for the design of discrete structures, which inspired this work, has been proposed by Stroppa (see §10, [Duthen & Stroppa 1990]).

Since the algorithms described here are neither related to a particular musical system nor to sound generation techniques or data-communication standards, they could as well be applied to the time-setting of processes outside the domain of computer music. There is formally no difference between a message destined to a sound processor and one controlling a laser beam, a robot, etc. Nevertheless, for the sake of clarity, the concepts and terminology will be introduced in reference to a musical environment.

2. The task environment of this study

The algorithms presented below have been implemented in a computer environment for **design-based** (stipulatory) or **improvisational rule-based composition** [Laske 1989, pp.51,53] called **Bol Processor BP2**, in which it is possible to design sets of musical items by way of rewriting rules.[6]

Several operational modes are available in BP2, from the one that leaves all decisions to the machine (stochastic improvisation) to the one that forces the composer to take stepwise decisions.

The interaction of modules is summarized in the block diagram of Fig.1.

---

[4] See for instance context-sensitive substitutions, §3 and appendix 3 in [Bel 1991a].

[5] The expression "sonic properties" is borrowed from [Laske 1972:27] but it is used here in a more restrictive sense. In Laske's view [*op.cit*.:30], a sound-object itself is a set of sonic properties.

[6] *ibidem.*

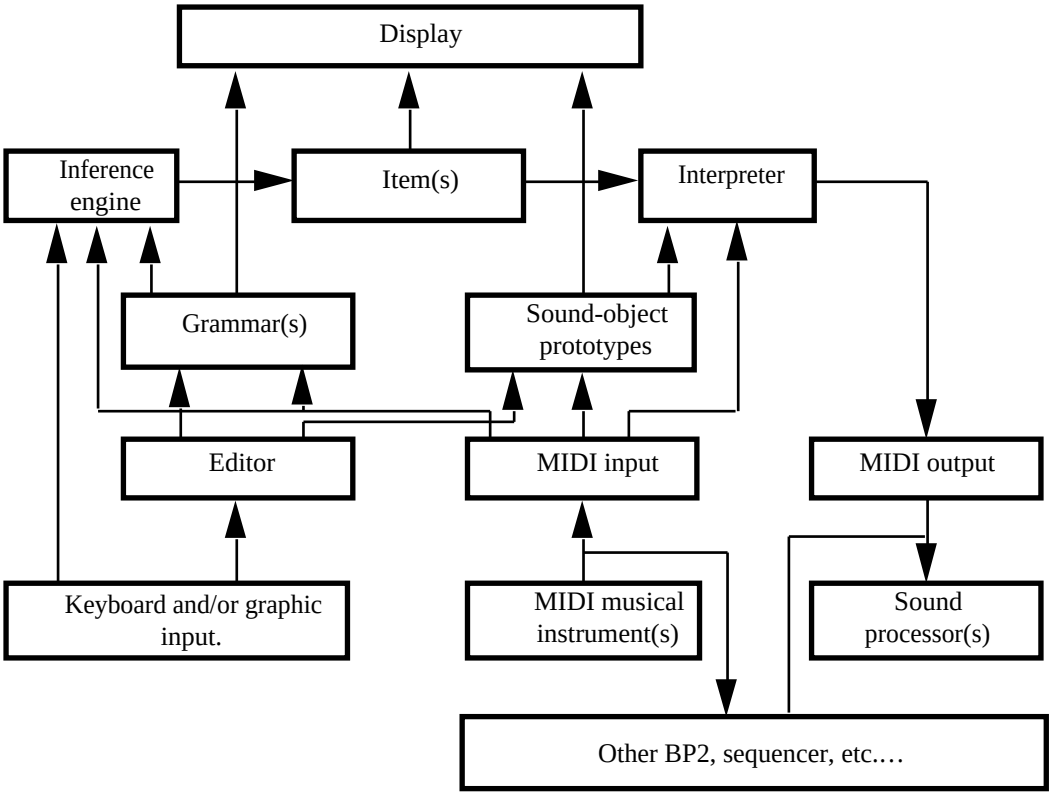


Fig.1 A block diagram of BP2

Three fields are used for storing a grammar, items generated by the grammar (on the basis of decisions taken by the inference engine) and "sound-object prototypes" loaded from a MIDI musical instrument (and edited manually). The terminal alphabet of the grammar is the set of labels of sound-objects. The interpreter works in three stages:

1) The item generated by the grammar is interpreted as a polymetric expression. The output is a complete expression (i.e. a bidimensional array of terminal symbols). (See §3.2, §7.2)

2) The expression is interpreted as a sound-object structure, using information about the structure of time and object prototype definitions. The main output is an array containing the performance parameters of objects in the structure: their start/clip dates, time-scale ratios, etc. (See §11)

3) MIDI messages are dispatched in real time to control the generation of sound-objects by the sound processor. (See the time-setting function in the appendix, §3)

The block diagram indicates that an external control can be exerted on the inference engine, grammars, and the interpretation module. Specific MIDI messages may be assigned to changes of rule weights, tempo, and the nature of time (striated/smooth). Other messages may be used for synchronizing events in the performance or assigning computation time limits. These features are used in improvisational rule-based composition.

Several BP2's may be linked together and with other devices such as MIDI sequencers. Messages on the different MIDI channels may be used for making machines communicate or controlling several sound processors. Therefore it must be kept in mind that "sound-objects" do not necessarily produce sounds. Depending on the implementation they may contain any kind of control/synchronization message as well.

3. The basic representation issues

3.1 Representation of discrete sound-object structures

Let us assume that "a", "b", "c", "e", "f", "g" and "-" are labels of arbitrary sound-objects. Label "-" may be reserved to silences, which are viewed as particular objects.

The picture below represents a structure of two sequences which, in first approximation, may be notated

$$S_1 = a\ b\ c\ a \quad \text{and} \quad S_2 = e\ -\ f\ g$$

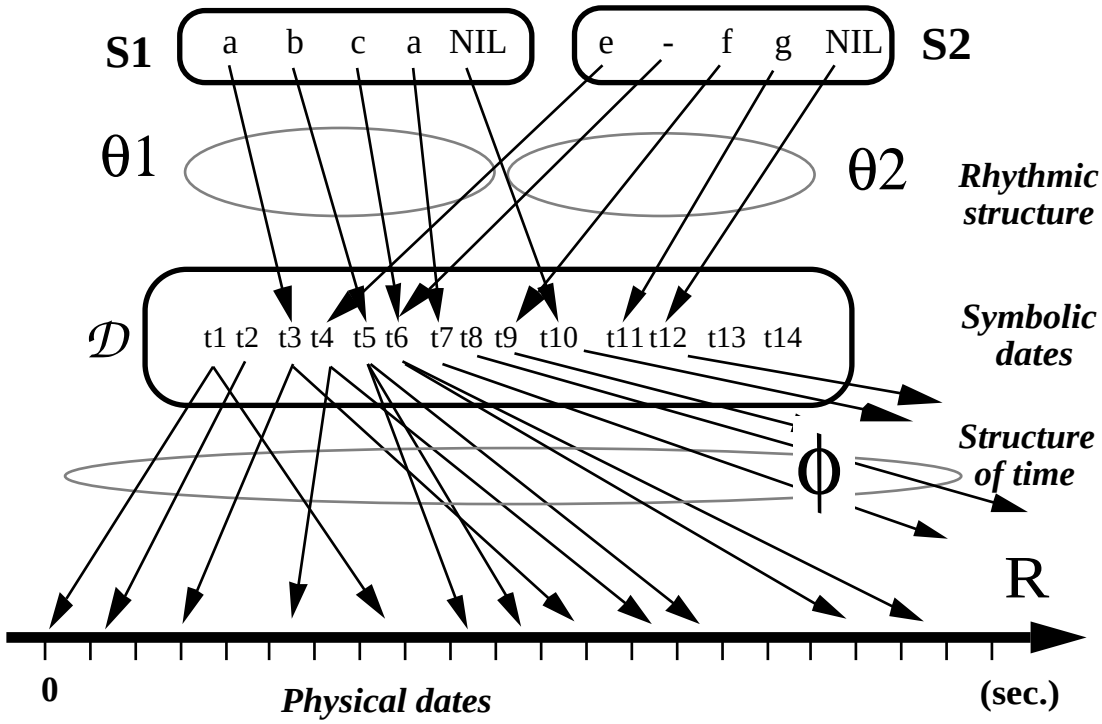


Fig.2 A representation of sequences S1 and S2

In Fig.2 a set of strictly ordered **symbolic dates** $\mathcal{D}$ = {t1,t2,…} is introduced along with θ_i , an injective mapping of each S_i into $\mathcal{D}$. By convention, each θ_i is a monotonous increasing function: sequentiality implies that all objects appearing in a sequence are ordered in increasing symbolic dates. Each mapping θ_i may in turn be viewed as a

restriction to S_i of a general mapping θ which we call the **rhythmic structure**[7] of the sound-object structure. The utility of "NIL" markers will be shown later.

Mappings of sequences to the set of symbolic dates introduce information about the ordering of any pair of events belonging to either sequence. In this way, a structure of sound-objects is described symbolically. Here, for instance, S_1 and S_2 are partly overlapping.

The set of symbolic dates $\mathcal{D}$ is then mapped to physical time, i.e. the set of real numbers R. We call this mapping Φ the **structure of time**.[8] In the example shown above, Φ is a multivocal mapping, which means, for instance, that each sound-object "a" and "e" at symbolic date t_3 would be performed twice. In the rest of this paper only strictly increasing (univocal) mappings will be considered, i.e.:

$$\forall i,j \in N, \quad t_i < t_j <=> \Phi(t_i) < \Phi(t_j) \ .$$

In this case, if we consider $Dist(t_i,t_j) = |\Phi(t_j) - \Phi(t_i)|$ (the absolute value of the difference), *Dist* is a distance on $\mathcal{D}$. Besides, since

$$\forall i,j,k \in N, \quad Dist(t_i,t_j) + Dist(t_j,t_k) \geq Dist(t_i,t_k)$$

($\mathcal{D}$,Dist) is also a *metric* space. ($\mathcal{D}$,Dist) is *Euclidian* (**metronomic time**) if the additional property holds:

$$\forall i,j,k,l \in N, \quad j - i = l - k \ => \Phi(t_j) - \Phi(t_i) = \Phi(t_l) - \Phi(t_k)$$

The composition of the two mappings ($\Phi \ . \theta$) is the **in-time structure** of the musical item, i.e. the mapping that permits its actual performance. Structure of time and in-time structures are two concepts borrowed from Xenakis [1963; 1972:57]. We find these concepts essential as they deal with sets of physical dates not necessarily structured as a Euclidian space.

3.2 Phase diagram

Both sequences of this example may be represented together in a single array (the **phase diagram**), the columns of which are labelled and ordered on symbolic dates:

t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	t11	t12	t13	t14
_	_	a	_	b	c	a	_	_	NIL	_	_	_	_
_	_	_	e	_	-	_	_	f	_	g	NIL	_	_

The array contains **empty sound-objects** "_" which may denote the prolongation of the preceding sound-object. These should not be confused with silences "-".

---

[7] This term is justified in [Bel 1990a:114].

[8] This was called *structure temporelle* by Xenakis [1963:190-1,200]

Using the information displayed in the array, S_1 and S_2 may be properly notated:

S_1 = a _ b c a _ _ $\qquad\qquad$ S_2 = e _ -_ _ f _ g

In general, there are several possible equivalent phase diagrams representing the same sound-object structure. An equivalent diagram would be for instance:

_	_	_	e	_	c	a	_	_	NIL	_	_	_	_	
_	_	a	_	b	-	_	_	f	_	g	NIL	_	_	

At this stage, we call **symbolic duration** of a sound-object the relative position of the next non-empty sound-object or "NIL" marker. A complete definition taking into account the "dilation ratio" will be proposed in §11. For example, in S_2 the symbolic durations of objects "e", "-", "f" and "g" are two, three, two and one respectively. In S_1, there are two consecutive occurrences of "a" with respective durations two and three.

If for example "a", "b", etc. would represent notes, assuming that "b" is a quarter note would imply that "e" is a half-note and "-" a dotted half-note rest.

3.3 Out-time objects

Sound-objects have strictly positive symbolic durations. In some cases it is useful to dispose of "flat" objects with null durations which we call **out-time objects**. These may be defined from sound-objects whose actions are executed "simultaneously" or in a very short sequence (see appendix, §2). In the BP2 environment, a typical application of out-time objects is the exchange of parameters or synchronization messages.

Given a sound-object labelled "a", the corresponding out-time object is labelled "<<a>>". Using this convention, a string like

<<a>> b

represents a structure in which out-time object "<<a>>" starts at the same symbolic date as sound-object "b".

4. Smooth vs. striated time

Pierre Boulez introduced the notions of **smooth time** ("temps lisse") and **striated time** ("temps strié") to characterize two typical situations in music performance. Striated time is *filled with (regular or irregular) pulse*, whereas smooth time *does not imply any counting*:

> […] dans le temps lisse, on occupe le temps sans le compter; dans le temps strié, on compte le temps pour l'occuper. […] ce sont les lois fondamentales du temps en musique.
>
> [Boulez 1963, p.107]

A particular case of striated time is the metronomic pulse. Examples of smooth time are common outside Baroque music, e.g. the slow melodic introductions in *raga* music.

In computer-generated music, these notions are bound to the structure of time (the Φ mapping): in striated time, Φ is known in advance, whereas in smooth time it is determined at the time of performance. Therefore, a **striated structure of time** is a set of physical dates defining **reference streaks** on which sound-objects should be

positioned (e.g. see §18.3.2), whereas a **smooth structure of time** is a set of dates determined by the sound-objects themselves (e.g. see §18.3.1). In both cases, however, a synchronization between different voices must be maintained.

5. The synchronization problem

We call **synchronization problem** the task of mapping all sound-objects in a structure to a set of symbolic dates. This mapping is the rhythmic structure θ of the musical piece (see §3.1).

Imbedding the complete rhythmic structure in the definitions of musical sequences introduces a rigidity that goes against the versatility of rewriting systems, as discussed in §5.1. In response to this, incomplete representations can be envisaged so long as a method is available for inferring the missing time information. A method for "synchronizing" incomplete descriptions sound-object structures, thereby completing their **sonological interpretation**, will be proposed in §7-8.

5.1 Rhythmic structures in a formal grammar — example

Suppose that we wish to superimpose two sequences A and B defined by rules:

A —> a b c
A —> d e f g
B —> h i

in which "a", "b", … "i" are labels of sound-objects. Alternate definitions of "A" indicate that it may contain either three or four objects. To start with, we do not know how to interpret the exact superimposition of two sequences: combining "abc" and "hi" may for example yield the following phase diagrams:

```
a b c       a b c       a _ b _ c _      a _ b c     etc...
h i _       _ h i       h _ _ i _ _      h i _ _

 (1)         (2)             (3)            (4)
```

Prolongational symbols "_" could also be replaced with silences "-". However, since silences do not explicitly appear in the grammar, we may postulate that creating them is not a valid choice. Further we discard interpretations (1) and (4) in which equal symbolic durations are not maintained within the same string "h i". Finally, it is reasonable to expect a synchronization of both the start and clip points of the synchronized sequences. Therefore there is no reason to start "a" before "h" as in interpretation (2).

Finally, the most intuitively appealing interpretation of a superimposition (in the absence of any additional information) is the one shown in (3). A notation of superimpositions is now introduced: {A,B} (equivalently, {B,A}) is the superimposition of sequences "A" and "B". We call "{A,B}" a **polymetric expression** whose **arguments** are "A" and "B". Nested expressions will be introduced in §5.2.

Using this notation, a grammar yielding all acceptable superimpositions of "A" and "B" would be:

S —> {A1,B1}
S —> {A2,B2}
A1 —> a _ b _ c _ B1 —> h _ _ i _ _
A2 —> d e f g B2 —> h _ i _

Once a string like "{d e f g, h _ i _}" has been produced, it is necessary to check that it contains equally many terminal symbols in both arguments, failing to which the phase diagram cannot be constructed.

Evidently it is cumbersome to be forced to give two possible versions of "B", the more so because they point to identical ratios of symbolic durations: "B2" is similar to "B1" in every respect. Ideally, the following grammar should be used:

S —> {A,B}
A —> a b c
A —> d e f g
B —> h i

expecting that there will be a method for interpreting a production like

{a b c, h i}

i.e. an **incomplete polymetric expression**, as

{a _ b _ c _ , h _ _ i _ _ }

i.e. a **complete polymetric expression** (see formal definition §7.2).

5.2 Event universe

An **event universe** (E,<,=,&) is a set structured with three relations:

1) a strict ordering which we name "before" and notate '<';

2) an equivalence relation which we name "matches" and notate '=';

3) a strict ordering which we name "then" and notate '&';

with the following consistency conditions:

$\forall(e_1,e_2,e_3) \in E^3$,

$e_1 \,\&\, e_2 \;\Rightarrow\; e_1 < e_2$;

$e_1 \,\&\, e_2$ and $e_2 = e_3 \;\Rightarrow\; e_1 \,\&\, e_3$;

$e_1 < e_2 \;\Rightarrow\; \text{not}\,(e_1 = e_2)$;

$e_1 < e_2$ and $e_2 = e_3 \;\Rightarrow\; e_1 < e_3$.

It is easy to prove that: $e_1 < e_2$ and $e_1 = e_3 \;\Rightarrow\; e_3 < e_2$.

Informally, the "<" ordering denotes a precedence between events while the "&" ordering denotes sequentiality.

The word "event" may be used to denote a time point, a time interval, or any other entity on which these relations may be applied meaningfully. In this study, events denote sound-objects ordered by their symbolic dates.[9]

Let there be a (bi-coloured) graph G such that:

$$e_1 < e_2 \quad \text{or} \quad e_1 = e_2 \;\Rightarrow G(e_1,e_2)$$

[9] In [Bel 1990b] the pair containing a sound-object label and a symbolic date is called a **time-object**.

