



## City Research Online

### City, University of London Institutional Repository

---

**Citation:** Desain, P.W.M. (1991). Structure and expressive timing in music performance. (Unpublished Doctoral thesis, City, University of London)

This is the accepted version of the paper.

This version of the publication may differ from the final published version.

---

**Permanent repository link:** <https://openaccess.city.ac.uk/id/eprint/28495/>

**Link to published version:**

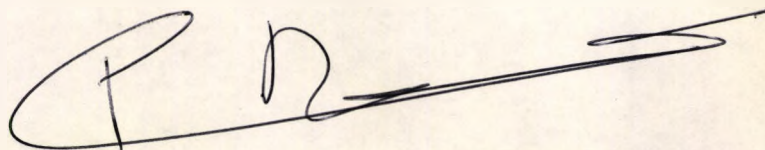
**Copyright:** City Research Online aims to make research outputs of City, University of London available to a wider audience. Copyright and Moral Rights remain with the author(s) and/or copyright holders. URLs from City Research Online may be freely distributed and linked to.

**Reuse:** Copies of full items can be used for personal research or study, educational, or not-for-profit purposes without prior permission or charge. Provided that the authors, title and full bibliographic details are credited, a hyperlink and/or URL is given for the original metadata page and the content is not changed in any way.

# Structure and expressive timing in music performance

Submitted by P.W.M. Desain  
to the City University, London  
as a thesis for the degree of Doctor of Philosophy  
in Music  
July, 1991

I certify that all the material in this thesis which is not my own work has been identified  
and that no material is included for which a degree has been previously been conferred upon  
me.

A handwritten signature in black ink, appearing to read 'P.W.M. Desain', written in a cursive style.

# Contents

## Preface

### The Quantization Problem: Traditional and Connectionist Approaches.

Desain, P. & H. Honing. (1991). The Quantization Problem: Traditional and Connectionist Approaches. In M. Balaban, K. Ebcioğlu & O. Laske (Eds.) *Musical Intelligence*. Menlo Park: The AAAI Press. (forthcoming).

### Quantization of Musical Time: A Connectionist Approach.

Desain, P. & H. Honing. (1991). Quantization of Musical Time: A Connectionist Approach. In P.M. Todd and D. G. Loy (Eds.) *Music and Connectionism*. Cambridge, Mass.: MIT Press. (forthcoming).

### A Connectionist and a Traditional AI Quantizer, Symbolic versus Sub-symbolic Models of Rhythm Perception.

Desain, P. (1991). A Connectionist and a Traditional AI Quantizer, Symbolic versus Sub-symbolic Models of Rhythm Perception. In I. Cross (Ed.) *Proceedings of the 1990 Music and the Cognitive Sciences Conference*. London: Harwood Press. (forthcoming).

### A (De)composable Theory of Rhythm Perception.

Desain, P. (forthcoming). A (De)composable Theory of Rhythm Perception. *To appear in Music Perception*.

### Lisp as a Second Language, Functional Aspects.

Desain, P. (1990). Lisp as a Second Language. *Perspectives of New Music*. Vol. 28(1):192-222.

### Parsing the Parser, A Case Study in Programming Style.

Desain, P. (1991). Parsing the Parser, A Case Study in Programming Style. In *Computers in Music Research*. Vol. 2 :39-90.

### Autocorrelation and the Study of Musical Expression.

Desain, P. & S. de Vos. (1990). Autocorrelation and the Study of Musical Expression. In *Proceedings of the 1990 International Computer Music Conference*. San Francisco: Computer Music Association. 357-360.

### Tempo Curves Considered Harmful.

Desain, P. and H. Honing. (forthcoming). Tempo curves considered harmful. *To appear in Contemporary Music Review*.

### Towards a Calculus for Expressive Timing.

Desain, P. and H. Honing. (forthcoming). Towards a Calculus for Expressive Timing. *Submitted to Psychology of Music*.

## Statement of co-authorship

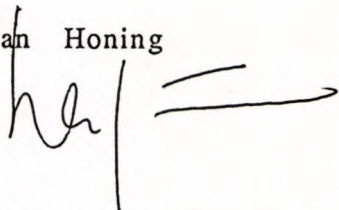
The first two articles: "The Quantization Problem: Traditional and Connectionist Approaches" and "Quantization of Musical Time: a Connectionist Approach" are 70% P. Desain's work. The last two papers: "Tempo Curves Considered Harmful" and "Toward a Calculus for Expressive Timing" are 70% H. Honing's work and 30% P. Desain's work. For article number 7 "Autocorrelation and the Study of Musical Time" Desain developed the initial ideas, and S. de Vos designed the programs and the formula for partial autocorrelation, reflecting an equal division of the work.

Signed Utrecht, 7 september 1991

Peter Desain

A handwritten signature in black ink, consisting of a large, stylized 'P' followed by a series of horizontal strokes.

Henkjan Honing

A handwritten signature in black ink, featuring a cursive 'h' followed by a vertical line and a horizontal line.

Siebe de Vos

A handwritten signature in black ink, starting with a large, looped 'S' followed by a vertical line and a horizontal line.



# Abstract

The research presented is based on the study of music, the study of mind and the study of machine. Most of it deals with the low level rhythm perception mechanisms that process the perceived time intervals roughly corresponding to a note, and infer rhythmic categories, a sense of tempo and local deviations thereof, and expectations for the events to come.

This main thread of the work consists of the first four articles. They deal with different methods of quantization, propose a connectionist model and compare it to a symbolic one, and elaborate an expectancy measure that can be used as the basis for higher level rhythm perception processes. Thus starting from the perspective of a technical tool the work develops towards cognitive theories of human rhythm perception.

Then two papers follow a side-track that deals with AI programming in LISP. They explain the programming style used in the various articles and apply it in a case study to a large computational model.

The next three articles follow a main line that studies the expressive timing signal extracted by quantization. The first of these tries a statistical approach, analyzing the tempo variations in a performance using autocorrelation. The second, in the form of a fictitious story, warns against a simplistic notion of tempo curves by showing that any transformation or manipulation based on the implied characteristics of such a notion is doomed to fail. The third concludes the thesis, linking expressive timing and structure in an attempt to enable transformations of expression. The motivation for this last article is the need for measurement instruments and tools that can cope with the complexity of performance data and are much more sophisticated than tempo curves. It assumes an intimate link between expression and structure - or rather the foundation of the concept of expression on structural musical units.

# Preface

The research to be presented is essentially multi-disciplinary. It is based on the study of music, the study of mind and the study of machine. Nowadays each of these topics is linked to the other in various research disciplines. In computer music, ways to design machines to make music are explored. In music cognition, mental processes that perceive and apprehend music are investigated. In artificial intelligence the mind is approached as a machine - and machines are built to learn more about mind. Though the articles in this thesis focus each on a narrow topic, the research in these various domains forms the ground work on which it was possible to base this contribution.

In this preface I will show how the articles relate. I will also sketch the research paradigm that underlies this work - and deal with some controversies within that paradigm. Because the articles that form the heart of this thesis are written in quite precise and technical terms, and intentionally so, I will feel free to explore some more intuitive insights and to ventilate speculative ideas here. These thoughts, arising sometimes in discussions with colleagues, motivated me in the present research and form the inspiration for future work. I will first give some observations on the methodology that is based on the relations between music, mind and machine and then focus on the subject of this thesis: expression and structure in music.

## Music

Music never functions in a vacuum: it is carried by pressure waves in air. It can be studied as a sound signal that is emitted by a performer, travels through the air and is picked up by the listener's ears. This is the domain of acoustics, (the behaviour of the sound in space) and of psycho-acoustics (the conversion of pressure waves into musical percepts in the ear and in parts of the brain) which form complex fields of study in their own right. Music also never functions in a social and cultural vacuum. The perception of music can be changed in subtle ways by the visual impression of the performer, sociological factors, fashion, the listener's associations and a multitude of other factors studied in sociology, anthropology and ethnomusicology. All these presuppose the human ability to remember and recognize musical fragments or even a specific style, composer or performer, high level tasks investigated in psychology. Those musical styles and periods can also be explored in their own right, independent of cognition, as is done in musicology.

In this thesis all these issues, however interesting, are ignored. And to name another severe restriction: the examples used all stem from the western classical music.

But what is this research about, if so much is excluded? Most of it deals with the early rhythm perception mechanisms that process the perceived time intervals roughly corresponding to a note, and infer rhythmic categories, a sense of tempo and local deviations

thereof and expectations for the events to come. All the caveats were necessary because even the most simple questions about the form, role and function of expressive timing and tempo are quite difficult to tackle. That is also one of the reasons why the subject was approached in a rather technical way - based on simple elapsed time intervals between performed note onsets.