In general, G is not a complete graph, which amounts to say that there are events in the universe that cannot be compared through "<" or through "=". We call this an **incomplete universe**. Solving the synchronization problem, therefore, amounts to inferring enough "<" and "=" relations so that there transitive closure may yield a complete graph G.

When G is a complete graph we may build a partition using the equivalence relation "=". Classes of this partition are strictly ordered with "<". The index of each class under this order may be used as a **symbolic date** of events collected in the class, as defined in §3.1.

Conversely, if all events have been assigned symbolic dates it is easy to build a unique complete graph G.

5.3 A graphic representation

1) Each event is represented as a labelled edge:

Fig.3

2) Relations $e_i = e_j$ and e = e (reflexivity) are represented:

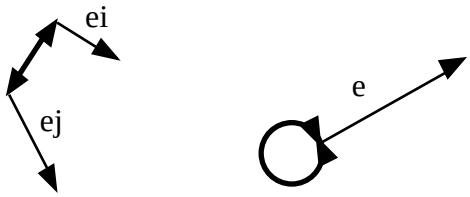


Fig.4

3) Relation $e_i < e_j$ is represented:

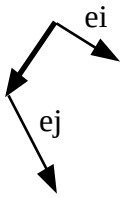


Fig.5

4) Relation e_i & e_j is represented:

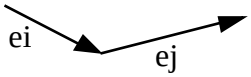


Fig.6

(or equivalently):

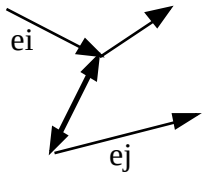


Fig.7

5) We decide to represent only a subset of relations so that the transitive closure may yield all known relations. For instance, a sequence may always be shown as a unilinear graph from which a complete graph may be inferred:

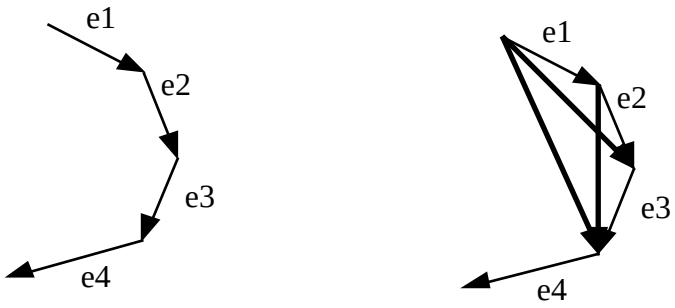


Fig.8

6. A method for synchronizing concurrent processes

In this paragraph we introduce the concept polymetric structure and the rules used for inferring "complete" polymetric expressions (see §7), i.e. to solve the synchronization problem.

Theorem

Any complete event universe E may be partitioned in sequences:

$E = S_1 \cup \ldots \cup S_k$

such that each S_i is totally ordered with "&".

Proof

A "&" relation between any pair of events may be found provided that "dummy" events are introduced for the sake of synchronization. Let there be for example two events

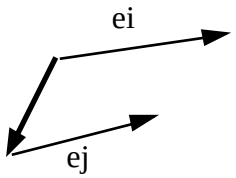


Fig.9

mapped to the following time-span intervals:

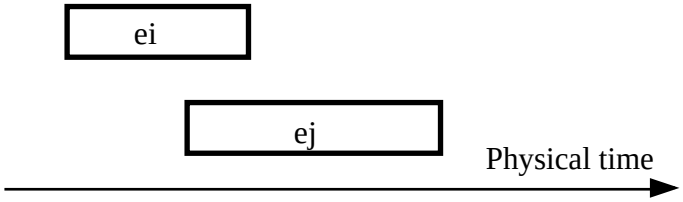


Fig.10

We create $\lambda 1$ and $\lambda 2$ such that:

$$e_i \ \& \ l1 \ , \quad e_j \ \& \ l2 \ , \quad e_j < l1 \ , \quad l1 < l2 \ ,$$

which yields the following representation

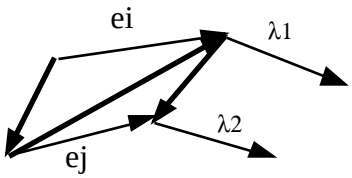


Fig.11

in which a complete information on sequentiality is contained. Events e_i and e_j are now part of sequences. ■

6.1 Polymetric expressions

Our interest is to deal with an event universe described in terms of sequences, with certain restrictions on explicit time relations which make it possible to use a bracketed string notation. The bracketed expression may later be computed to yield a complete representation.

Each sequence S_i of the event universe is notated as a string:

$$S_i = e_{i1} \ldots e_{ip} \quad \text{such that} \quad e_{ij} < e_{ik} \iff j < k$$

In this representation, events $e_{i1},\ldots, e_{ip}$ are labels of sound-objects (as in §3.2).

Under certain conditions an event universe may be notated as a **polymetric expression**. [Bel 1990b, chapter VII.]. For example, the following (incomplete) universe

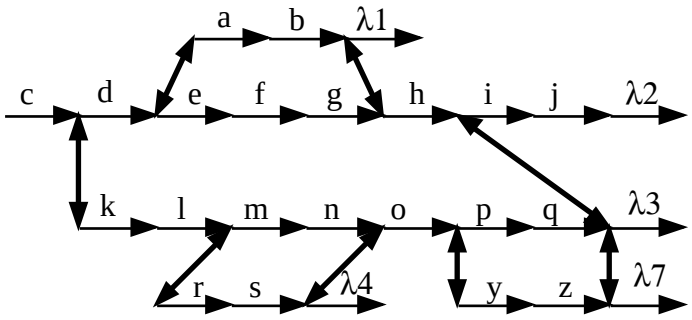


Fig.12 A polymetric structure

leads to the (nested) polymetric expression:

c { d { e f g, a b } h, k l { m n, r s } o { p q, y z } } i j

which is an extension of the notation introduced in §5.1.

Conversely, a polymetric expression denotes a universe called a **polymetric structure**.

Events λ1, λ2, … are not indicated in the polymetric expression because the closing bracket is sufficient to mark the end of a sequence. Other equivalent expressions may be written since the order of arguments between brackets is arbitrary.

The "precedence" relation is not shown on Fig.3. It is easy to infer, for instance, that "d" precedes "r", but no such relation can be proved between "f" and "n". Both the graphic representation and the polymetric expression are therefore **incomplete**.

The polymetric expression representing the event universe of Fig.12 is easy to write because it describes a universe partitioned in sequences. Each sequence is an argument of the expression. Nevertheless, there are (incomplete) event universes that do not lend themselves to a polymetric representation even though they are partitioned in sequences. Fig.13 is an example. The necessary properties enabling a polymetric representation of an incomplete event universe are listed in [Bel 1990b:109].

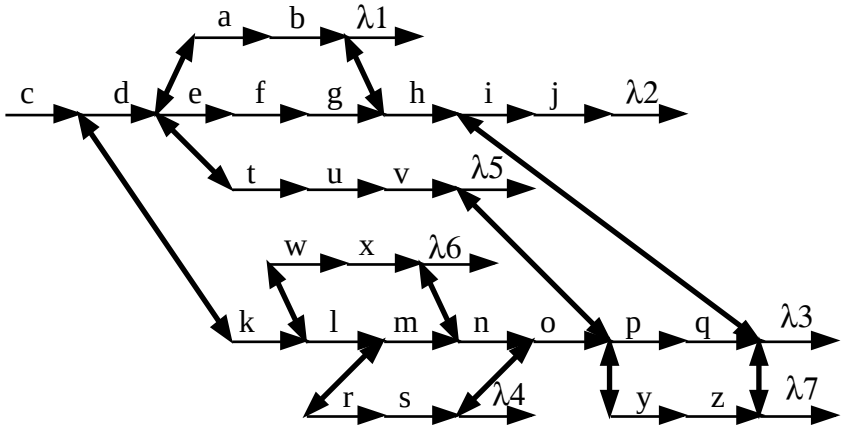


Fig.13 An arbitrary event universe

In spite of this, any **complete** event universe, i.e. one in which all precedence relations are known, may be represented with a polymetric expression. (See §7.1 or [Bel 1990b:110]) Therefore, in this study only (possibly incomplete or inconsistent) polymetric structures will be considered.

6.2 Inferring missing relations

The problem is to find a general approach to the synchronization problem in an event universe represented with a polymetric expression. In the universe shown Fig.12, for instance, we hope to know the relative ordering of sound-objects "g" and "n". Evidently, if these objects were (metric) time intervals stringed together in sequences, then computing durations would solve the problem. This is the traditional numerical approach. Allen [1983] has proposed a more flexible representation, starting from the formalization of all possible topological configurations of time-span intervals, from which it is possible to infer the list of plausible temporal relations between any pair of objects.[10] We estimate, though, that reasoning on time-span intervals at such an abstract level of the musical representation is too restrictive. In §16 it will be shown how to deal with time-span intervals in a realistic sense.

6.3 Symbolic tempo

The concept of "tempo" is introduced here in response to the bias against time-span intervals. This concept is purely abstract even though, at a lower level, it has implications on the locations of sound-objects on the physical time axis (see §10, §15.2).

[10] Similar representations of time relations in music have been suggested by Vecchione [1984] and Oppo [1984].

Let E be an event universe. We call **symbolic tempo** any mapping V of E to the set of strictly positive rational numbers Z^+ that fulfills the property:

Consistency property

Let "<" denote the "precedence" relation and "=" the "simultaneity" relation.

Let there be two sequences

$e_i \ldots e_{i+p}$ and $e_j \ldots e_{j+p}$

such that $e_i = e_j$ (i.e. starting on simultaneous events)

Then:

$$\sum_{l=i}^{i+p-1} \frac{1}{V(e_l)} = \sum_{l=j}^{j+q-1} \frac{1}{V(e_l)} \iff e_{i+p} = e_{j+q}$$

$$\sum_{l=i}^{i+p-1} \frac{1}{V(e_l)} < \sum_{l=j}^{j+q-1} \frac{1}{V(e_l)} \iff e_{i+p} < e_{j+q}$$

The preceding relations make sense intuitively if one equates $\frac{1}{V(e_i)}$ to the symbolic duration of e_i. It can be proved that if all tempos are known then the synchronization problem can be solved using these relations. [Bel 1990b, theorem VII.2] The problem, therefore, is to define a set of rules for tempo assignment and a procedure for propagating tempos that makes it possible to maintain the consistency of the event universe.

6.4 Tempo assignment

6.4.1 Explicit tempo marker

We use the syntactic form "/n" to indicate that the next event in a sequence is assigned an integer tempo *n*. For example,

/n e_i

means that

$$V(e_i) = n$$

This notation indicates infomally that the symbolic durations of objects following "/n" are divided by *n*.

Using empty sound-objects "_" it is possible to apply this notation to non-integer tempos. For example, the tempos of "a" and "b" in the sequence

/3 a _ _ _ /3 b _

are 3/4 and 3/2 respectively, which means, conversely, that the respective symbolic durations of "a" and "b" are 4/3 and 2/3.

6.4.2 Default assignment

The default tempo of the first event in a sequence is 1.

6.4.3 Tempo propagation after a divergence

If $e_0\ e_1$ is a sequence and there exist events e_2, …, e_k such that $e_1 = e_2 = \ldots = e_k$, then $V(e_1) = V(e_0)$ by default.

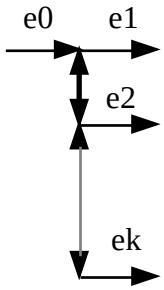


Fig.14 A divergence in a polymetric structure

This means that when "entering" the polymetric structure

$$\ldots\ e_0\ \{e_1\ldots,e_2\ldots,\ldots,e_k\ldots\}\ldots$$

the default tempo of the first argument is $V(e_0)$.

This means that when "entering" the polymetric structure

$$\ldots\ e_0\ \{e_1\ldots,\ e_2\ldots,\ \ldots,\ e_k\ldots\}\ldots$$

the default tempo of the first argument is $V(e_1) = V(e_0)$.

This rule will be illustrated in §7.3, in which the consistency condition will be used to determine the tempos of other sequences. Consistency may also impose a tempo $V(e_1) \neq V(e_0)$ whenever one of the arguments of the polymetric expression contains an explicit tempo marker.

6.4.4 Tempo propagation after a convergence

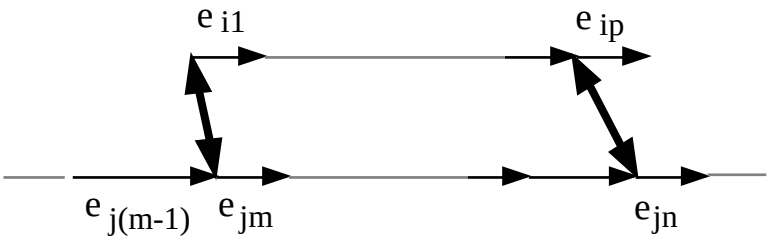


Fig.15 Divergence and convergence in a polymetric structure

$$V(e_{jn}) = V(e_{j(m-1)})$$

Informally, the tempo "after" a polymetric expression is the same one as "before" the expression.

6.4.5 Tempo propagation in a sequence

If $e_i\ e_j$ is a sequence and there is no *k* such that $e_j = e_k$, then

$$V(e_j) = V(e_i)$$

7. Interpreting and representing polymetric expressions

The interpretation algorithm has been introduced in [Bel 1990a] and explained in detail in [Bel 1990b, chapter VIII]. This and the following paragraph summarize its main features.

7.1 Polymetric representation of a complete universe

In §5.2 it was stated that in a complete event universe it is possible to assign each event a symbolic date. Therefore it is possible to represent the universe as a phase diagram whose columns are labelled with symbolic dates, as illustrated in §3.2. Conversely, a given phase diagram may lead to a polymetric expression. For example, Fig.7 is an interpretation of the diagram shown in §3.2.

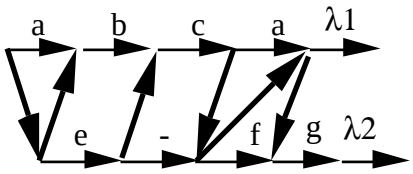


Fig.16 A complete event universe[11]

This universe does not fulfil the conditions yielding a polymetric representation, as indicated in [Bel 1990b:109-110]. Yet it can be replaced with equivalent ones fulfilling these conditions. For example, a silence may be appended to S_1, yielding the new phase diagram

t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	t11	t12	t13	t14
_	_	a	_	b	c	a	_	_	-	_	NIL	_	_
_	_	_	e	_	-	_	_	f	_	g	NIL	_	_

i.e. the complete polymetric expression:

{ a _ b c a _ _ - _ , _ e _ - _ _ f _ g }

[11] Note that this graph contains less information than the phase diagram. It would be unchanged if for instance the column labelled “t8” were deleted; additional columns containing only “_” could also be inserted at will.

Another way of representing the universe of Fig.16 as a polymetric expression consists of splitting the sound-object “f”. For this we use a concatenation symbol notated “&”, by way of which two segments of the same sound-object may appear in different arguments.[12] Thus we get for example

{ a _ b c a _ _ , _ e _ - _ _ f& } &f g

or equivalently:

a& { &a b c a _&, e _ - _ _ } { &a, f& } &f g

etc… Each expression implies a particular **surface structure**, here taken to mean “segmentation”. The latter one is illustrated in Fig.17:

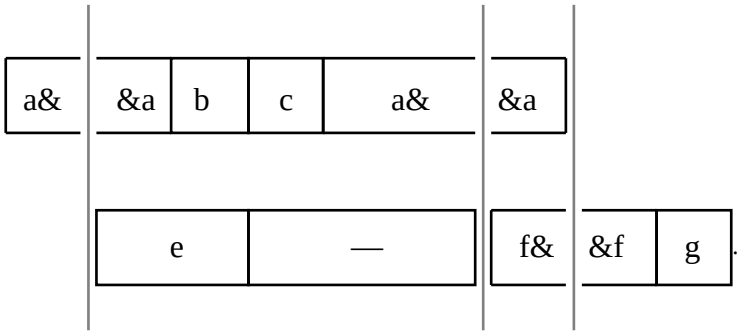


Fig.17 A segmentation of sound-objects yielding a polymetric expression

Since it is not possible to determine the surface structure of an arbitrary sound-object structure, the input of the polymetric interpretation algorithm is a well-formed polymetric expression, not a phase diagram.[13]

7.2 Complete polymetric expressions

Let Vt be a set of labels of sound-objects. A syntactic definition of complete polymetric expressions over Vt is proposed now.

[12] See for instance notes “B6” and “G#5” in the musical example of §9. The limitation of this notation (ambiguity) is discussed in [Bel 1990c].

[13] This was implied by the last sentence of §5.2.

Definition

1) $\forall\ x_i \in Vt$,

x_i , x_i&, &x_i and "_" are complete polymetric expressions of identical symbolic durations. (See §11 the definition of the symbolic duration of a single object x_i .)

2) Given a list of complete polymetric expressions $P_1,\ldots, P_{imax}$ of symbolic duration *n*,

$\forall i \in [1, imax-1]$,

$\{P_i\}$ and $\{P_1,\ldots, P_{i+1}\}$ are complete polymetric expressions of symbolic duration *n*.

3) If P_1 and P_2 are two complete polymetric expressions of respective symbolic durations n_1 and n_2 , then $P_1\ P_2$ is a complete polymetric expression of symbolic duration (n_1+n_2).

4) If P is a complete polymetric expression of symbolic duration *n*,

$\forall k \in N$ with $k \geq 1$,

/k P is a complete polymetric expression of symbolic duration $\frac{n}{k}$.