One can question whether with such restrictions to simple measurable quantities, one can still study meaningful matters. Or does the scientific, technological approach kill the magic of music? In a sense it does, and in a sense it doesn't. It does kill magic by refusing to assume any direct communication of human emotion from performer to listener by wizardry. But by assuming that, if such things are communicated at all, they must be communicated via the music signal itself, it takes the unraveling of this signal as its primary goal. Technology then becomes very helpful, and the discovery of subtle and intricate patterns of musical performance, the almost unbelievable consistency in fraction-of-second timing of human performers and the delicate ways in which musical structure is communicated exposes a great wealth of wonder and magic. Besides, music always has been filled with techniques and technology, aiming towards the mastery of it, be it in instrument building, composition or control over the instrument in performance.

This reliance on objective measurements does not give researchers the right to dismiss other realities. When a music teacher teaches a student how to play e.g. 'sadly' certainly something is conveyed. And the fact that it might be difficult to discover this sadness in the measured musical signal does not mean it is not there. More often than not performers know very well what happens in the music even if they state it in unobservable terms. It is the researcher's task to make sense out of it in objective ways. Once some progress is made in that direction, what happens in music can still only be described - not prescribed. More than once researchers have made serious mistakes in this matter - to the point of circularity, like Manfred Clynes (1987) who claims that a performer who does not play according to his theory of the 'composers pulse' does not play well.

## Mind

The modern cognitive and computational approach to the musical mind differs quite a lot from the older psychology of music, in that it develops formalized, testable models of aspects of the musical mind instead of intuitive, metaphorical concepts. Nevertheless there is an old metaphorical approach to the musical mind that has reappeared recently and that is, in my opinion, dangerous and misleading. Before I present an alternative I will explain this theory. It is based on an apparent similarity between musical and physical motion. Helmholtz (1885) was already quite explicit in his appreciation of the similarities, and even attributes to them a central role in the evocation of emotion.



..., it becomes possible for motion in music to imitate the peculiar characteristics of motive forces in space, that is, to form an image of the various impulses and forces which lie at the root of motion. And on this as I believe, essentially depends the power of music to picture emotion. (Helmholz, quoted in Todd, 1989)

But he still pictures music that chooses to resemble physical motion - not in any magical way forced to do so. Todd starts to blur the distinctions involved:

*For example, it seems intuitively appealing that by increasing the energy the maximum velocity or tempo increases. Every musician knows that the faster the tempo the more work they have to do. It may be that the 'energy' is also salient to the listener which could make a contribution to affect. (Todd, 1989 p 156)*

However appealing these similarity-based theories might be to naive bystanders, there is no evidence whatsoever that a walnut is good for the brain because it looks like one (to name a theory built on the same foundations). I do not object to the testing of the possible use of square root functions to model musical ritards, or the use of constant acceleration of musical velocity in modelling expressive timing - just because both ideas happen to describe physics of falling objects and constant gravity as well. However, I do object to the idea that physical motion is more than a mere metaphor in these matters. Some authors move from the simplicity of falling physical bodies to moving bodies of human performance and explain the similarity as embodiment of musical thought, and thus propose a healthy alternative to the wholly mentalistic approach AI researchers tend to take. Furthermore this approach is again open for scientific inquiry (Clarke, forthcoming; Davidson, 1991) and my criticism is not aimed in that direction.

To understand my objections to the reliance on metaphor it is important to note that in the search for simple similarities, alternative explanations of the phenomena are easily overlooked, as is shown by some studies of the final ritard. This large deceleration at the end of a piece is often observed to have a certain form (a square root curve) and it can indeed be modelled as the speed of a mass under a constant deceleration, a constant braking force (Kronman & Sundberg, 1987).

However, there is another explanation possible that is based on the structure of the music and the architecture of temporal perception itself. For a large tempo change such as a final ritard to be still perceivable as a slowing down, it should not slow down too fast, otherwise the rhythmic categories will not be communicated intact and tempo cannot be perceived. Any quantizing and tempo tracking model like the models proposed by Longuet-Higgins (1976) and Desain & Honing (1989) will predict a form of maximal deceleration that can still be followed. It might be that a good model can indeed predict - by its

limitations - the limits of acceptable rubato and final ritard. Because of the nature of these models they will also predict that a) these limits are different for various rhythmic structures and b) the slowing down might well be required to work in stepwise fashion - because the models propose separate tempo tracking mechanisms on different levels of the metrical hierarchy. Both predictions are consistent with the findings that music from different composers or style periods require different final ritards to work well musically, and some evidence that in final ritards there is indeed a tendency to decrease tempo in a stepwise manner (Clynes, 1987). Such observations immediately show the importance of investigating this possible explanation further. If it can be shown to hold, it will be a much more attractive explanation than the physical motion theory because it explains properties of good music performance directly from the musical material and from the perceptual processes themselves. I am convinced that music is based on, plays around with, and makes use of the architecture of our perceptual systems much more than that it imitates our physical surroundings.