7.3 Interpreting a polymetric expression

The basic idea of the interpretation algorithm is illustrated here on simple examples of polymetric expressions.[14] In §5.1 the method for superimposing two sequences of different lengths was informally introduced. Given the incomplete expression

{ a b, c d e }

the tempo propagation rule in §6.4.3 yields a set of equivalent complete expressions

{a _ _ b _ _ , c _ d _ e _ }

{a _ _ _ _ _ b _ _ _ _ _ , c _ _ _ d _ _ _ e _ _ _ }

{a _ _ _ _ _ _ _ _ b _ _ _ _ _ _ _ _ , c _ _ _ _ _ d _ _ _ _ _ e _ _ _ _ _ }

etc…

with respective durations six, twelve, eighteen, that may be notated

/m {a _ _ b _ _ , c _ d _ e _ }

where *m* is an arbitrary strictly positive integer denoting the tempo of the polymetric structure. The symbolic duration of the whole structure is $\frac{6}{m}$.

[14] The interpretation of a nested expression is illustrated in [Bel 1990b], pp.123-6.

To determine *m* we use the rule in §6.4.3, i.e., informally, the default tempo of a polymetric structure is the one of its first argument taken separately. The first argument is

a b

with default tempo one (see rule in §6.4.2), which may be indicated by an explicit tempo marker:

{/1 a b, c d e }

According to the definition in §7.2, the symbolic duration of the structure must be two. Therefore m = 3 and the correct interpretation is:

/3 {a _ _ b _ _ , c _ d _ e _ }

The rule in §6.4.3 cannot be used if at least one argument in the expression contains an explicit tempo marker, e.g. in the expression

{a b {/3 a b c, d e}, f g h i j k}

Therefore, the interpretation algorithm performs the transformations

{a b /6 {a _ b _ c _ , d _ _ e _ _ }, f g h i j k}
{/6 a _ _ _ _ _ b _ _ _ _ _ {a _ b _ c _ , d _ _ e _ _ }, f g h i j k}
{/6 a _ _ _ _ _ b _ _ _ _ _ {a _ b _ c _ , d _ _ e _ _ }, /6 f _ _ g _ _ h _ _ i _ _ j _ _ k _ _ }

yielding the complete polymetric expression

/6 { a _ _ _ _ _ b _ _ _ _ _ {a _ b _ c _ , d _ _ e _ _ }, f _ _ g _ _ h _ _ i _ _ j _ _ k _ _ }

and a possible phase diagram:

a	_	_	_	_	_	b	_	_	_	_	_	a	_	b	_	c	_	NIL
_	_	_	_	_	_	_	_	_	_	_	_	d	_	_	e	_	_	NIL
f	_	_	g	_	_	h	_	_	i	_	_	j	_	_	k	_	_	NIL

7.4 Polymetric representation of a sequence

Introducing a string of empty objects as the first argument of a polymetric expression is a good method for suppressing explicit tempo markers in a sequence, as will be shown in the example:

a b c _ /3 d _ e

This sequence may be notated

a b c _ /3 {_ _ _ , d _ e}

which is equivalent to

a b c _ /3 {3, d _ e}

using the convention that any string of *n* empty objects "_" which does not follow a non-empty object may be replaced with integer *n* itself. The following transformation

a b c _ {3/3, d _ e}
a b c _ {1, d _ e}
a b c _ { _ , d _ e}

is valid even though it leads to an incomplete polymetric expression.

An advantage of this notation is visible when the expression is used in a grammar, as for example:

S —> /2 A /3 A
A —> a b c _ { 1 , d _ e}
… [other rules]

An intuitive meaning of this grammar is that the musical item denoted by variable "A" should be performed twice. The tempos of the successive occurrences will be two and three.

The unique production of this grammar is

/2 a b c _ { 1 , d _ e} /3 a b c _ { 1 , d _ e}

which the interpretation algorithm will transform as follows:

/6 a _ _ b _ _ c _ _ _ _ _ { 3 , d _ e} a _ b _ c _ _ _ { 2 , d _ e}
/6 a _ _ b _ _ c _ _ _ _ _ { 3 , d _ e} a _ b _ c _ _ _ /6 { 2 , d _ e}
/6 a _ _ b _ _ c _ _ _ _ _ { 3 , d _ e} a _ b _ c _ _ _ /18 { 6 , d _ _ _ e _ }
/6 a _ _ b _ _ c _ _ _ _ _ { 3 , d _ e} a _ b _ c _ _ _ /9 { 3 , d _ e}
/6 a _ _ b _ _ c _ _ _ _ _ d _ e a _ b _ c _ _ _ /9 d _ e

It can be seen that for example the symbolic duration of sound-object "d" is 2/6, i.e. 1/3, in its first occurrence, and 2/9 in its second occurrence. Thus the tempo has been multiplied, as expected, by 3/2. The same remark applies to all sound-objects.

7.5 Undetermined rests

It is possible to insert (one single) undetermined rest, notated "…", in an argument of a polymetric expression. The idea is that this rest may have different durations depending on other arguments of the expression, therefore it should be calculated by the interpretation algorithm. For example,

{/3 a … b, /2 c d}

should be interpreted

{/3 a - b, /2 c d}

since the duration of the structure is determined by the second argument of the expression.

Let us now consider

{a {b c, d e f}, /2 … g h i j }

in which the duration of the first argument is 3. Let *x* be the duration of the undetermined rest. We may write

3 = x + duration(g) + duration(h) + duration(i) + duration(j)

knowing that the durations of "g", "h", etc. are 1/2 because of the explicit tempo marker "/2". Therefore,

3 = x + 1/2 + 1/2 + 1/2 + 1/2

yields x = 1 = 2/2. The final interpretation is:

{/1 a {/1 b c, d e f}, /2 - _ g h i j }

In both preceding examples the undetermined duration was easy to compute because of explicit tempo markers. In general, let

$$\frac{pmax}{qmax}$$

be the symbolic duration of the polymetric structure, with pmax,qmax $\in$ N^* . Let *a* be the rank of the argument containing an undetermined rest, and

$$def[a] = \frac{p[a]}{q[a]}$$

the sum of the durations of all determined substructures in argument *a*

Since argument *a* does not contain an explicit tempo marker, its tempo *m* is undetermined. If we call

$$\frac{pgap[a]}{qgap[a]}$$

the duration of the undetermined rest (with pgap[a],qgap[a] $\in$ N), then the symbolic duration of argument *a* is:

$$\frac{pgap[a]}{qgap[a]} + \frac{p[a]}{m \,.\, q[a]}$$

This duration must be equal to the one of the polymetric structure itself. The minimum value of *m* that yields pgap[a] ≥ 0 is:

$$m = \text{Integer part of } \left(\frac{p[a].qmax}{q[a].pmax} \right)$$

If m = 0 the interpretation algorithm returns an error message "*not enough time for inserting a rest*". In other cases, the value calculated in this way is the one corresponding to the most "evident" solution. [Bel 1990b, p.119]

An undetermined rest is used in the example of §9.

8. Minimal and dilated notations of a polymetric structure

The primary motivation of "rescaling" polymetric expressions is to save memory and yield the simplest phase diagram of a sound-object structure. The **dilation ratio** defined in §8.2 is used to encode sound-objects and calculate their time-scale ratios as shown in §15.

8.1 Simplifying a sequence

Simplifying a sequence means trying to suppress empty objects in its notation. For example,

/6 a _ _ b _ _ c _ d _ is simplified /2 a b /3 c d

/5 a _ _ b _ _ c _ _ _ _ d _ _ _ is simplified /5 a _ _ b _ _ /1 c /5 d _ _ _

8.2 Dilation ratio

In the second example above it was not possible to suppress all empty objects. However, let us multiply all tempi by the same **scale factor** *s*, for instance s = 24. The "dilated" sequence becomes:

/120 a _ _ b _ _ /24 c /120 d _ _ _ which is simplified /40 a b /24 c /30 d

It is easy to figure out that the lowest acceptable value of *s* is twelve, yielding a **minimal notation**:

/20 a b /12 c /15 d

Let *Prod* be the lowest common multiple (LCM) of all tempi (20, 12, 15), i.e. sixty. The number of empty objects appended to each occurrence of "a" or "b" is:

$$\frac{60}{20} - 1 = 2$$

The same operation yields four empty objects after "c" and three after "d", hence the **dilated notation** of this sequence:

a _ _ b _ _ c _ _ _ _ d _ _ _

This notation is used for setting the columns of the phase diagram. Compared with the original notation, it multiplies all durations by a ratio of five that we call the **dilation ratio**. It is easy to prove that the dilation ratio is:

$$\text{Ratio} = \frac{\text{Prod}}{\text{s}}$$

8.3 Simplifying a polymetric expression

The preceding operations remain valid for a polymetric expression provided that the scale factor *s* is the same in all its arguments. If $s_1,\ldots, s_i,\ldots, s_{imax}$ are the scale factors of minimal notations of arguments $A_1,\ldots, A_i,\ldots, A_{imax}$, then any *s* which is a common multiple of the s_i is valid.

While interpretating a polymetric expression the algorithm attempts to find a minimal notation, therefore it changes scales to avoid generating empty objects "_". A minimal polymetric expression is one in which all arguments of the expression are minimal.

The interpretation of the following expression

{i {a b, c d e}, j k}

has been given as an illustration of the interpretation algorithm. [Bel 1990b, pp.124-6] The result is

{/6 i {/6 a b, /9 c d e}, /4 j k}

in which

Prod = LCM (6,9,4) = 36

and the dilation ratio is:

$$\text{Ratio} = \frac{\text{Prod}}{\text{s}} = \frac{36}{6} = 6$$

Therefore, the dilated expression is

{i _ _ _ _ _ {a _ _ _ _ _ b _ _ _ _ _ , c _ _ _ d _ _ _ e _ _ _ }, j _ _ _ _ _ _ _ _ k _ _ _ _ _ _ _ _ }

yielding for instance the phase diagram:

i	_	_	_	_	_	a	_	_	_	_	_	b	_	_	_	_	_	NIL
_	_	_	_	_	_	c	_	_	_	d	_	_	_	e	_	_	_	NIL
j	_	_	_	_	_	_	_	_	k	_	_	_	_	_	_	_	_	NIL

8.4 Polymetric structures with out-time objects

A sequence of out-time objects is viewed as a prefix of the sequence starting on the next sound-object (or the 'NIL' marker if there is no next object). For example,

{i {a <<h>> b, c d e}, j <<f>> <<g>> k <<l>>}

i.e. in dilated notation

{i_ _ _ _ _ {a_ _ _ _ _ <<h>> b_ _ _ _ _ ,c_ _ _ d_ _ _ e_ _ _}, j_ _ _ _ _ _ _ _ <<f>> <<g>> k_ _ _ _ _ _ _ _ <<l>>}

yields the following phase diagram:

i	_	_	_	_	_	a	_	_	_		_	_	b		_	_	_	_	_	NIL										
_	_	_	_	_	_	_	_	_	_		_	_	<<h>>	NIL																
_	_	_	_	_	_	c	_	_	_		d	_	_		_	e	_	_	_	NIL										
j	_	_	_	_	_	_	_	_	k		_	_	_		_	_	_	_	_	NIL										
_	_	_	_	_	_	_	_	_	<<f>>	NIL																				
_	_	_	_	_	_	_	_	_	<<g>>	NIL																				
_	_	_	_	_	_	_	_	_		_	_	_	_		_	_	_	_	_	<<l>>	NIL									

9. Polymetric interpretation of a conventional musical score

This example is taken from the *COMPOSE Tutorial and Cookbook* [Ames 1989:2]. See the musical score in Fig.18.

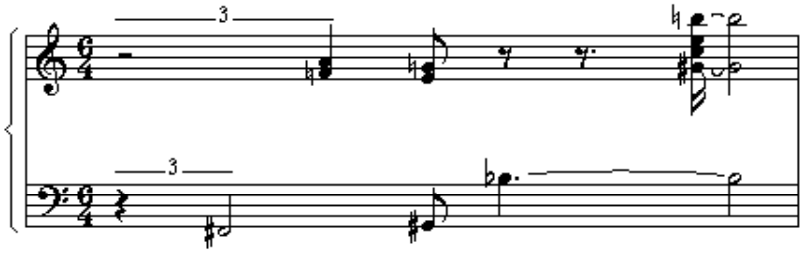


Fig.18 A musical score

This score may be represented with the pitch-versus-time[15] diagram of Fig.19.

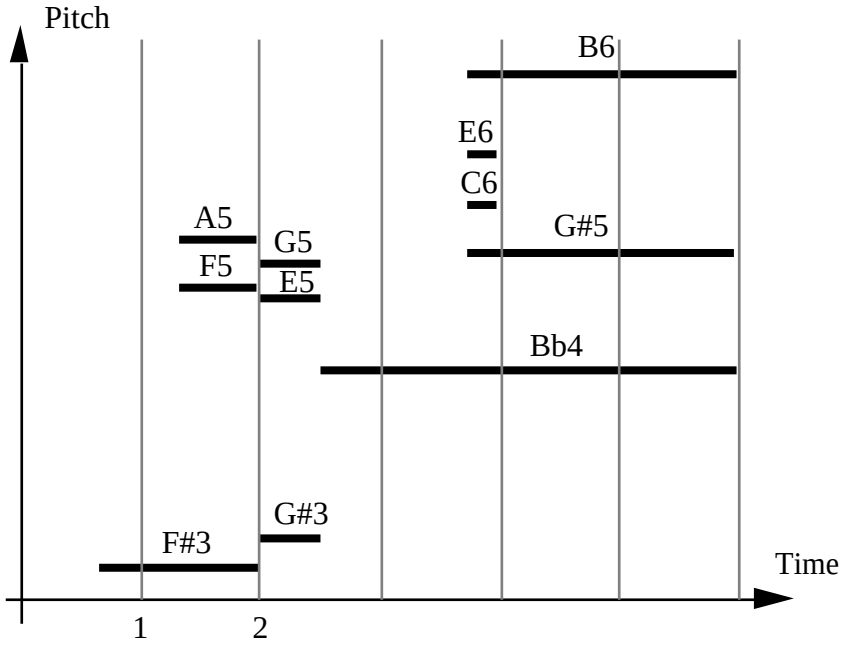


Fig.19 The pitch/time diagram

In COMPOSE the score is stored as an **event table** similar to Fig.20. [Ames 1989:3]

[15] In this paragraph we deal with symbolic, not physical, time, although the presumed structure of time is a metronomic pulse.

Period	Duration	Pitch
0.667	0.667	R *[rest]*
0.667	1.333	F#3
0.667	0.667	F5, A5
0.5	0.5	G#3, E5, G5
1.25	3.5	Bb4
0	2.25	C6, E6
2.25	0.25	G#5, B6

Fig.20 The event table

In Ame's terminology, "period" stands for the time elapsed from the on-setting of an event (note, rest or chord) to the on-setting of the next event. Both periods and durations are measured with a time unit which is the duration of a quarter note.

If the piece, or a variation of it, has been generated by a grammar, its deep structure should be visible at some level of the representation. Here we propose one of the many possible structural analyses of Ame's example:

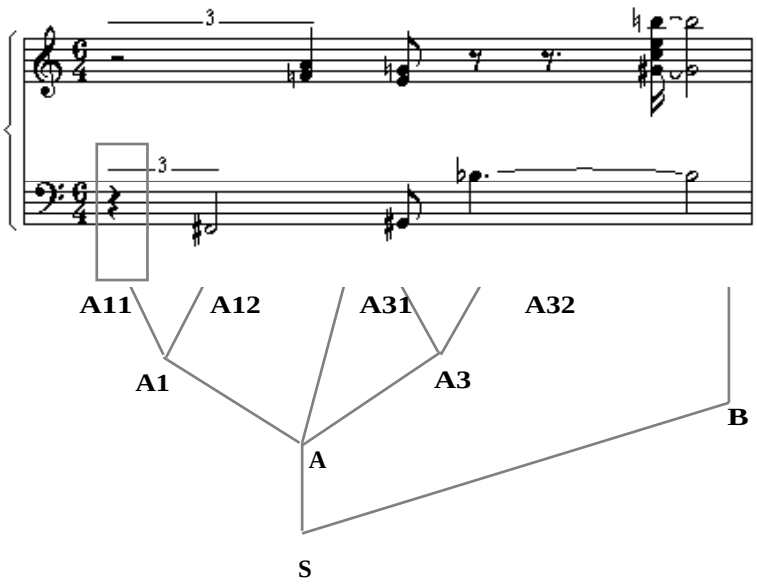


Fig.21 A possible hierarchy

As suggested by this tree, some rests (e.g., A11) may be viewed as part of the structure while others are just functioning as delays.

The tree-structure in Fig.21 could be the parsing tree of a production by the context-free grammar:

```
S     —> {A , … B}
A     —> {2, A1, - - A2} {4, A3}
A1    —> A11 A12              or equivalently   A1   —>   {A11  A12}
A3    —> A31 A32              or                A3   —>   {A31  A32}
A11   —> -                    or                A11  —>   {-}
A12   —> {2, F#3}
A2    —> {F5, A5}
A31   —> {1/2, G#3, E5, G5}
A32   —> {3/2, Bb4&} {2, &Bb4}
B     —> {1/4, G#5&, C6, E6, B6&} {2, &G#5, &B6}
… etc.        [Other rules]
```

Using only the rules listed above yields the unique production

{{2, {- {2, F#3}}, 2 {F5, A5}} {4, {1/2, G#3, E5, G5} {3/2, Bb4&} {2, &Bb4}}, … {1/4, G#5&, C6, E6, B6&} {2, &G#5, B6}}

displaying the surface structure of this particular production (see §7.1).

In this expression, “G#5&” and “&G#5” denote two segments of the same object “G#5” (see §7.1). The expression contains an undetermined rest “…” (see §7.5) produced by the first rule in the grammar.