### Machine

Artificial Intelligence research (AI for short) has always had two faces, a technological and a cognitive one. The first solely strives to design technical systems (machines) that behave intelligently and the latter seeks to make testable, formal models of intelligent human behaviour. My own orientation towards Artificial Intelligence in this thesis is mainly motivated by the possible understanding of human cognition that it might bring. Curiosity about the musical mind is the main driving force and practical applications e.g. for music production that may come out of the research are considered a by-product. The occurrence of the word *system* or *model* in publications gives one a fair guess about the approach a researcher takes. A problem arises when the terms (and the orientations) are confused: a successful system only has to behave up to input-output specification but the internal mechanisms of a successful model are supposed to tell us something about reality. It may not be a problem in the initial stages of the research that this issue is sometimes unclear, but finally, if one is to learn something about human intelligence, it must be made explicit how the model relates to the phenomenon modelled. This testing of artificial intelligence models, or even stating the models in ways that generate testable predictions, is a field that is barely developed. There is a huge gap between experimental psychology, with its sophisticated tools for testing simple processes, and cognitive science which has hardly any tools for testing their more complex models. Even the way in which computational models can be described such that it is clear what it is that is modelled, and what is simple implementation detail, is problematic, especially since programming languages do not support the specification of those issues. However, an adequate programming style, a clear description of these issues and the publication of the program in



the form of a micro version helps in determining the value of an algorithm as a model for a cognitive function. Artificial Intelligence at its worst can be seen in articles that make anthropomorphic claims about unpublished programs. The laborious so-called rational reconstruction of those programs by others to check the claims is then the only remaining route to scientific progress (Ritchie & Hanna, 1990).

Artificial Intelligence is not a very homogeneous domain. At present there is within AI a clash of two competing paradigms: Connectionism and the Symbolic paradigm. Since in my own research the use of connectionist and/or symbolic representations is a recurrent theme, it is good to dwell on both a bit more. The so called Good Old Fashioned Artificial Intelligence, (i.e. the symbolic approach) has established itself firmly as a research methodology in the past decades. The methods and tools it uses are symbolic, highly structured representations of domain knowledge and transformations of these representations by means of formally stated rules. At the heart of this methodology is the use of symbols that have no content in themselves: information processing is of a syntactic nature. It is easy to misinterpret the behaviour of such a system since the symbols often carry suggestive name tags that may seduce one into attributing more sense, more intelligence, to the program than is actually implemented in the rules themselves. One has to realise that some of these clever programs actually do not achieve very much, being based on a very smartly developed knowledge representation that solved, or evaded the problem beforehand. Any extension of these systems, however small, or any attempt to generalize the results is doomed to fail because the knowledge representation is designed just to give ad-hoc solutions to a small set of problems. I feel that this approach carries the symbolic approach to its ridiculous extreme.

However, other kinds of symbolic AI have contributed more or less generalizable theories to the field, and have proposed models of human information processing. These rule-based theories can function as abstract formal descriptions of aspects of cognition. Some authors even go beyond that and claim that mental processes are symbolic operations performed by mental representations of rules.

Until the connectionist paradigm emerged there was no real alternative to this view. In the new paradigm the departure from a reliance on the explicit mental representation of rules is central, and the approach to cognition is fundamentally different. Connectionism opens the possibility of defining models which have characteristics that are hard to achieve in traditional AI, in particular robustness, flexibility and the possibility of learning. The connectionist boom has produced lots of interesting work, although many researchers have lost their critical attitude impressed as they were by the good performance of some prototypical models. This has resulted in thousands of papers presenting more and more examples of problems

that could be learned by a neural net, the proof being the simulation of such. Levelt, bothered by this waste of effort, concluded that connectionist models are, *mutatis mutandis*, as handy as a city map on a 1:1 scale (Levelt, 1989). Indeed more study is needed of the limitations of these models. A connectionist model that 'works' well, constitutes in itself no scientific progress, if questions like the scalability to larger problems and the dependency of the model on a specific input representation, cannot be answered.

The theoretical observation that a connectionist system can simulate any symbolic computation machine (is *Turing machine equivalent*) and vice versa tends to dismiss the relation between the paradigms as a non-issue. I think that the language in which problems are stated and the level at which research is conducted is of major importance - each language obscures some matters while clarifying others. A related idea about the relation between the paradigms is the presentation of Connectionism as an implementation level theory, which can coexist with a more abstract symbolic theory on a higher level. This view is often associated with the claim that connectionism is superior to the symbolic approach because the computational units resemble cells found in the brain (the term 'neural networks' stems from that postulated isomorphism). This has to be rejected firmly. The simple computational method used in connectionism is miles away from true biological modelling and the chosen computational level of abstraction can never be a ground for superiority.

Against the background of this debate within AI and cognitive science on the role of connectionist models, some researchers have concentrated on a technical examination of the weak and strong points of both symbolic and neuro-computing, to be able to combine them in so-called hybrid systems. They claim that because symbolic computing is best suited for higher level functions such as reasoning and planning, and neuro-computing is more applicable for low-level, perceptual and classification tasks, systems containing modules from both paradigms should be devised. This approach is not free of problems, to put it mildly. At its worst it can be described as: 'we do not understand how neural nets work, we do not know how rule-based systems work, let's combine them and see what happens'. Such a pragmatic approach can only obscure the real issues.

There is, however, another way to deal with the challenge of connectionist work. By comparing both paradigms one quickly discovers that the formalisms used are often of such an idiosyncratic nature that it is impossible to make claims about the behaviour of models from both paradigms. Concentrating on general abstract descriptions of behaviour then becomes a very fruitful activity. It yields new ways to look at the connectionist and the symbolic models and to characterize them further - a positive contribution in itself. For example, consider the benefits of describing the



input-, state- and solution spaces, a trivial exercise for connectionist systems. In symbolic systems these constructs often remain hidden in the program code and are not made explicit in the articles, but they can help enormously in characterizing such systems. These analyses also yield ways to describe connectionist systems on a more general level than simulation runs can. One such point that is often neglected is the representation:

*[for most aspects of connectionist modelling] there exists considerable formal literature analyzing the problem and offering solutions. There is one glaring exception.: the representation problem. This is a crucial component, for a poor representation will often doom the model to failure, and an excessively generous representation may essentially solve the problem in advance. Representation is particularly critical to understanding the relation between connectionist and symbolic computation, for the representation often embodies most of the relation between a symbolically characterized problem (e.g. a linguistic task) and a connectionist solution. (Smolensky, 1990)*

In representation issues the symbolic paradigm has, because of its very nature, much to offer to connectionism. I think a combined study of both paradigms might overcome the controversy. In the end the differences may turn out not to be that essential. One example supporting this view is the research that showed that a certain kind of network can still support modularity and recursive (de)composition of constructs (Pollack, 1990) - a central issue in symbolic AI. However, at the moment we are still confronted with a new and hardly understood paradigm.

I expect further progress from the elaboration of continuous knowledge representations, the most eye catching feature of connectionism in comparison to the symbolic paradigm that uses discrete concepts (be it memory locations, categories, inference operations or production rules). Continuous learning curves are a *sine qua non* of multi-layer learning algorithms. And the behaviour of neural nets has been described, with great benefit, as continuous over time, making the whole apparatus of partial differential equations applicable. Simulation of such networks on computers is done by applying *time-sampling* as an approximation to the time-continuous change of state in a network. It might prove beneficial to carry this idea to its extreme. It is strange indeed that the discreteness of the individual network cells has not yet been considered to be a space-sampling of a basically space-continuous computing model. Instead of a vector space, the input, output and state space of the system then become function spaces. It might even be possible to consider the cell layers of a network as space-sampled, continuous, two dimensional computation. Instead of a network we can

then metaphorically talk about a lump of 'computing material'. It might well be that the analytical methods available for systems of differential equations for continuous functions, and sampling theory, can thus again be applied to connectionist systems, and produces results for unanswered questions like the number of hidden layers or the number of cells in those layers needed for a certain task.

This finalizes some methodological considerations. I hope to have sketched some of the ways in which the concepts of music, mind and machines interrelate and how these relations can be a fruitful basis for research. After these detours we now have to home in on the topics of this thesis: expression and structure in music.

### **expression**

Perhaps contrary to common usage, the word expression in this thesis does not denote what music expresses to the individual listener. All the links to musical affect, to emotion and even to aesthetics are considered too complex to tackle before more mundane issues are understood. Expression is assumed to be a syntactical concept - dealing only with the form of the music. In the first stages of the research expression was defined as the pattern of deviations of attributes of performed notes from their value notated in a score. Everything added by the performer to the score, all deviations from a strict mechanical performance, was termed expression. This definition, however useful in the initial study, soon lost its attractiveness. In general listeners can appreciate expression in music performance without knowing the score and a full reconstruction of the score in the form of a mental representation is impossible. Take for instance the notion of the loudness of notes. Should a listener be required to fully reconstruct the dynamic markings in the score before it is possible to appreciate the deviations from this norm as expressive information added by the performer? Such a nonsensical conjecture indeed follows from a rigid definition of expression as deviation from the score. Seashore was a bit more careful (albeit a bit more vague too) when he defined expression, independent of a score, as:

*artistic deviation from the fixed and regular: from rigid pitch, uniform intensity, fixed rhythm, pure tone . (Seashore, 1938, quoted in Todd, 1989)*

It is possible to find more elaborate ways of defining expression on the basis of performance information only. In later stages of the research this was achieved by basing expression on the notion of structural units, using this working definition: expression within a unit is the pattern of deviations of its parts with respect to the norm set by the unit itself. Take e.g. a metrical hierarchy of bars and beats. The expressive tempo within a bar can be defined as the pattern of deviations of the tempo of each beat from the tempo of the bar. Or take the loudness of the individual notes of a chord. The dynamic expression within a chord can be



defined as the set of deviations of the loudness of the individual notes from the mean loudness of the chord. Using this definition, expression can be extracted from the performance data itself, taking more global measurements as reference for local ones, based on the concept of known units. Thus the structural description of the piece becomes central, both to establish the units which will act as a reference and to determine the sub-units that will act as atomic parts whose internal details will be ignored. A similar definition works well for the expression carried by the difference of two voices or formed by the difference between e.g. a theme and a variation. Expression between two units is defined as the pattern of deviations of their parts with respect to a norm set by both units themselves. The norm could be some kind of average, or even one of the units themselves. E.g. the timing of the accompaniment could be taken as the reference when considering the expression carried by the timing lead the melody voice has over it in ensemble timing. In taking on this intimate link between expression and structure - or rather the foundation of the concept of expression on structural units - the nature of the structural description becomes a crucial concern.

### Structure

Structure in music is not a simple concept, because of the multitude of structural descriptions in use. Let us start with hierarchical structures like metre, rhythmic grouping and phrasing in which the structural links are *part-of* relations. These overlaying structural analyses, concerned with different aspects of the piece, may violate each others boundaries - like a phrase ending in the middle of a measure. There can be ambiguity: multiple mutually exclusive analyses or interpretations of the same aspects of a piece. There may be local violation of otherwise hierarchical structure, like two overlapping phrases (a situation seldom encountered in linguistics). The need for local structural relations like grace notes and other ornamentations is obvious too. These can be described by a part hierarchy, but there are also structural relations that cannot be treated likewise, like symmetrical associations between recurrent motives. Besides these collections of musical events, and the simple relations between them, we need formalization of the various rhythmic, melodic and harmonic roles that can be ascribed to such collections. I think that the complexity sketched mirrors the complexity found in the expressive signal itself, since the various structures are the source of expression and are conveyed to the listener by that means. A full theory of expression should be able to link these various structural descriptions to the components of the expressive signal. This thesis can only offer a small contribution to this long-term aim.



### Contents of the thesis

Because this thesis consists of a number of published articles, that have to be more or less independent, some overlaps are unavoidable. Each article has several links to the others which makes it quite difficult to impose a linear order. The present ordering highlights two main lines, and a side track. The first main thread consists of the first four articles. It deals with different methods of quantization, proposes a connectionist model and compares it to a symbolic one, and elaborates an expectancy measure that can be used as a base for higher level rhythm perception processes. Then two papers follow a side-track that deals with AI programming in LISP. They explain the programming style used in the various articles. The next three articles follow a main line that studies the expressive timing signal extracted by quantization. The first one tries a statistical approach. The second warns against a simplistic notion of tempo curves. The third concludes the thesis, linking timing and known structure in an attempt to enable transformations of expression. Details of publication can be found in the contents.

### The Quantization Problem: Traditional and Connectionist Approaches

This paper constitutes a first attempt to understand quantization and the research done in this field. It still approaches the problem from the perspective of a technical tool. It was presented at the *AI and Music Workshop* in 1988 in Cologne and will appear in a book on Musical Intelligence. Traditional and AI methods for quantization are explained and compared. Simplified algorithms of the described methods are included as micro versions.

### Quantization of Musical Time: A Connectionist Approach

In this paper a connectionist model is elaborated that converges from performed time intervals to an equilibrium state in which the score durations can be read out. It was published in the *Computer Music Journal* and will appear in a book on neural networks and music. For this book an addendum was written containing a mathematical description of the network plus some material from a presentation at the *International Computer Music Conference* in 1989 in Columbus, Ohio.