Time information is redundant in this polymetric expression, yet it is consistent. The interpretation algorithm yields the complete polymetric expression which is displayed

{{{-_ _ _ _ _ _ _ {F#3_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ }} , -_ _ _ _ _ _ _ -_ _ _ _ _ _ _ {F5_ _ _ _ _ _ _ , A5_ _ _ _ _ _ _ }}{{G#3_ _ _ _ _ , E5_ _ _ _ _ , G5_ _ _ _ _ }{Bb4_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ &}{&Bb4_ }} , -_ _ {G#5_ _ & , C6_ _ , E6_ _ , B6_ _ &}{&G#5_ , &B6_ }}

while it is stored in minimal notation by BP2:

{{{/18 -, {/9 F#3}}, /18 - - {F5, A5}}{{/24 G#3, E5, G5} {/8 Bb4&} {/6 &Bb4}}, 5/16 {/48 G#5&, C6, E6, B6&} {/6 &G#5, B6}}

Let us for instance examine why it is acceptable to write a rule like

```
A12   —>      {2, F#3}
```

knowing that, although “F#3” is a half-note on the score, its actual duration is 4/3 units (i.e. 1.333 as shown on the second line of the event table, Fig.21). The additional information allowing a correct interpretation is contained in rules:

```
A     —>      {2, A1, - - A2} {4, A3}
A1    —>      A11  A12
```

The first rule indicates that the piece is made of two sections, the first one lasting two units of (symbolic) time and the second one 4 units. In the first section, A1 has a two unit duration while A2 has a 2/3 unit duration like each of the two silences “-”. Then we derive:

A1 => A11 A12 => A11 {2, F#3} => - {2, F#3}

In the last expression, "-" and "F#3" share 1/3 and 2/3 of the duration of the sequence respectively. Therefore the actual duration of "F#3" is:

$$\frac{2}{3} \text{ x } 2 \; = \; \frac{4}{3}$$

Representing the piece as a tree-structure (an outcome of some musicological analysis) makes it easy to relate it to a set of acceptable "variations", i.e. a language generated by a grammar. The initial grammar can be modified to generate this set rather than a unique piece. Let us for example decide that in another variation "F#3" should be "slightly longer". We may write:

A12 —> {3, F#3}

so that A1 is now derived as "- {3, F#3}" in which "-" and "F#3" share 1/4 and 3/4 of the duration of the structure respectively. Now the duration of "F#3" is:

$$\frac{3}{4} \text{ x } 2 \; = \; \frac{3}{2}$$

The polymetric interpretation algorithm automatically readjusted the duration of the rest preceding "F#3". To enter the same modification in the event table Fig.21 it would be necessary to recalculate both the duration of the rest (now 0.5) and the period of "F#3" (now 0.833). Understandably, the interpretation algorithm takes care of such modifications.

A system manipulating a grammar and interpreting its productions as polymetric expressions is therefore aware of the "structure of the piece", by which we mean constraints on synchronization yielding information on durations and start/clip times. Due to the structure of the piece indicated in the grammar, the statement "*F#3 should be longer*" means that its start time must be earlier while its clip time remains synchronized with the clip time of chord {F5,A5}. Therefore, a limitation of the event-list representation is that it makes synchronization explicit only on start-times.

10. The time-setting problem — an informal introduction

In the rest of this paper we deal with the problem of instanciating the sound-objects of a complete polymetric structure. Informally, instanciating a sound-object means dispatching to the sound processor all the messages that are listed in its prototype (see appendix).

A naive interpretation of sequences of sound-objects would be to arrange all corresponding time intervals in a strictly sequential way. Duthen and Stroppa [1990] have suggested a more abstract approach, starting from the assumption that any sound-object may possess one or several time points playing a particular role, e.g. a climax. These points are called **time pivots**. Further they suggest to construct sound structures using a set of synchronization rules. When two objects are synchronized, two pivots — the ones selected in that particular context — are superimposed while other pivots may be used to infer the new pivots of the compound object. Fig.22 represents two sound-objects with respective pivots (A, B, C, D) and (X, Y, Z) being used to build a compound object, assuming that there is a rule saying that C and Y should coincide. Other rules assign

pivots (I, J, K, L) to the compound object. Note that some of the new pivots, like L, may not coincide exactly with lower-level pivots.

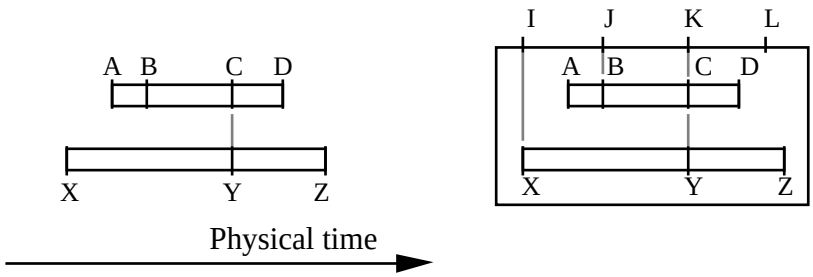


Fig.22 Synchronizing two sound-objects *à la* Stroppa

This approach is attractive but it is hard to implement if the formalism of synchronization rules remains too general. Moreover, in our approach, synchronization is primarily a matter of symbolic time: the partial ordering of objects in a polymetric structure. Therefore we retained a simplified version of Stroppa's idea, assigning each object one single pivot.

Let us for instance consider a polymetric structure {S1,S2,S3} derived as

{a _ b c d _ e , a _ f _ g h _ , j i _ a _ i _ }

yielding the phase diagram:

a	_	b	c	d	_	e	NIL
a	_	f	_	g	h	_	NIL
j	i	_	a	_	i	_	NIL

Suppose that all sound-object prototypes labelled "a", "b", "c", …, "i" are defined — they may have been recorded from a musical instrument as suggested in §2. As will be shown below (§12), the definition of each object contains the relative location of its pivot and metrical properties allowing the calculation of its "time-scale ratio" — informally, a factor adjusting the duration of the sound-object to the current tempo of performance.

The following is a graphic representation of a possible instance of this polymetric structure:

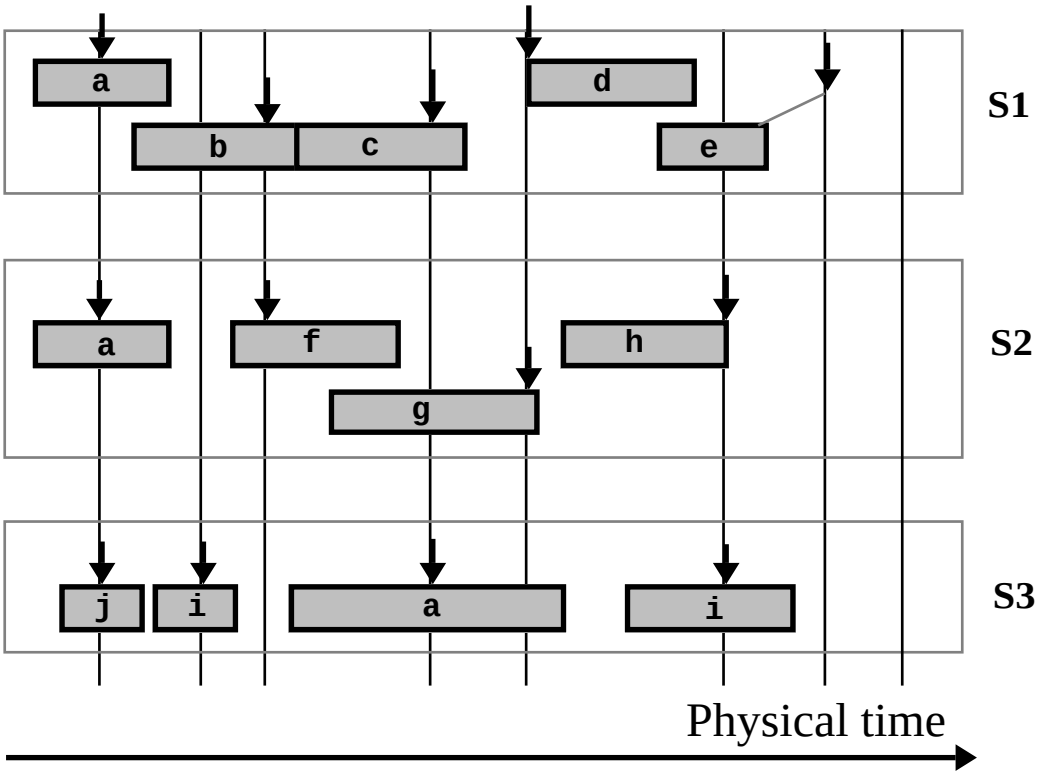


Fig.23 A structure of sound-objects

The structure of time is an irregular pulsation represented with vertical lines (time **streaks**). The time-span interval of each sound-object is shown as a rectangle with arbitrary vertical width and position. These positions have been chosen to separate objects on the graphic: it is clear for example that “c”, “f”, “g” and “a” have overlapping time-span intervals between the third and fourth streaks. Lengths of rectangles represent the physical durations of sound-objects. Out-time objects are not shown in these examples as they would appear as vertical segments.

Vertical arrows indicate time pivots. As shown with object “e”, the pivot is not necessarily a time point within the time-span interval of the sound-object.

This graphic represents the **default positioning** of objects, with pivots located exactly on time streaks. Although it is reasonable that instances of “c”, “f” and “a” are overlapping between the third and fourth streaks since they belong to distinct sequences which are performed simultaneously, it may not be acceptable that “f” overlaps “g” in a single sequence S2; the same with “d” and “e” in sequence S1. For similar reasons, it may not be acceptable that the time-span intervals of “j” and “i” are disjoint in sequence S3 while no silence is shown in the symbolic representation. The philosophy of our approach is to bind these topological constraints to properties of objects (see §13) instead of imbedding all the information in a symbolic representation. Therefore, **the symbolic representation contains no more information than the ordering of pivots**.

The topological situations of time-span intervals depend on the structure of time. For example, although the two instances of “i” in S3 have identical symbolic durations, their physical durations (i.e. their time-scale ratios) are different because the “beat offsets” (i.e. the duration from one streak to the next) are different.

How could one deal with a constraint such as <<*the end of sound-object "f" may not overlap another sound-object in the same sequence*>> ? If object "g" is relocatable then it may be delayed (shifted to the right) until the constraint is satisfied. We call this a **local drift** of the object (see §12.1). Yet the end of "g" will also overlap the beginning of "h". Assume that this too is not acceptable and "h" is not relocatable. One should therefore look for another solution, for example truncate the beginning of "h" (see §14). If this and other solutions are not acceptable then one may try to shift "f" to the left or to truncate its end. In the first case it might be necessary to shift or truncate "a" as well.

So far we mentioned a kind of constraint propagation within one single sequence. In the time-setting algorithm the three sequences are considered in order S1, S2, S3. Suppose that the default positioning of objects in S1 satisfies all constraints and no solution has been found to avoid the overlapping of "f" and "g" in S2. Another option is to envisage a **global drift** to the right of all objects following "f" in S2. The global drift is notated Δ on Fig.24. All time streaks following the third one are delayed (see dotted vertical lines).

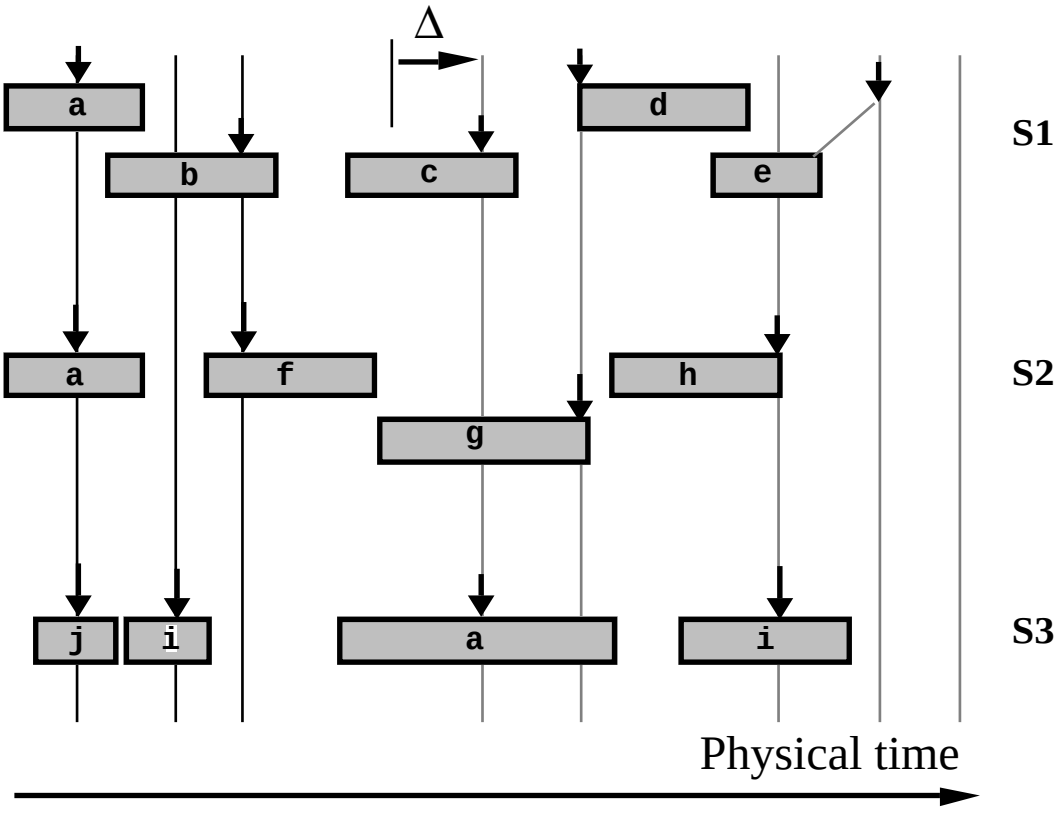


Fig.24 A structure using global drift

This solution is labelled "Break tempo" because its effect is similar to the the *organum* in conventional music notation. Although the global drift increases the delay between the third and fourth streaks, the physical durations of sound objects are not changed because their time-scale ratios have been calculated beforehand.

Now the positioning of objects in S2 is acceptable, but it might have become inacceptable in S1: there may be a property of “b” or “c” saying that their time-span intervals cannot be disjoint, so that “c” could be shifted to the left, etc. Evidently, whenever a global drift is decided the algorithm must start again from the first sequence.

The process of locating — i.e. “instantiating” — sound-objects, as illustrated in this example, is the task of the **time-setting algorithm** which will be partly described in §16.

11. Encoding a polymetric structure of sound-objects

The essential data structures used by the time-setting algorithm are introduced here. In §3.1 we indicated that the structure of time Φ is a strictly increasing function mapping the set of symbolic dates $\{t_i\}$ to physical time. Therefore the structure of time is encoded as an increasing list of physical dates $\{T(i): i = 1, \ldots, imax\}$ such that $T(i) = \Phi(t_i)$.

In striated time all T(i) are given as input data. In smooth time they are calculated only once the sound-objects have been located in the first sequence.

11.1 Encoding structures of sound-objects

A polymetric structure of sound-objects like

/3 ab { cde, ab } cd

is represented with the following phase diagram (dilation ratio: Ratio = 6, see §8.2):

a _ b _ a _ _ b _ _ c _ d _ NIL
_ _ _ _ c _ d _ e _ NIL

Symbols “a”, “b”, …, designate sound-objects E_k, in which *k* is an arbitrary index ($k \geq -1$). Symbol “_” designates the empty sound-object E_0. “NIL” is mapped to a virtual object E_{-1} delimitating the end of each sequence.

The strictly positive values of *k* in this structure may be for instance:

1 **2** **3** **4** **5** **6**
a _ b _ a _ _ b _ _ c _ d _
c _ d _ e _
7 **8** **9**

All *k*'s are contained in a bidimensional array called the **phase table** Seq(nseq,i):

Phase table *Seq(nseq,i)*

i =	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
nseq = 1	1	0	2	0	3	0	0	4	0	0	5	0	6	0	-1
nseq = 2	0	0	0	0	7	0	8	0	9	0	-1	0	0	0	0

E_k is the k-th sound-object, with k = Seq(nseq,i). The column index *i* is the **rank** of object E_k in its sequence. If we call *inext* the rank of the next non-empty sound-object or "NIL" marker in the sequence, the **symbolic duration** of sound-object E_k is:

$$d(k) = \frac{inext - i}{Ratio}$$

in which *Ratio* is the dilation ratio (see §8.2).

T(i) is the date of the **reference streak** of objects with rank *i* in the structure. If T(i) and the complete polymetric expression are known, it is possible to compute **performance parameters**, thereby determining the dates of all elementary messages contained in sound-object prototypes (see appendix). These parameters are listed in an array, the **instance table**, for example:

Instance table

k	1	2	3	4	5	6	7	8	9
j = Obj(k)	1	2	1	2	3	4	3	4	5
d(k)	1/3	1/3	1/2	1/2	1/3	1/3	1/3	1/3	1/3
t1(k), t2(k), α(k), etc…	…	…	…						

In this table, d(k) is the symbolic duration of E_k. Parameters t1(k) and t2(k) are the physical **start/clip date**s of the k-th sound-object, and α(k) its time-scale ratio (see §12.2). Obj(k) is a pointer allowing object identification in the **symbol table**:

Symbol table

j =	1	2	3	4	5
symbol	a	b	c	d	e

Each E_k (with $k \geq 0$) may be seen as an **instance** of an **object prototype** Ep_j with j = Obj(k). If k = 0 the object is empty (labelled "_"), therefore Ep_0 is the empty-object prototype. Object prototypes are formally defined in §1 of the appendix.

11.2 Encoding out-time objects

Let <<f>> and <<g>> designate out-time objects. The polymetric expression

/3 ab { c <<f>> de, a <<g>> <<f>> b } cd

is interpreted

/3 ab { /3 c <<f>> de, /2 a <<g>> <<f>> b } /3 cd

which yields the dilated expression:

/6 a _ b _ { c_ <<f>> d _ e _ , a _ _ <<g>> <<f>> b _ _ } c _ d _

with Ratio = 1. The corresponding tables are:

Symbol table

j =	1	2	3	4	5	6	7
symbol	a	b	c	d	e	f	g

Instance table

k =	1	2	3	4	5	6	7	8	9	**10**	**11**	**12**
j = Obj(k)	1	2	1	2	3	4	3	4	5	6	7	6
d(k)	1/3	1/3	1/2	1/2	1/3	1/3	1/3	1/3	1/3	**0**	**0**	**0**
t1(k), t2(k), α(k), etc…	…	…	…									

Phase table *Seq(nseq,i)*

i =	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
nseq = 1	1	0	2	0	3	0	0	4	0	0	5	0	6	0	-1
nseq = 2	0	0	0	0	7	0	8	0	9	0	-1	0	0	0	0
nseq = 3	0	0	0	0	0	0	**10**	**11**	-1	0	0	0	0	0	0
nseq = 4	0	0	0	0	0	0	0	**12**	-1	0	0	0	0	0	0

12. Metrical properties of non-empty sound-objects

In this and the next three paragraphs, **sonic properties** and **transformations** of sound-objects are introduced. These are used by the time-setting algorithm.

A **sonic property** P of a sound-object E_k is a predicate P(j) defined by the corresponding sound-object prototype Ep_j, with j = Obj(k). In this way, all properties of a sound-object are “inherited” from the (unique) sound-object prototype bearing the same label.

12.1 Time pivot

Once a sound-object E_k (given k = Seq(nseq,i)) has been instantiated by the time-setting algorithm, its time pivot is located at physical date:

$$T(i) + \Delta(i) + \delta(k)$$

in which $\delta(k)$ is the **local drift** of the object and $\Delta(i)$ the **global drift** of its reference streak. If $\delta(k) \neq 0$ the object has been **relocated**. Relocation is only allowed for objects with a property notated "Reloc" (see §12.3.1).

An object which is assigned a pivot is called a **striated sound-object**. An object which has no definite pivot is a **smooth sound-object**. In reality, a smooth object is assigned a pivot at the beginning of its time-span interval and declared with property Reloc, so that the pivot location is only a default value.

Physical dates tmin(j) and tmax(j) are the respective **start/clip dates** of the object prototype Ep_j, taking its pivot as the time origin, while Dur(j) = tmax(j)-tmin(j) is its **physical duration** (see appendix, §1).

The following is a (non-limitative) list of properties relevant to the positioning of striated objects.

12.1.1 Pivot in the beginning (PivBeg), in the end (PivEnd)

The pivot coincides with the first (resp. last) message of the sound-object. Therefore,

Case PivBeg: tmin(j) = 0 , tmax(j) = Dur(j)
Case PivEnd: tmin(j) = -Dur(j) , tmax(j) = 0

12.1.2 Pivot centered (PivCent)

The pivot is exactly in the middle of the time-span interval of Ep_j, hence:

$$\text{tmin(j)} = - \text{tmax(j)} = - \frac{\text{Dur(j)}}{2}$$

12.1.3 General case (PivSpec)[16]

The pivot is at some physical date t0 in reference to the first message. Therefore:

$$\text{tmin(j)} = - t0 , \quad \text{tmax(j)} = \text{Dur(j)} - t0$$

Fig.25 shows a sequence of three objects labelled "a", "e", "d" with respective properties PivBeg, PivCent, PivEnd, and physical durations 1.3 s., 0.8 s. and 0.4 s., arranged on a metronomic structure of time with period 0.5 s.

[16] Other properties relative to pivots may be found in [Bel 1990c].

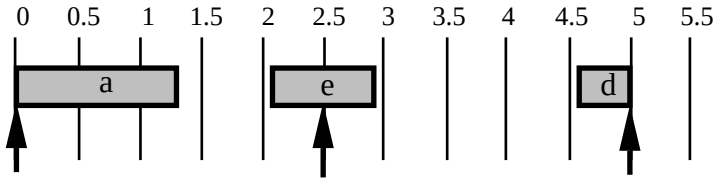


Fig.25 Three typical objects and their pivots

12.2 Scaling objects in time

The **physical duration** of a non-empty sound-object is generally not the same as the one of its prototype. The actual physical duration of E_k is

$$\alpha(k) \,.\, Dur(j)$$

where $\alpha(k)$ is the **time-scale ratio** and Dur(j) the duration of sound-object prototype Ep_j, given j = Obj(k). Methods for calculating $\alpha(k)$ are proposed in §15. The way $\alpha(k)$ is taken into account for calculating the dates of elementary messages is explained in §3 of the appendix.

Certain ranges of values of $\alpha(k)$ may not be acceptable. For instance, some objects should never be streched while others should not be contracted. The following properties define acceptable ranges of $\alpha(k)$.

Elasticity (FixScale, OkRescale, OkExpand, OkCompress)

Case OkRescale: any value of $\alpha(k)$ is acceptable.
Case FixScale: $\alpha(k) = 1$.
Case OkExpand: $\alpha(k) \geq 1$ is acceptable.
Case OkCompress: $\alpha(k) \leq 1$ is acceptable.

If the value calculated for $\alpha(k)$ (see §15) is not in an acceptable range, then the value $\alpha(k) = 1$ is imposed.

12.3 Relocating objects or streaks

These properties are relevant to the **local drift** $\delta(k)$ of objects and the **global drift** $\Delta(i)$ of streaks.

12.3.1 Relocatability (Reloc)

An object is **relocatable** if its local drift $\delta(k)$ is allowed to take positive or negative values. A classical example of relocatable objects on a metronomic structure of time is the performance of *rubato*.

12.3.2 Break of tempo (BrkTempo)

There is a **break of tempo** on sound-object E_k (given k = Seq(nseq,i) with k > 0) if all streaks following the reference streak of the object are delayed, i.e. the global drift $\Delta(ii) > \Delta(i)$ for all ii > i. An object allowed to break tempo has property BrkTempo.

13. Topological properties of non-empty sound-objects

These properties are used to check whether or not topological configurations of time-span intervals are acceptable in a sequence (see informal example in §10).

13.1 Covering the beginning (OverBeg), the end (OverEnd)

The OverBeg property means that the beginning of the time-span interval of an object may be covered by other objects in the same sequence. OverEnd is defined similarly.

13.2 Continuity in the beginning (ContBeg), in the end (ContEnd)

A sound-object has property ContBeg if its time-span interval must be connected with (or overlap) the time-span interval of one of the preceding objects in the sequence. ContEnd is defined similarly: the time-span interval must be connected with (or overlap) the one of any object following it in the sequence.

14. Reshaping sound-objects

Many transformations of sound-objects could be imagined, e.g. a non-linear "distorsion" of the **local time** of an object (see $\text{LocalTime}_k(t)$ in §3 of appendix), or even adding/suppressing messages. The only transformation we consider here is truncating the beginning or the end of a sound-object. Properties allowing this are labelled "TruncBeg" and "TruncEnd" respectively.

A variable notated "TrBeg(k)" indicates the amount of physical time by which the beginning of sound-object E_k has been truncated. Similarly, TrEnd(k) is relative to the truncating of its end. Evidently,

$0 \leq \text{TrBeg}(k)$, $0 \leq \text{TrEnd}(k)$, and $\text{TrBeg}(k) + \text{TrEnd}(k) <$ physical duration of E_k .

Truncating a sound-object does not mean that all elementary messages contained in the truncated part have been deleted. Some of them are relocated at the new start/clip dates of the object, as shown in §3 of the appendix.

15. Calculating time-scale ratios $\alpha(k)$

The time-scale ratio $\alpha(k)$ of a sound-object E_k depends on its symbolic duration d(k), its nature (smooth or striated) and the structure of time (smooth or striated). This paragraph introduces rules for calculating $\alpha(k)$.

A sound-object prototype Ep_j may be defined in reference to a metronome beat, the period of which is notated Tref(j). If no metronomic reference is used then conventionally Tref(j) = 0.

15.1 Calculating α(k) in smooth time

In smooth time it possible to imagine that there is a clock with period Tclock measuring physical durations. This case is called **measured smooth time**. If no such clock is available we always take:

$$\alpha(k) = d(k)$$

where d(k) is the symbolic duration of E_k (see §11.1) . The duration of sound-object prototype Ep_j is Dur(j). Empty objects "_" are mapped to prototype Ep_0 such that Dur(0) = 0.

In measured smooth time, α(k) is calculated as follows:

Object type	Tref(j)	Dur(j)	α(k)
striated	> 0	= 0	d(k) . Tclock / Tref(j)
striated	> 0	> 0	d(k) . Tclock / Tref(j)
smooth	= 0	> 0	d(k) . Tclock / Dur(j)
smooth	= 0	= 0	d(k)

Using this table, α(k) cannot be calculated in silences, i.e. non-empty sound-objects conventionally notated "-" for which both Tref(j) and Dur(j) are undetermined. A physical duration is therefore assigned to a silence in reference to the "local period". Let E_{kprec} be the non-empty sound-object immediately preceding E_k in the sequence. The **local period** is:

$$P = \frac{t2(kprec) - t1(kprec)}{d(kprec)}$$

in which t1(kprec) and t2(kprec) are the respective start/clip physical dates of E_{kprec}, and d(kprec) its symbolic duration. The physical duration of silence E_k will therefore be:

$$P.\ d(k)$$

If a sequence of sound-objects starts with a silence in measured smooth time, the first silence is ignored because *kprec* would not be defined.

15.2 Calculating α(k) in striated time

15.2.1 Silences

For all silences, α(k) = 0 in striated time.

15.2.2 Striated sound-objects

The value proposed for $\alpha(k)$ is:

If $d(k) > 0$,

$$\alpha(k) = \frac{T(inext) - T(i)}{Tref(j)}, \quad \text{with } j = Obj(k) \text{ and } k = Seq(ligne, i)$$

where *inext* is the rank of the next non-empty sound-object or the "NIL" marker in the sequence,

or, if $d(k) = 0$ (case of out-time objects),

$$\alpha(k) = 0$$

15.2.3 Smooth sound-objects

To calculate $\alpha(k)$, Tref(j) is replaced with Dur(j). This yields:

If $d(k) > 0$,

$$\alpha(k) = \frac{T(inext) - T(i)}{Dur(j)}, \quad \text{with } j = Obj(k) \text{ and } k = Seq(ligne, i)$$

or, if $d(k) = 0$ (case of out-time objects),

$$\alpha(k) = 0$$

The physical duration (see §12.2) of a smooth object E_k is therefore

$$\alpha(k) \,.\, Dur(j) = T(inext) - T(i)$$

for a sound-object, and zero for an out-time object.

16. The time-setting algorithm

Instantiating (i.e. **setting in time**) a sound-object structure means determining the **performance parameters** $\alpha(k)$, $\delta(k)$, t1(k), t2(k), TrBeg(k), TrEnd(k) and $\Delta(i)$ for all sound-objects E_k such that k = Seq(nseq,i) with i = 1,…,imax and nseq = 1,…,nmax. Variable *i* is the index of the reference streak of sound-object E_k. This instantiation is accomplished by the time-setting algorithm which was informally introduced in §10.

Results on the correctness and complexity of this algorithm have already been published in [Bel 1990b, chapter IX]. In this paragraph, pseudo-code descriptions of the main procedures are given with enough details for the design of an implementation in procedural programming languages.

The space complexity can easily be discussed as the dimensions of all arrays necessary to the computation ar indicated. In actual implementations arrays should be replaced with data-structures best fit to the programming environment.

16.1 Main loop

```
Procedure Time_set

Global variables:
  nmax = maximum number of sequences
  imax = maximum length of a sequence
  kmax = maximum number of objects in the structure
  Seq[nmax,imax],Δ[imax],T[imax], Obj[kmax],t1[kmax],t2[kmax],
    t"1[kmax],t"2[kmax],TrBeg[kmax], TrEnd[kmax],α[kmax],δ[kmax]

Input: Seq[],Obj[],T[]

Output: T[],t1[],t2[],TrBeg[],TrEnd[],α[],δ[]

Begin
  For (i = 1; i ≤ imax)
  |    Δ[i] <— 0;
  |    If(nature_of_time = smooth)
  |    then
  |    |    T[i] <— 0;  /* T[i] is not known in smooth time. */
  |    Endif
  |    i <— i + 1;
  Endfor
  Calculate_alpha;

Again:
  For (nseq = 1; nseq ≤ nmax)
  |    Fix(nseq);  /* Calculate t1[], t2[] */
  |    If(Locate(nseq,nature_of_time) = failed)
  |    then
  |    |    Display "Can't set time";
  |    |    /* Here we can restart the algorithm releasing constraints
  |    |       ContBeg, ContEnd, OverBeg or OverEnd. */
  |    |    Stop;
  |    Endif
  |    If((nature_of_time = smooth) and (nseq = 1))
  |    then
  |    |    Interpolate_streaks;
  |    Endif
  |    BTflag <— false;
  |    For (i = 1; i ≤ imax)
  |    |    T[i] <— T[i] + Δ[i];
  |    |    If (Δ[i] ≠ 0)
  |    |    then
  |    |    |    BTflag <— true;
  |    |    Endif
  |    |    Δ[i] <— 0;
  |    |    i <— i + 1;
  |    Endfor
  |    If (BTflag and (nseq > 1)) go to Again;
  |    nseq <— nseq + 1;
  Endfor
End
```

Procedure `Fix()` is invoked to calculate the default start/clip dates t1(k) and t2(k) of each object as follows:

```
Procedure Fix(nseq)

Begin
  For (i = 1, i ≤ imax)
  |    k <— Seq[nseq,i];
  |    j <— Obj[k];
  |    If(j = 1)   /* Obj[k] is a silence */
  |    then
  |    |    …;     /* Compute with local period, etc. (see §15.1) */
  |    else
  |    |    t1[k] <— α[k].tmin[j] + T[i];
  |    |    t2[k] <— α[k].tmax[j] + T[i];
  |    Endif
  |    i <— i + 1;
  Endfor
End
```

The `Locate()` function called in the main loop relocates the sound-objects found on each line of the phase diagram. When it is successful, the following operations are necessary before the next sequence is processed:

(1) If time is smooth and nseq = 1, values of Δ(i) must be calculated for all empty objects with rank *i*. This is done by interpolation, as illustrated in §18.3.1.

(2) T(i) must be updated:

```
T[i]  <—  T[i] + Δ[i]
Δ[i]  <—  0
```

(3) If nseq ≠ 1 and Δ(i) ≠ 0 for some *i*, it means that a correction of type {Break tempo} has been done. As indicated in the example of §10, the algorithm must be restarted with the new values of T(i) while current values of α(k) remain unchanged. Therefore the main loop is interrupted by a `go to Again`.

16.2 `Locate()` function — flowchart

Each sequence is scanned “from left to right” (i.e. increasing dates). The following flowchart shows the detailed operation.

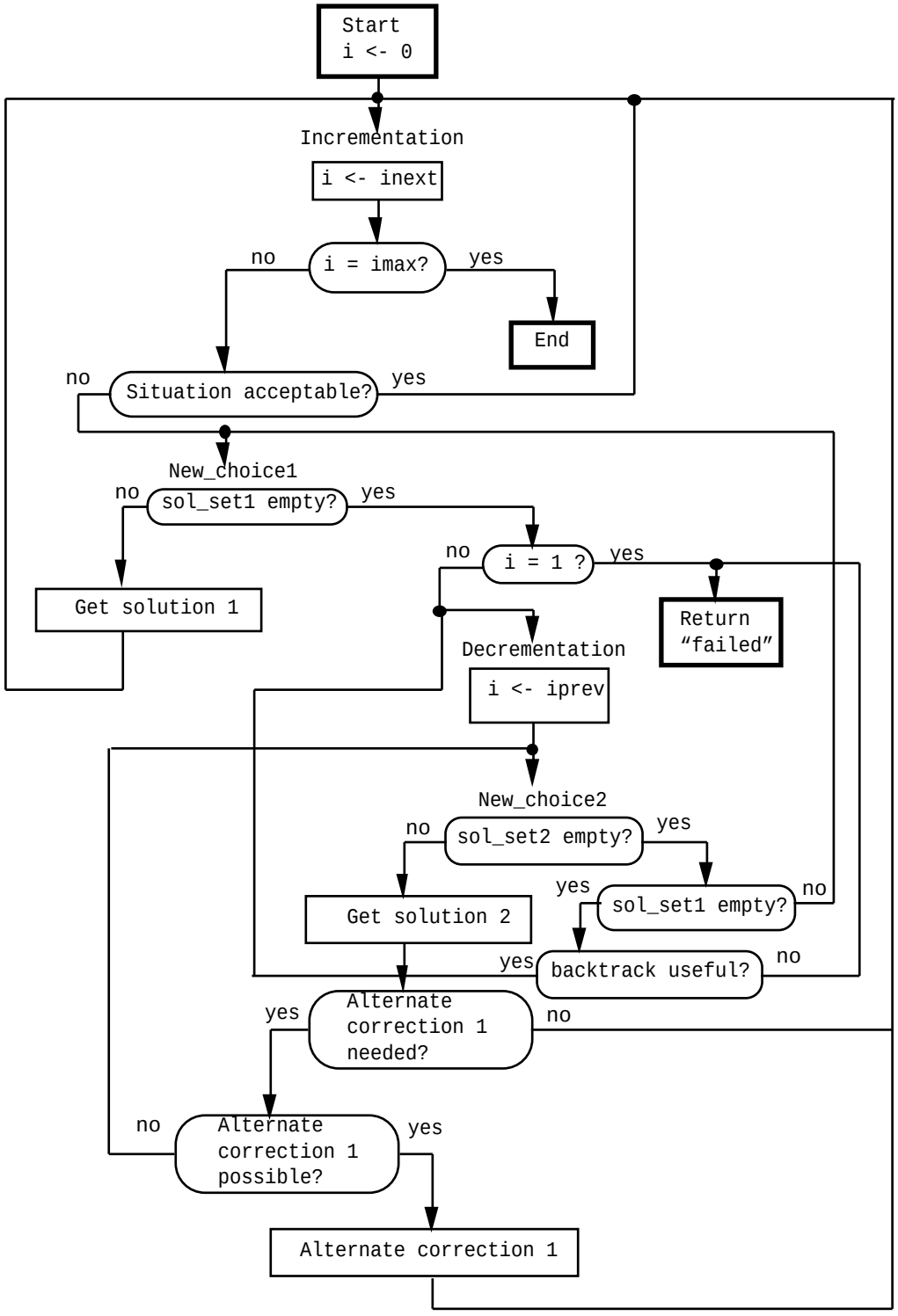


Fig.26 The Locate() flowchart

The main test in this function is “Situation acceptable?” (see §16.2). If the situation of E_k does not fulfill topological constraints, then a first solution set `sol_set1` is calculated. This set contains possible corrections of E_k aimed at changing its start date t1(k). Its clip date t2(k) may also incidently be changed in this process. Each time the program jumps at `New_choice1`, a solution is tried and deleted from the set. This

correction is called **correction 1**. If `sol_set1` is empty, the situation of one of the preceding sound-objects must be revised (see `Decrementation`). Another solution set `sol_set2` is calculated, yielding **correction 2**, i.e. mainly a modification of the clip date t2(k).