### A Connectionist and a Traditional AI Quantizer. Symbolic versus Sub-symbolic Models of Rhythm Perception

Two incompatible quantization models, namely the Longuet-Higgins Musical Parser and the Desain & Honing connectionist quantizer, were studied further in order to find ways to compare and evaluate them. Different perspectives to describe their behaviour were developed. This paper was presented partly at the *Horssen Workshop on Rhythm Perception and Production* in 1990 and in full at the *Music Cognition Conference* in Cambridge later that year. It will appear in the proceedings thereof.

### A (De)composable Theory of Rhythm Perception

Out of the study of the two incompatible models arose a measure of the expectancy of events projected into the future by a complex temporal sequence. It can be decomposed into basic expectancy components projected by each time interval implicit in the sequence. A preliminary formulation of these basic curves is proposed and the (de)composition method is stated in a formalized, mathematical way. The resulting expectancy of complex temporal patterns is believed to be useful to model topics such as clock and meter inducement, rhythmicity, and the perceived similarity of temporal sequences. This theoretical paper will appear in *Music Perception*.

### Lisp as a Second Language, Functional Aspects

Looking at the LISP programs emerging from the computer music community, the old imperative style can often be seen between the lines of LISP code. It is a pity to neglect the elegant ways of expressing algorithms in LISP, and doing so will often result in disappointing performance and maintainability. In this article the functional style of programming is explained and illustrated with examples from computer music. It appeared in *Perspectives of New Music* and was used as material for several programming workshops for composers.

### Parsing the Parser, a Case Study in Programming Style

This paper takes, as an example, the musical parser designed and described by Longuet-Higgins and re-implements it in a functional programming style in LISP. This yields a micro version that makes the theoretical issues stand out more clearly. It was published in *Computers in Music Research*.

### Autocorrelation and the Study of Musical Expression

In this paper a method was designed to analyze the tempo variations in a performance using autocorrelation. Peaks in the autocorrelation function are interpreted as periods of repeated components in the musical structure. Partial autocorrelation is used to remove the multiples of a fundamental period. It was presented at the *International Computer Music Conference in Glasgow* in 1990 and appeared in the proceedings thereof.

### Tempo Curves Considered Harmful

This fictitious story shows that we have to be aware of the notion of a Tempo Curve, because it lulls its users into the false impression that it has a musical and psychological reality. There is no abstract tempo curve in music, nor is there a mental tempo curve in the head of a performer or listener. It shows that any transformation or manipulation based on

the implied characteristics of such a notion is doomed to fail. It will appear in *Contemporary Music Review* and it will be presented at the *International Computer Music Conference* of 1991 in Montreal.

#### Towards a Calculus for Expressive Timing

This paper is an attempt to identify ways in which structural knowledge can be used to enable transformations of musical performances that make musical sense. The motivation for this work is the need for measurement instruments and tools that can cope with the complexity of performance data and are much more sophisticated than tempo curves. This paper is submitted to *Psychology of Music*.

#### **Communication**

The working title of this thesis was 'The Communication of Structure by Expressive Timing in Music Performance'. The link between structure and expression was indeed one of the basic hypotheses underlying most of the work. However, I have not yet been able to deduce effective procedures to generate timing from structure or by which a listener can infer structure from timing. This was partly because a lot of ground work and tool building had to be done first. That work is reported on in this thesis. The recent work of Longuet-Higgins & Lisle (1989), Todd (1989) and Drake & Palmer (in preparation) indicate that there might be ways in which the communication of structure by expressive timing can be formalized. However, no clear picture that deals in a unified way with all kinds of structure (metrical, rhythmic, phrase, local surface etc.) has emerged yet. I hope that my work has produced results that may in the long run contribute to the understanding of that issue.



### Acknowledgements

Besides the colleagues that helped me in the various stages of the research, and who are credited in the appropriate articles, I want to express my gratitude to Eric Clarke. He did a wonderful job, as a supervisor of this thesis, a close collaborator and last but not least as a friend. Henkjan Honing has been a great companion, from our first wavering attempts at a cooperative composition project in 1984 to the present solid patterns of collaboration in research. At Nijmegen University, Gerard Kempen was the first to put me on the track of AI research. At the Utrecht School of the Arts, Johan de Biggelaar and Ton Hokken shared the vision of a Dutch research institute for AI, Art and Cognition, which proved to be a mirage. I am thankful for their effort in the quest for research facilities during those years. I would like to thank Dirk-Jan Povel, for supporting my future work in the field of music cognition.

### Copying

I grant powers of discretion to the University Librarian to allow this thesis to be copied in whole or in part without further reference to me. This permission covers only single copies made for study purposes, subject to normal conditions of acknowledgement.

A handwritten signature in black ink, appearing to be 'T. R.', written over a faint, illegible stamp.