To facilitate backtracking, `sol_set1` is stored as an array indexed on *i*.

16.3 The `Locate()` function

While positioning objects temporary values are used for the start/clip dates t1(k) and t2(k). These are notated t'1(i), t"1(i), t'2(i) and t"2(i), given k = Seq(line,i).[17]

We use index 1 for variables dealing with corrections "to the left" of E_k (constraints on t1(k)), whereas index 2 indicates corrections "to the right" (constraints on t2(k)). Thus, $\delta 1(i)$ and $\delta 2(i)$ denote temporary local drifts caused by corrections 1 and 2, while the final value of the local drift is $\delta(k) = \delta 1(i) + \delta 2(i)$.

$d\Delta 0(i)$ is the temporary value of the global drift $\Delta(i)$ of the i-th streak when taking into account the temporary global drifts of the preceding streaks. $d\Delta 1(i)$ and $d\Delta 2(i)$ are the temporary changes of the global drift $\Delta(i)$ of the i-th streak caused by corrections 1 and 2 respectively. The effect of these changes on $\Delta(i)$ is shown in §16.4.

t'1(i) is the new value of t1(k) when correction 1 and global drifts have been taken into account. In general,

```
t'1[i] <— t1[k] + Δ[i] + dΔ0[i] + shift1[i]
```

in which shift1(i) represents the amount of correction 1. Similarly, t'2(i) is the new value of t2(k).

t"1(i) is the new value of t1(k) once correction 2 has also been taken into account. In general,

```
t"1[i] <— t'1[i] + shift2[i]
```

in which shift2(i) is the amount of correction 2. Similarly, t"2(i) is the new value of t2(k).

For any object E_k we notate Ts(i) the maximum value of t"2(ii) for ii < i, in which t"2(ii) is the temporary value of t2(kk) for any object E_{kk} in the same sequence as E_k:

[17] Indexing these arrays on *i* instead of *k* saves memory space.

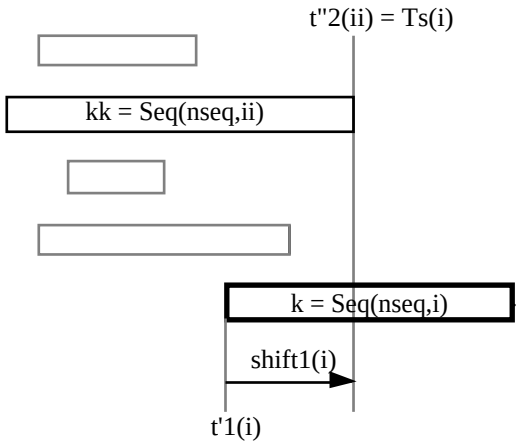


Fig.27 Finding Ts(i)

In order to locate E_k one must take into account topological properties ContEnd(jj), OverEnd(jj) and BrkTempo(jj) of E_{kk}, with jj = Obj(kk). Therefore the following values will to be stored:

```
ContEndPrev[i] <— ContEnd[jj]
OverEndPrev[i] <— OverEnd[jj]
BrkTempoPrev[i] <— BrkTempo[jj]
```

16.4 Incrementation

The `Locate()` procedure normally scans the sequence from left to right, i.e. the phase table Seq(nseq,i) with increasing values of *i.* When incrementing *i* the algorithm first looks for the next non-empty object:

```
Incrementation:
  /* Apply global drift to the next streak */
  dΔ0[i+1] <— dΔ0[i] + dΔ1[i] + dΔ2[i];

  /* Calculate the rank of the next non-empty sound-object */
  For (inext = i+1; i ≤ imax)
  |    dΔ0[inext] <— dΔ0[i+1];
  |    dΔ1[inext] <— dΔ2[inext] <— 0;
  |    If (Seq[nseq,inext] ≠ 0) break For;
  |    inext <— inext + 1;
  Endfor
```

Then the current value of Ts(i) is compared with the (temporary) clip date t"2(i) of the object with rank *i*. If the latter is bigger then Ts(i), ContEndPrev(i), etc., must be updated:

```
/* Update Ts */
If (Ts[i] < t"2[i])
then
|    Ts[inext] <— t"2[i];
|    ContEndPrev[inext] <— ContEnd[j];
|    OverEndPrev[inext] <— OverEnd[j];
|    BrkTempoPrev[inext] <— BrkTempo[j];
else
|    Ts[inext] <— Ts[i];
|    ContEndPrev[inext] <— ContEndPrev[i];
|    OverEndPrev[inext] <— OverEndPrev[i];
|    BrkTempoPrev[inext] <— BrkTempoPrev[i];
Endif
```

Now *i* can be incremented to the next value. If the end of the sequence is reached then the solution may be assessed by the user. If it is accepted then the temporary values of start/clip dates, local drift, etc., are taken into consideration. If the solution is not accepted then other solutions will be searched:

```
iprev <— i;
i <— inext;
k <— Seq[nseq,i];
If (k = -1)   /* End of sequence "NIL" */
then
|    If (Solution_accepted)
|    then
|    |    For (i = 1; i ≤ imax)
|    |    |    Δ[i] <— Δ[i] + dΔ0[i]+ dΔ1[i]+ dΔ2[i];
|    |    |    k <— Seq[nseq,i];
|    |    |    If (k > 0)
|    |    |    then
|    |    |    |    t1[k] <— t"1[i];
|    |    |    |    t2[k] <— t"2[i];
|    |    |    |    δ[k] <— δ1[i] + δ2[i];
|    |    |    Endif
|    |    |    i <— i + 1;
|    |    Endfor
|    |    Return(success);
|    else
|    |    …    /* Find more solutions (not described here) */
|    Endif
Endif
```

The following is the initialisation of all temporary performance parameters of the new object considered, following which a correction value *shift1* is calculated. Informally, *shift1* is the amount of overlapping between the current object and the rightmost preceding object in the sequence (see Fig.27).

```
j <— Obj[k];
TrBeg[k] <— TrEnd[k] <— 0;
t"1[i] <— t'1[i] <— t1[k] + Δ[i] + dΔ0[i];
t"2[i] <— t'2[i] <— t2[k] + Δ[i] + dΔ0[i];
shift1[i] <— Ts[i] - t"1[i];
```

Now it is necessary to check that the current location of the sound-object is acceptable. This assessment is the output of function `Situation()` which will be examined now.

If the function returns 'true' then incrementation is called again.[18] The following instructions are easy to trace in the flowchart given in §16.2.

```
If (Situation(i,nseq,shift1,nature_of_time,t"1[i],t"2[i]) = acceptable)
then
|    shift1 <— 0;
Endif
sol_set1[i] <—
   Possible_choices(i,nseq,t'1[i],t'2[i],shift1,Ts[i],nature_of_time,1)
   ;
If(shift1 = 0)
then
|    go to Incrementation
Endif
If (sol_set1[i] = empty)
then
|    Tsm <— Ts[i];
|    shift2 <— - shift1;
|    go to Decrementation;
Endif
go to New_choice1;
```

16.5 Situations

Function `Situation()` is described now.

```
Function Situation(i,nseq,shift,nature_of_time,t1,t2)

Begin
  If (i = 1) Return(acceptable);
  k <— Seq[nseq,i];
  j <— Obj[k];
  If ((nature_of_time = smooth) and (nseq = 1)) Return(inacceptable);
  If ((shift < 0) and (not ContBeg[j]) and (not ContEndPrev[i]))
  then
  |    Return(acceptable);                    /* 1 */
  Endif
  If (shift = 0)  Return(acceptable);        /* 2 */
  If (shift > 0)
  then
  |    If(j = 1)   /* Obj[k] is a silence */
  |    then
  |    |    Return(acceptable);
  |    Endif
  |    If ((Ts ≤ t2) and OverBeg[j] and OverEndPrev[i])
  |    then
  |    |    Return(acceptable);
  |    Endif                                  /* 3 */
  |    If ((Ts > t2) and OverBeg[j] and OverEnd[j] and OverEndPrev[i] and
  |      not (ContEnd[j]))
  |    then
  |    |    Return(acceptable);               /* 4 */
  |    Endif
  Endif
  Return(inacceptable);
End
```

[18] Note that this and other procedures are not recursive. We used 'go to' statements deliberately in the pseudo-code to connect modules of instructions at the same level.

The implication of this function is that, in striated time or if nseq > 1, four situations are possible depending on the position of E_k relative to E_{kk}:

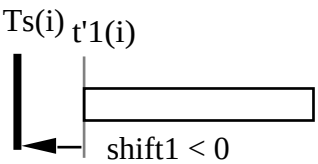
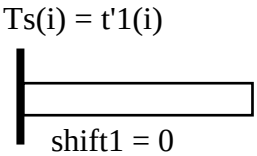
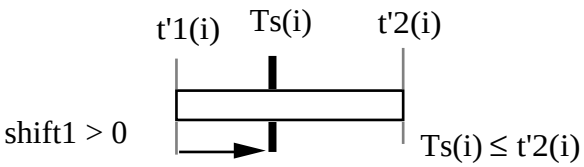
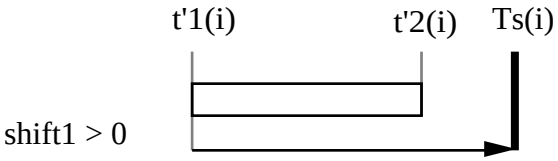
Nr	Configuration	Condition
1		(not ContBeg(j)) and (not ContEndPrev(i))
2		none
3		OverBeg(j) and OverEndPrev(i)
4		OverBeg(j) and OverEnd(j) and (not ContEnd(j)) and OverEndPrev(i)

Fig.28 The four situations

We call **canonic correction 1** the smallest modification of t'1(i) yielding an acceptable solution. It is easy to prove that the canonic correction 1 is:

```
t'1[i]  <—  t'1[i] + shift1[i]
```

There are three possible transformations yielding this canonic correction:

— Local drift: both the start and clip dates are incremented.

— Global drift (only if shift > 0): same as above, but the reference streak is also relocated.

— Truncating the beginning (only if shift > 0): only the start date is changed.

16.6 Solution set 1

The same function

```
Possible_choices(i,nseq,shift,t1,t2,Ts,nature_of_time,side)
```

may yield a solution set for correction 1 or for correction 2 (see infra) depending on the setting of the flag *side*. When it is set to 1 the function yields `sol_set1(i)`, i.e. an

arbitrarily ordered list of possible changes relative to object E_k that may be used for the canonic correction 1.

```
Function
  Possible_choices(i,nseq,shift,t1,t2,Ts,nature_of_time,side)

Begin
  sol <— empty;
  k <— Seq[nseq,i];
  j <— Obj[k];
  If ((side = 1) and (nature_of_time = smooth) and (nseq = 1) and (shift
    > 0))
  then
  |    Return({Break tempo});
  Endif
  If (Reloc[j] or j = 1)  /* j = 1 for a silence. */
  then
  |    sol <— sol ∪ {Shift object};
  Endif
  If ((side = 1) and (i > 1) and (shift > 0) and (BrkTempoPrev[i] or
    (nature_of_time = smooth)))
  then
  |    sol <— sol ∪ {Break tempo};
  Endif
  If ((side = 1) and (shift > 0) and TruncBeg[j] and (t2 > Ts))
  then
  |    sol <— sol ∪ {Truncate beginning};
  Endif
  If ((side = 2) and (shift < 0) and TruncEnd[j] and (t1 < (Ts + shift)))
  then
  |    sol <— sol ∪ {Truncate end};
  Endif
  Return(sol);
End
```

16.7 Correction 1

This correction is performed on the basis of a solution selected in the solution set `sol_set1(i)`. The choice may either be decided by the user or by an arbitrary enumeration of `sol_set1(i)`.

```
New_choice1:
  If (sol_set1[i] = empty)
  then
  |    shift2 <– - shift1;
  |    go to Decrementation;
  Endif
  sol1 <– Next_choice(sol_set1[i]);
  sol_set1[i] <– sol_set1[i] \ {sol1};
  t'1[i] <– t1[k] + Δ[i] + dΔ0[i] + shift1;
  t'2[i] <– t2[k] + Δ[i] + dΔ0[i];
  dΔ1[i] <– 0;
  δ1[i] <– 0;
  TrEnd[k] <– 0;
  If (sol_set1 = Truncate beginning)
  then
  |    TrBeg[k] <– shift1;
  else
  |    TrBeg[k] <– 0;
  |    t'2[i] <– t'2[i] + shift1;
  |    If (sol1 = Shift object) δ1[i] <– shift1;
  |    If (sol1 = Break tempo) dΔ1[i] <– shift1;
  Endif
  t"1[i] <– t'1[i];
  t"2[i] <– t'2[i];
  go to Incrementation;
```

Procedure `Next_choice()` is not described here.

It can be proved that if `sol_set1(i)` is not empty for any value of *i*, then `Locate()` returns a straightforward solution, so that in this case the time-complexity of this procedure is O(n), given *n* the number of objects in the sequence.

16.8 Solution set 2

If the solution set `sol_set1(i)` is empty, it is necessary to assert:

```
shift2 <– - shift1
```

which means informally: “since t'1(i) cannot be increased by shift1, then try to decrease Ts(i) by -shift1”. Here again the correction will be canonic since |shift2| is the minimum value yielding situation 2 (see Fig.28) for the object with rank *i*.

```
Decrementation:
  If (i = 1) Return(failed);
  Tsm <– Ts[i];
  i <– iprev;
  For (iprev = i - 1; iprev ≥ 0)
  |    If (Seq[nseq,iprev] > 0) break For;
  |    iprev <– iprev - 1;
  Endfor
  k <– Seq[nseq,i]; /* k is strictly positive */
  j = Obj[k];
  If (Ts[i] = Tsm) go to Decrementation;
  sol_set2 <–
    Possible_choices(i,nseq,shift2,t"1[i],t"2[i],Tsm,nature_of_time,2);
  go to New_choice2;
End
```

The new value of *i* is *iprev*, the rank of the preceding non-empty sound-object. Then E_{kk} is searched comparing Ts[i'] with Tsm for each i' < i:

```
If (Ts[i] = Tsm) go to Decrementation
```

A new solution set `sol_set2` is determined by invoking the function

```
Possible_choices(i,nseq,shift,t1,t2,Ts,nature_of_time,side)
```

with shift = shift2, t1 = t"1(i), t2 = t"2(i), Ts = Tsm, and side = 2.

16.9 Correction 2

For this correction a solution sol2 is selected in `sol_set2`.

```
New_choice2:
  If(sol_set2 = empty)
  then
  |    If(sol_set1[i] ≠ empty)
  |    then
  |    |    go to New_choice1;
  |    else
  |    |    If((i > 1) and (Reloc[j] or (shift2 > 0 and 0 < ibreak < i)))
  |    |    then
  |    |    |    go to Decrementation;
  |    |    else
  |    |    |    Return(failed); /* Backtracking possible here */
  |    |    Endif
  |    Endif
  Endif
  sol2 <— Next_choice(sol_set2);
  sol_set2 <— sol_set2 \ {sol2};
  t"1mem <— t"1[i]; t"2mem <— t"2[i];
  δ2mem <— δ2[i]; dΔ2mem <— dΔ2[i];
  t"2[i] <— t"2[i] + shift2;
  If(sol2 = Truncate end)
  then
  |    TrEnd[k] <—  TrEnd[k] - shift2;
  else
  |    t"1[i] <— t"1[i] + shift2;
  |    If(sol2 = Shift object) δ2[i] <— δ2[i] + shift2;
  |    If(sol2 = Break tempo) dΔ2[i] <— dΔ2[i] + shift2;
  Endif

  shift3 <— Ts[i] - t"1[i];
  shift4 <— Alternate_correction1(i,nseq,shift3,t"2[i]);
  If(shift4 = 0)
  then
  |    go to Incrementation;
  else
  |    If(sol_set2 ≠ empty)
  |    then
  |    |    t"1[i] <— t"1mem; t"2[i] <— t"2mem;  /* TrEnd[k] was not
  |    |                                                modified */
  |    |    δ2[i] <— δ2mem; dΔ2[i] <— dΔ2mem;
  |    |    go to New_choice2;
  |    else
  |    |    shift2 <— - shift4;
  |    |    go to Decrementation;
  |    Endif
  Endif
```

The condition

```
If((i > 1) and (Reloc[j] or (shift2 > 0 and 0 < ibreak < i)))
```

is the one mentioned on the flowchart (Fig.26) as “backtrack useful”. It is used to cut the search space of solutions. Indeed, trying to modify any object preceding Obj(k) will certainly not improve the situation unless Obj(k) itself is either relocatable or, in case shift2 > 0, it is situated to the right of another object that may break the tempo. In view of this test the index of the leftmost object in the sequence with property BrkTempo (see §12.3.2) is stored as *ibreak*.

16.10 Alternate correction 1

Changing the clip date of sound-object E_k modifies the topological constraints on objects following E_k in the sequence. These constraints will be evaluated and taken into account while further scanning the sequence. Once a solution has chosen in `sol_set1(i)`, correction 1 is performed and the following objects in the sequence are examined. (See `incrementation` on the flowchart, Fig.26)

On the other hand, whenever correction 2 modifies the start date of E_k it is necessary to check that the constraints on all objects preceding E_k in the sequence are still satisfied. Let E_{kk} and *shift* be defined as in §16.2. The function `Alternate_correction1()` checks the topological situation between E_k and E_{kk} (as per Fig.28). In situation 3 it attempts to truncate the beginning of E_k. If the correction was successful then ‘0’ is returned and incrementation starts again (see flowchart). Otherwise, another solution is selected for correction 2. If `sol_set2` is empty, then another solution is tried for correction 1, etc.

```
Function Alternate_correction1(i,nseq,shift,t2)

Begin
  k <— Seq[nseq,i];
  j <— Obj[k];
  t1 <— t"1[i];
  If (shift = 0)  Return(0);     /* Situation 2 */
  If (shift < 0 and (not ContBeg[j]) and (not ContEndPrev[i]))
     Return(0);                  /* Situation 1 */
  If (shift > 0)                 /* Situation 3 or 4 */
  then
  |     If(OverBeg[j] and OverEndPrev[i] and (((t1 + shift) ≤ t2) or (not
  |       ContEnd[j]))  Return(0);
  |     If (TruncBeg[j])
  |     then
  |     |     If ((t1 + shift) ≤ t2) /* Situation 3 */
  |     |     then
  |     |     |     TrBeg[k] <— TrBeg[k] + shift;
  |     |     |     t"1[i] <— t"1[i] + shift;
  |     |     |     Return(0);
  |     |     Endif
  |     Endif
  Endif
  Return(shift);
End
```

This function returns 0 in acceptable situations. In situation 3, if TruncBeg(j) is true it first truncates the beginning of the object, then it returns 0. In all other cases it returns the input value shift3.

If the returned value shift4 is 0, *i* is incremented again. If shift4 ≠ 0 the next solution in `sol_set2` is considered. Once `sol_set2` is empty the algorithm jumps back to `Decrementation`.

It is proved [Bel 1990b:164] that correction 2 always improves the situation and that if there is a solution it will be found after a finite number of steps.

16.11 Multiple corrections

Correction 2 may be performed several times on the same object, first when calling `Decrementation` on an object with rank i1 > i, then on an object with rank i2 > i1, and so on.

It can be proved that for any object the number of type-2 corrections is finite.

17. Complexity of the time-setting algorithm

The following results have been proved in [Bel 1990b:165-168]:

— The `Locate()` procedure halts after a finite number of steps.

— If, for some value of *nseq*, `Locate()` returns a solution with $\Delta(i) \neq 0$ for some value of *i* (i.e. a global drift), then there is a solution in which T(i) may be replaced with $T(i)+\Delta(i)$.

— In the worst case the time complexity of the `Locate()` procedure is $O(imax^3)$, where *imax* is the number of objects in the sequence.

— If no global drift is created, the time complexity of the time-setting algorithm is $O(nmax.imax^3)$, where *nmax* is the number of sequences and *imax* the maximum length of a sequence. In the worst case, the time complexity is $O(nmax^2.imax^3)$.

18. Typical examples

18.1 Non-empty sound-object prototypes

The table below defines six non-empty striated sound-object prototypes. Objects “a” and “a'” are identical as far as messages are concerned, but they differ in their durations and sonic properties.

The first line in the table is the duration of the object prototype in seconds. Reference periods (the metronome value when entering the prototype) are given on the second line.

	a	a'	b	c	d	e
Dur(j)	1.0	1.5	2.0	4.0	1.0	0.5
Tref(j)	1.0	1.5	1.0	2.0	2.0	1.0
Properties	PivBeg OkRescale OverBeg OverEnd TruncEnd	PivBeg OkExpand Reloc BrkTempo TruncEnd	PivBeg OkRescale Reloc OverEnd TruncBeg	PivBeg OkRescale OverBeg OverEnd	PivEnd OkRescale OverBeg OverEnd	PivCent OkRescale OverBeg OverEnd

18.2 Calculating ime-scale ratios α(k) and time-setting a sequence

(This is an application of rules listed in §15)

Consider the polymetric expression

/2 abc /3 de

and its dilated notation

/6 a _ _ b _ _ c _ _ d _ e _

with dilation ratio: Ratio = 6 (see §8.2). The sequence contains: imax = 14 symbols.

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
E_k	a	_	_	b	_	_	c	_	_	d	_	e	_	NIL

18.2.1 Non-measured smooth time

The solution is

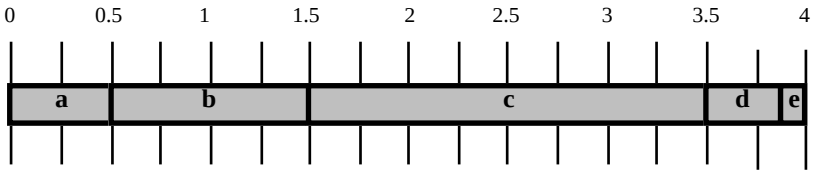


Fig.29 /2 a b c /3 d e
(non-measured smooth time, smooth or striated objects)

with α(k) = 0.5 for “a”, “b” and “c”, and 0.33 for “d” and “e”.

18.2.2 Striated objects in measured smooth time

Let us take Tclock = 1s. The instance table yields:

	a	b	c	d	e
d(k)	1/2	1/2	1/2	1/3	1/3
Dur(j)	(1 s.)	(2 s.)	(4 s.)	(1 s.)	(0.5 s.)
Tref(j)	1	1	2	2	1
α(k)	0.5	0.5	0.25	0.166	0.33
physical duration	0.5 s.	1 s.	1 s.	0.16 s.	0.16 s.

(Data between brackets have not been used.)

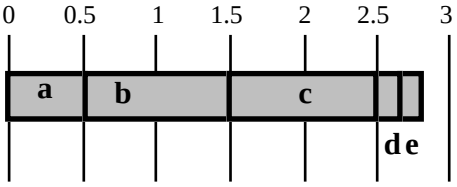


Fig.30 /2 a b c /3 d e
(measured smooth time, striated objects)

18.2.3 Sequence of smooth objects in measured smooth time

Here we suppose for a while that all objects defined in §18.1 are smooth.

	a	b	c	d	e
d(k)	1/2	1/2	1/2	1/3	1/3
Dur(j)	1 s.	2 s.	4 s.	1 s.	0.5 s.
α(k)	0.5	0.25	0.125	0.33	0.66
physical duration	0.5 s.	0.5 s.	0.5 s.	0.33 s.	0.33 s.

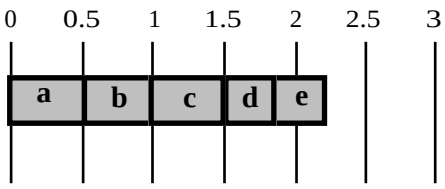


Fig.31 /2 a b c /3 d e
(measured smooth time, smooth objects)

18.2.4 Striated objects in striated time

Let us assume a metronomic structure of time with period 3 seconds. The following is the table of T(i) in seconds and the corresponding objects in the sequence:

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
T(i)	0	0.5	1	1.5	2	2.5	3	3.5	4	4.5	5	5.5	6	6.5
E_k	a	_	_	b	_	_	c	_	_	d	_	e	_	NIL
k	1	0	0	2	0	0	3	0	0	4	0	5	0	-1

$\Delta(i) = 0$ for every *i*. As to "b", for instance, i = 4, inext = 7, and Tref(j) = 2 s. Therefore

$$\alpha(2) = \frac{3 - 1.5}{1} = 1.5$$

The physical duration of this object is: 1.5 x 2 = 3 s.

Positioning objects implies taking into account properties PivBeg for "a", "b" and "c", PivEnd for "d" and PivCent for "e". The final lay-out is:

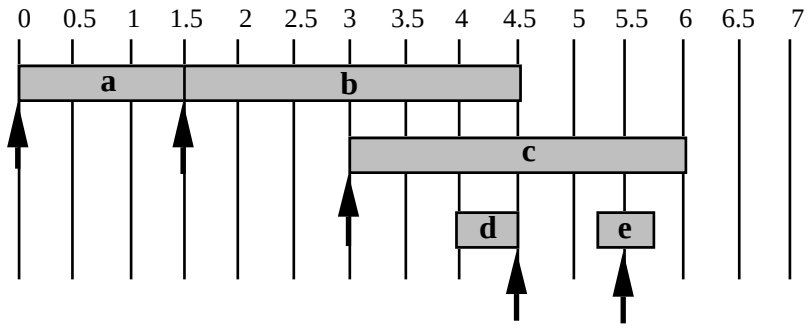


Fig.32 /2 a b c /3 d e
i.e. /6 a _ _ b _ _ c _ _ d _ e _
(metronomic striated time, period 3s., striated objects)

18.3 Time-setting polymetric structures

(See **Main loop** §16.1)

Consider

/3 ab { cde, ab } cd

which is interpreted

/6 a _ b _ { c_ d _ e _ , a _ _ b _ _ } c _ d _

or equivalently

/6 a _ b _ { a _ _ b _ _ , c_ d _ e _ } c _ d _

yielding a possible phase diagram (Ratio = 6):

a _ b _ a _ _ b _ _ c _ d _ NIL

_ _ _ _ c _ d _ e _ NIL

18.3.1 Non-measured smooth time, smooth objects

The first sequence (“ababcd”) makes no difficulty. Then streaks are created by interpolating the physical durations of non-empty sound-objects in the first sequence. See the dotted lines on Fig.33:

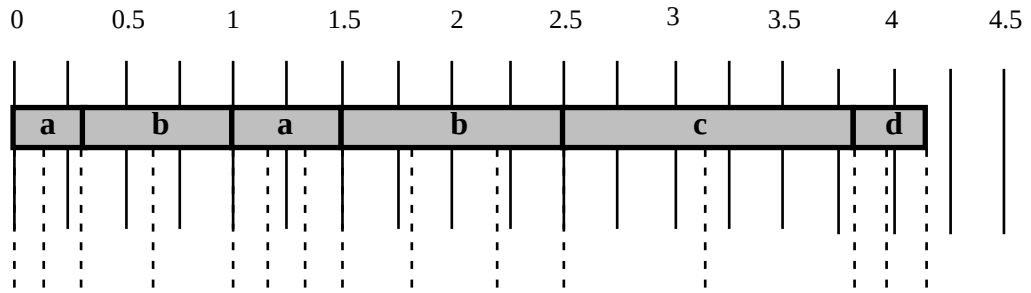


Fig.33 Interpolating streaks on the first sequence

The next step is the time-setting of the second sequence, using the interpolated streaks:

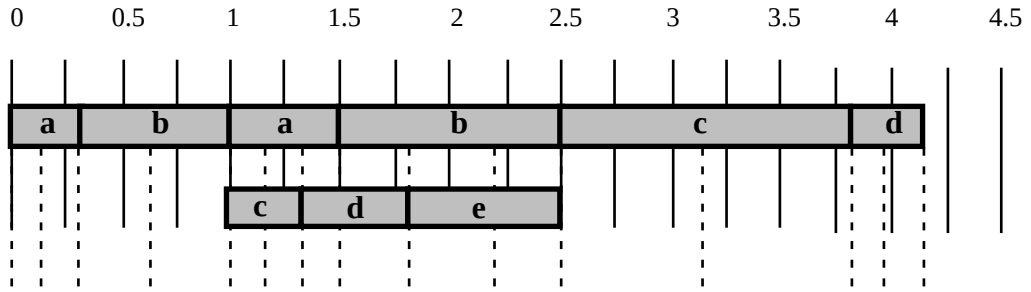


Fig.34
/3 ab { cde , ab } cd
i.e. /6 a _ b _ { a _ _ b _ _ , c _ d _ e _ } c _ d _
(non-measured smooth time, smooth objects)

18.3.2 Striated time

Consider examples:

/3 ab { cde , ab} cd
i.e. /6 a _ b _ { c _ d _ e _ , a _ _ b _ _ } c _ d _

/3 a'b { cde , a'b} cd
i.e. /6 a' _ b _ { c _ d _ e _ , a' _ _ b _ _ } c _ d _

These have similar symbolic representations but different time-settings because of the properties of sound-objects “a” and “a'”. With metronomic time, period 3 s., we get:

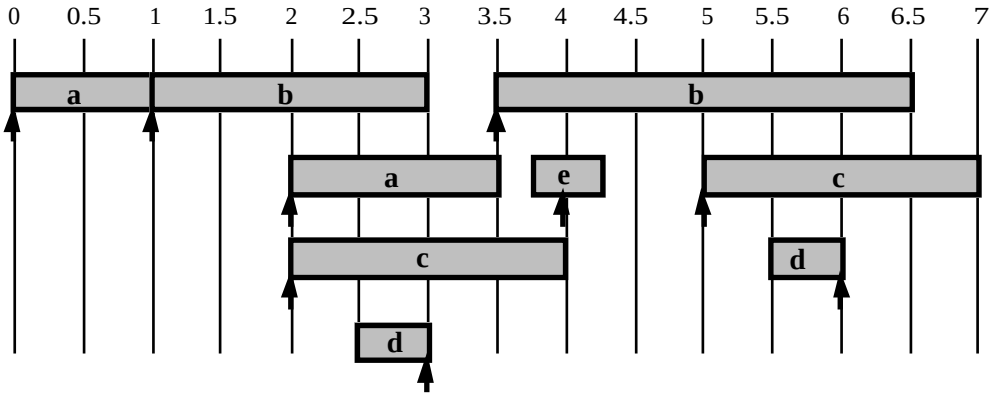


Fig.35
/6 a _ b _ { c _ d _ e _ , a _ _ b _ _ } c _ d _
(metronomic time, period 3 s.)

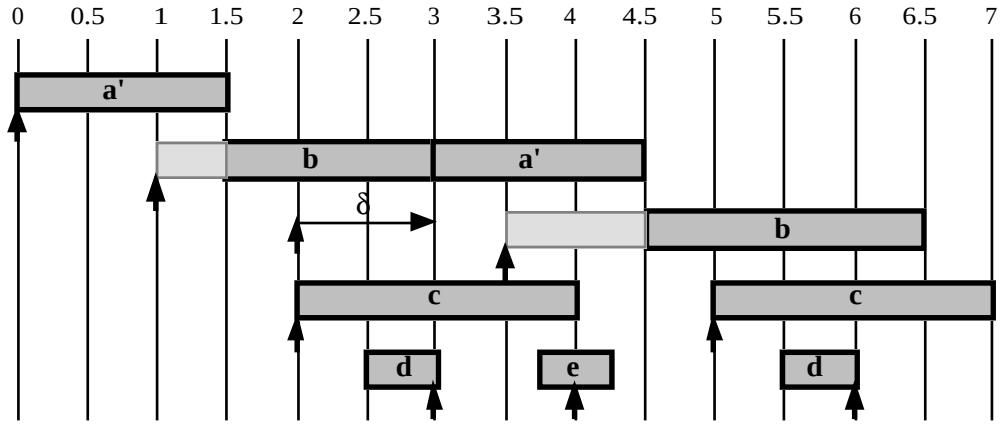


Fig.36
/6 a' _ b _ { a' _ _ b _ _ , c _ d _ e _ } c _ d _
(metronomic time, period 3 s.)

In the latter example, since “a'” does not have property OkCompress it cannot accept $\alpha(k) = 0.66$, therefore $\alpha(k)$ is forced to 1. Object “b” should start before “a'” clips, but “b” does not have property OverBeg. Therefore the beginning of “b” is truncated. Since “a'” does not have OverBeg, its second instance cannot start before “b” clips. A local drift of “a'” solves this constraint. Now “b” may overlap “a'”, which is not acceptable, therefore “b” is truncated in its beginning.

The last example

{b - ba, /2 a'c {de, /3 a - c}}

i.e. /6 {b _ _ - _ _ b _ _ a _ _ , a' _ _ c _ _ {d _ _ e _ _ , a_ - _ c _ }}

is shown in two performances (Fig.37-38) with respective metronome periods 3 s. and 2 s. The fact that "a'" does not have the OverEnd property forces a correction 2 (with negative drift δ) to prevent it from overlapping "c".

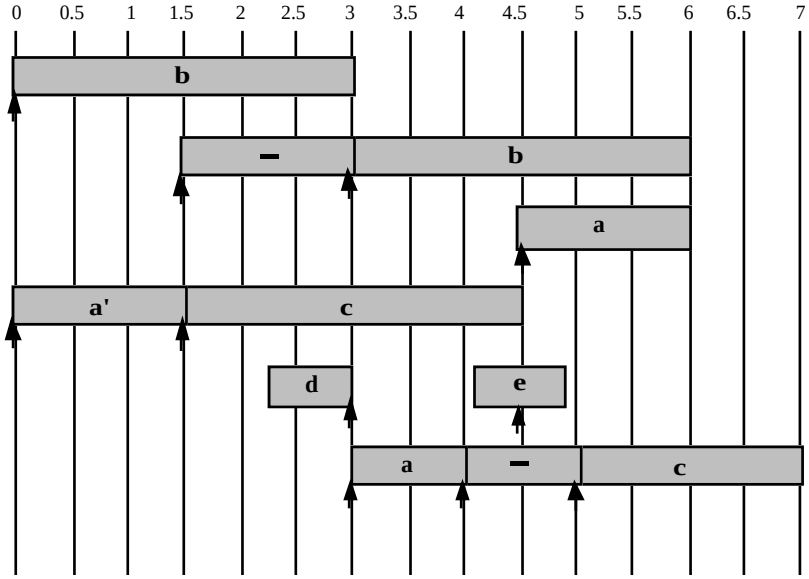


Fig.37
/6 { b _ _ - _ _ b _ _ a _ _ , a' _ _ c _ _ { d _ _ e _ _ , a _ - _ c _ }}
(metronomic time, period 3 s.)

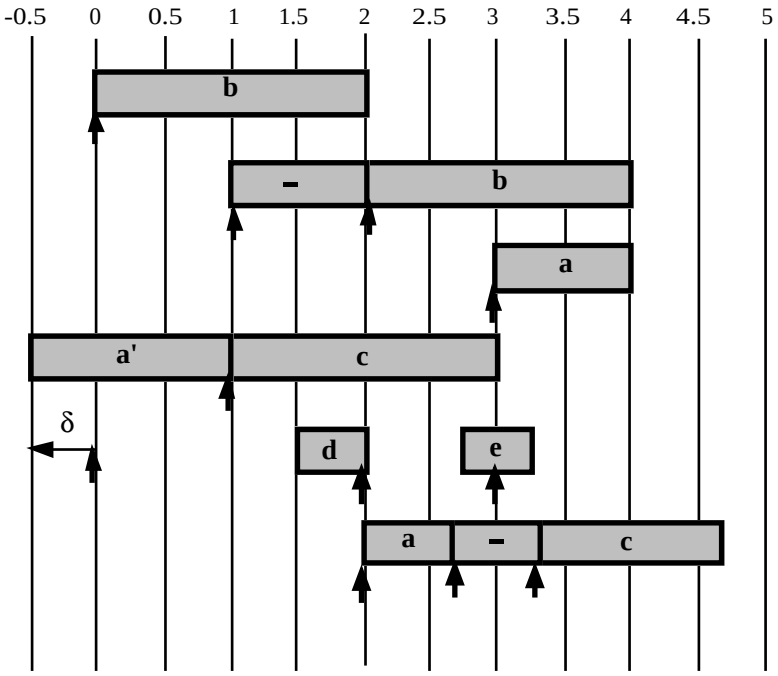


Fig.38
/6 { b _ _ - _ _ b _ _ a _ _ , a' _ _ c _ _ { d _ _ e _ _ , a _ - _ c _ }}
(metronomic time, period 2 s.)

19. Conclusion

The musical examples in §18 give an indication of the variety of solutions proposed by the time-setting algorithm (on the basis of properties of sound-objects and different structures of time) for the performance of a given musical item. Equally versatile is the interpretation of polymetric expressions generated by formal string-grammars, as shown in part B. Both approaches, therefore, contribute to compensate the rigidity of the timing of computer-generated musical pieces: the synchronization and accurate timings of concurrent musical processes are handled by the computer on the basis of (possibly incomplete) information on structures and sound-objects. This allows a composer to explore sets of musical productions generated by rewriting rules, either in a systematic way — assessing every decision of the machine — or in situations involving one or several computers/sound processors interacting in real-time improvisation.

References

Allen, J.F. 1983

Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, 26, 11:832-843

Ames, C. 1989

Tutorial and Cookbook for COMPOSE. Oakland CA (USA): Frog Peak Music.

Anderson, D.P.; and Kuivila, R. 1989

Continuous abstractions for discrete event languages. *Computer Music Journal,* 13, 3:11-23.

Bel, B. 1990a

Time and musical structures. *Interface*, 19, 2-3:107-135.

1990b

Acquisition et Représentation de Connaissances en Musique. Thèse de Doctorat, Faculté des Sciences de St-Jérôme, Université Aix-Marseille III.

1990c

Bol Processor BP2: reference manual. ISTAR France, Marseille.

1991a

BP grammars. In *Readings in AI and Music*, eds. M. Balaban, K. Ebcioglu & O. Laske, AAAI Press. (Forthcoming)

1991b

Symbolic and sonic representations of sound-object structures. In *Readings in AI and Music*, eds. M. Balaban, K. Ebcioglu & O. Laske, AAAI Press. (Forthcoming)

Blacking, J. 1984

What languages do musical grammars describe? In *Musical Grammars and Computer Analysis*, eds. M. Baroni and L. Callegari, 363-370. Firenze (Italy): Olschki.

Borin, G.; De Poli, G.; and Sarti, A. 1990

Formalization of the sound generation process — Structures and metaphors. In Proceedings of Colloque "Musique et Assistance Informatique" (MAI 90), eds. B. Vecchione and B. Bel, 231-241. Marseille (France): Laboratoire Musique et Informatique.

Boulez, P. 1963

Penser la musique aujourd'hui. Paris (France): Gonthier.

Chomsky, N.; and Halle, M. 1968

Principes de phonologie générative. Paris (France):Seuil, 1973. Original edition: New York: Harper and Row.

Duthen, J.; and Stroppa, M. 1990

Une représentation de structures temporelles par synchronisation de pivots. In *Proceedings of Colloque "Musique et Assistance Informatique"* (MAI 90), eds. B. Vecchione and B. Bel, 471-479. (Forthcoming)

Jaffe, D. 1985

Ensemble timing in computer music. *Computer Music Journal*, 9, 4:38-48.

Jakobson, R. 1963

Essais de linguistique générale. Paris (France): Editions de Minuit.

Laske, O. 1972

On problems of a performance model for music. Utrecht (The Netherlands): Institute of Sonology, Utrecht State University.

1989

Composition Theory: An Enrichment of Music Theory. *Interface*, 18:45-59.

Oppo, F. 1984

Per una teoria generale del languaggio musicale In *Musical Grammars and Computer Analysis*, eds. M. Baroni and L. Callegari, 115-130. Firenze (Italy): Olschki.

Risset, J.C. 1989

Computer music experiments 1964-… In *The Music Machine*, ed. C. Roads, 67-74. Cambridge MA (USA): MIT Press.

Schaeffer, P. 1966

Traité des objets musicaux. Paris (France): Seuil.

Truax, B. 1990

Chaotic Non-Linear Systems and Digital Synthesis: An Explanatory Study. In *Proceedings of the International Computer Music Conference* (ICMC), Glasgow:100-103.

Vecchione, B. 1984

Pour une science de la réalité musicale. Thèse de Doctorat, Université de Provence Aix-Marseille I.

Xenakis, I. 1963

Musiques formelles. Paris (France): La Revue Musicale. Augmented translation: *Formalized music*, 1971. Bloomington (USA): Indiana University Press.

1972

Vers une métamusique. In *Architecture et Musique*, 38-70. Paris (France): Casterman.

Appendix

The formal definitions of object prototypes are given here as a guide-line for the design of a procedure dispatching elementary messages in real time. The procedure itself is not described as it is highly dependant on the implementation. The main design requirement, which this formalism helps fulfilling, is a low-level representation of musical items that does not contain all the information stored in sound-object prototypes, so that in the end very large sound-objects can be instantiated and performed in real time.

1. Sound-object prototypes

Let $A = \{A_1,\ldots,A_N\}$ be an arbitrary set of **elementary actions** or **messages** destined to a sound processor, $\mathcal{P}(A)$ the set of all subsets of A, and R the set of real numbers, representing physical time. In a macro-level description of sound, A_i may be an instruction label (e.g. a MIDI code), whereas in a micro-level description it may be a vector of numerical parameters. Both representations may coexist in the same implementation.

Let $\{Ep_j \mid j = 0, \ldots jmax\}$ designate a finite set of **sound-object prototypes** defined as follows:

Definition

> The j-th sound-object prototype is a mapping
>
> Ep_j: $\quad R \cup \{nil\} \longrightarrow \mathcal{P}(A)$
>
> such that $Ep_j(nil) = \varnothing$, and $Ep_j(t) \neq \varnothing$ for a finite set of values of $t \in R$.

Conventionally, the time origin indicates the pivot of the sound-object. The utility of the "nil" value will be shown below.

It is assumed that sound-object prototypes are predefined. They may be played on an instrument (see §2) and edited as a list of time-stamped MIDI codes. They may also be defined as arbitrary functions mapping time to macro-level or micro-level parameters (see [Truax 1990:101-102]).[19]

By convention, Ep_0 designates the prototype of the **empty sound-object** labelled "_", with $Ep_0(t) = \varnothing$ for any $t \in R$. Ep_1 may designate the **silence** notated "-".

Given an arbitrary time origin, the **start/clip dates** of the prototype are respectively tmin(j) and tmax(j) such that:

$$Ep(tmin(j)) \neq \varnothing\ ,\quad Ep(tmax(j)) \neq \varnothing\ ,\ \text{and}\ \forall t \notin [tmin(j),tmax(j)],\ Ep_j(t) = \varnothing\ .$$

Therefore, $Ep_j(tmin(j))$ contains the very first message of the object and $Ep_j(tmax(j))$ its last message.[20] An illustration of the Ep_j mapping of a sound-object prototype is given in Fig.39.

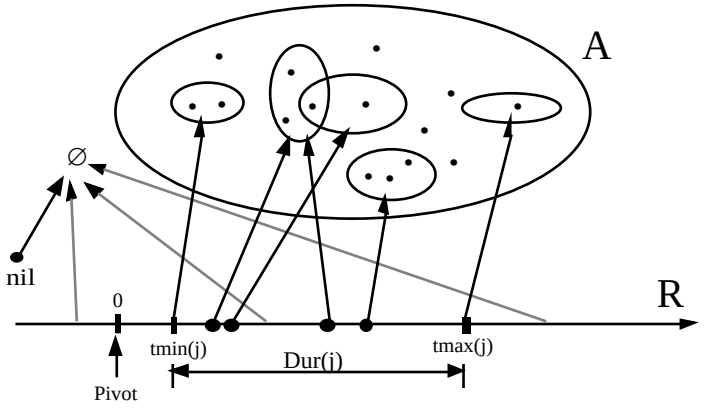


Fig.39 A sound-object prototype and the Ep_j mapping

2. Out-time object prototypes

Given a sound-object prototype Ep_j, the prototype of the corresponding **out-time object** $E'p_j$ is:

$$E'p_j(0) = \bigcup_{t \in [tmin,tmax]} Ep_j(t)$$

$$E'p_j(t) = \varnothing \quad \forall t \neq 0$$

Informally, an out-time object has all its messages at the same date. In an actual implementation, the strict ordering of messages may be maintained while the delay from

[19] Non-linear mappings generating "chaotic objects" are being implemented in Bol Processor BP2.

[20] In smooth sound-objects (see §12.1) the time origin is such that tmin = 0, i.e. the default position of the pivot is the date of the first message.

one message to the next is rendered very small. Fig.40 shows the $E'p_j$ mapping of the out-time object prototype corresponding to the sound-object prototype defined in Fig.39.

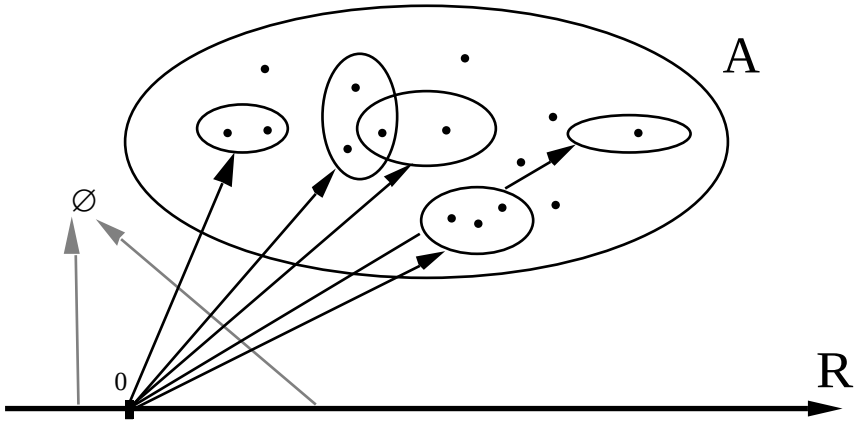


Fig.40 An out-time object prototype and the $E'p_j$ mapping

3. The time-setting function f(t)

Let t1(k) and t2(k) be the physical start/clip dates of a (non-empty) sound-object E_k. The object may be truncated, in which case either TrBeg(k) or TrEnd(k) is positive (see §14). Its time pivot is located at physical date T(i) + Δ(i) + δ(k), where *i* is the rank of its reference streak and k = Seq(nseq,i).

Let *imax* and *nmax* be the maximum values of *i* and *nseq* in a sound-object structure (i.e. the dimensions of the phase diagram). We call **time-setting** of the sound-object structure the function

$$f: \quad R \longrightarrow \mathcal{P}(A)$$

such that

$\forall t \in R,$

$$f(t) = \left(\bigcup_{\substack{i = 1, imax \\ nseq = 1, nmax}} \left[Ep_j(LocalTime_k(t)) \right] \right) \cup \left(\bigcup_{\substack{i = 1, imax \\ nseq = 1, nmax}} \left[E'p_j(LocalTime'_k(t)) \right] \right)$$

with j = Obj(k) and k = Seq(nseq,i)

The first member of f(t) is the set of instances of **sound-objects** (i.e. E_k with d(k) > 0), for which **local time** is defined as follows:

(1) $\forall t \in [t1(k),t2(k)]$,

$$LocalTime_k(t) = \frac{t - T(i) - \Delta(i) - \delta(k)}{\alpha(k)}$$

with j = Obj(k) and k = Seq(nseq,i) ;

(2) $\forall t < t1(k)$,

(2.1) $LocalTime_k(t) = nil$ if $Ep_j(LocalTime_k(t))$ is of type ON ;

(2.2) $LocalTime_k(t) = LocalTime_k(t1(k))$ otherwise ;

(3) $\forall t > t2(k)$,

(3.1) $LocalTime_k(t) = nil$ if $Ep_j(LocalTime_k(t))$ is of type ON ;

(3.2) $LocalTime_k(t) = LocalTime_k(t2(k))$ otherwise.

Cases (2) and (3) refer to objects truncated in the beginning and the end respectively. The "**type ON**" subsets of A are the ones that contain a message initiating a process in the sound processor (e.g. a "NoteOn" message in a MIDI implementation). These are deleted if found in the truncated part of an object (see cases 2.1 and 3.1). Other messages are kept but they are relocated at the start/clip dates of the object (see cases 2.2 and 3.2). Fig.41 shows a truncated sound-object whose prototype could be the one shown in Fig.39.

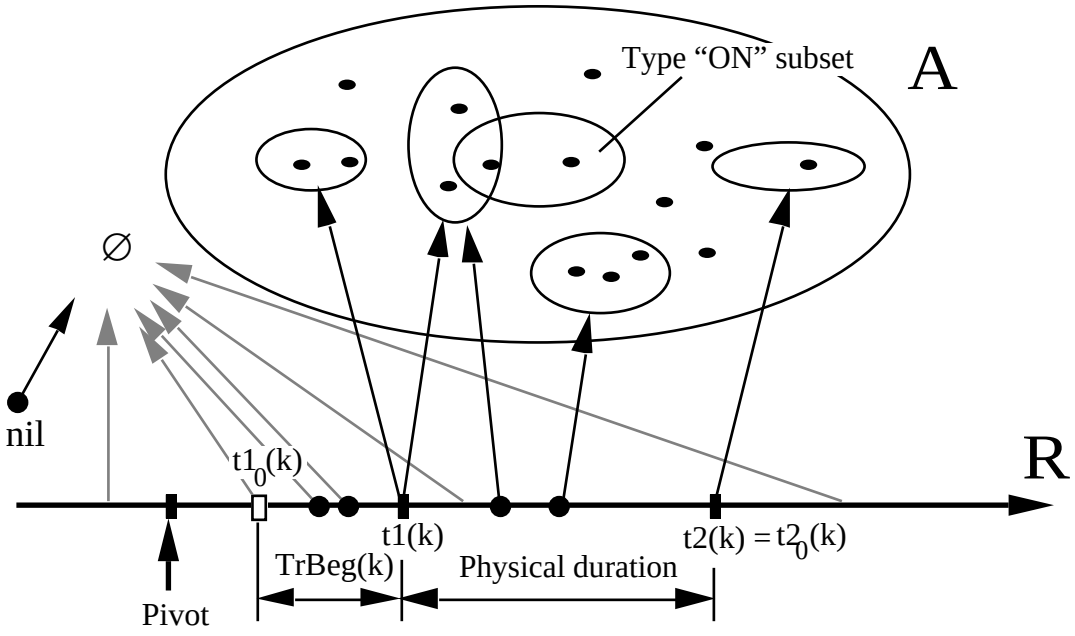


Fig.41 A sound-object truncated in its beginning

The second member of f(t) is the set of instances of **out-time objects** (i.e. E_k with d(k) = 0), for which local time is:

$$\text{LocalTime}'_k(t) = t - T(i) - \Delta(i) - \delta(k) .$$

Given the time-setting function f(t), all messages referenced by f(t) can be dispatched in real time to the sound processor. Instantiating a sound-object is therefore possible as soon as $\alpha(k)$, t1(k), t2(k), TrBeg(k) and TrEnd(k) have been calculated. The actual sequence of messages defining the object prototype may be retrieved from some other part of the memory — possibly a hard disk.

Since for every *t*, f(t) is a subset of A, messages are not arranged in a strict sequence. In a practical implementation an arbitrary ordering of simultaneous elementary actions may be imposed so that the corresponding messages are dispatched in sequence with a very small delay.

An additional necessary feature is the management of ON/OFF processes: when a sound-object structure containing several sequences is instantiated, the same process may be invoked several times (by several occurrences of the same type ON message) before it is stopped by a type OFF message. This can be avoided, either by ignoring redundant ON messages or by sending an additional OFF message just before triggering again the process.