## References

- Chandrasekaran, B. (1990) What Kind of Information Processing is Intelligence? A Perspective on AI Paradigms and a Proposal. In T. Partridge and Y. Wilks (Eds.), The foundations of artificial intelligence, a sourcebook. Cambridge: Cambridge University Press.
- Clarke, E.F. (forthcoming) Generativity, Mimesis and the Human Body in Music Performance In I. Cross (Ed.) Proceedings of the 1990 Music and the Cognitive Sciences Conference. London: Harwood Press.
- Clynes, M. (1987) What can a musician learn about music performance from newly discovered microstructure principles (PM and PAS)? in A. Gabrielson (Ed.) Action and Perception in Rhythm and Music, Royal Swedish Academy of Music, No. 55.
- Davidson, J. (1991) The Perception of Expressive Movement in Music Performance. Ph.D. thesis, City University, London.
- Desain, P. and H. Honing (1989) Quantization of Musical Time: A Connectionist Approach. This volume.
- Desain, P. (1991). A Connectionist and a Traditional AI Quantizer, Symbolic versus Sub-symbolic Models of Rhythm Perception. This volume.
- Drake, C & C. Palmer (in preparation) Recovering Structure from Expression in Music Performance.
- Gutknecht, M., R. Pfeifer (1990) An Approach to Integrating Expert Systems with Connectionist networks. AICOM vol 3(3).
- Kronman, U and J. Sundberg Is the Musical Retard an Allusion to Physical Motion? In A. Gabrielson (Ed.) Action and Perception in Rhythm and Music, Royal Swedish Academy of Music, No. 55.
- Longuet-Higgins, H.C. (1976) The Perception of Melodies. Nature 263.
- Longuet-Higgins, H.C. & E.R. Lisle (1989) Modeling Music Cognition. *Contemporary Music Review* 3(1).
- Levelt, W.J.M. (1989) De Connectionistische Mode, Symbolische en Subsymbolische modellen van Menselijk Gedrag. In C. Brown, P. Hagoort, T. Meijering (Eds.) Vensters op de Geest, Cognitie op het Snijvlak van Filosofie en Psychologie. Utrecht: Stichting Grafiet.
- Pollack, J.B. (1990) Recursive Distributed Representations. *Artificial Intelligence* 46.
- Ritchie, C.D. & F.K. Hanna (1990) AM: A Case Study in AI Methodology. In T. Partridge and Y. Wilks (Eds.), The foundations of artificial intelligence, a sourcebook. Cambridge: Cambridge University Press.
- Smolensky, P (1990) Tensor Product Variable Binding and the Representation of Symbolic Structures in Connectionist Systems. *Artificial Intelligence* 46.
- Todd, N.P. (1989) Computational Theory and Implementation of an Abstract Expression System: a Contribution to Computational Psychomusicology. PhD thesis, University of Exeter.
- Touretzky, D.S. (1990) BolzCons: Dynamic symbol structures in a connectionist network. *Artificial Intelligence* 46.



**THE QUANTIZATION PROBLEM:  
TRADITIONAL AND CONNECTIONIST APPROACHES**  
(revised version<sup>1</sup>)

Peter Desain & Henkjan Honing

Center for Knowledge Technology  
Utrecht School of the Arts  
Lange Viestraat 2b  
NL-3511 BK Utrecht

Music Department  
City University  
Northampton Square  
UK-London EC1V OHB

**ABSTRACT**

Quantization separates continuous time fluctuations from the discrete metrical time in performance of music. Traditional and AI methods for quantization are explained and compared. A connectionist network of interacting cells is proposed, which directs the data of rhythmic performance towards an equilibrium state representing a metrical score. This model seems to lack some of the drawbacks of the older methods. The algorithms of the described methods are included as small Common Lisp programs.

**KEYWORDS**

Quantization, rhythm perception, connectionism, expressive timing.

**1. THE QUANTIZATION PROBLEM**

Musical time can be considered as the product of two time scales: the discrete time intervals of a metrical structure, and the continuous time scales of tempo changes and expressive timing (Clarke 1987). In the notation of music both kinds are present, though the notation of continuous time is less developed than that of metric time (often just a word like *rubato* or *accelerando* is notated in the score). In the experimental literature, different ways in which a musician can add continuous timing changes to the metrical score have been identified. There are systematic

---

<sup>1</sup>*This paper was presented at the first AI and Music Workshop, St. Augustin, Germany in September 1988. It has been updated with references to new work and some material from (Desain & Honing 1991). Micro versions of the main algorithms were added as well.*

changes in certain rhythmic forms e.g. shortening triplets (Vos & Handel 1987) and consistent time asynchronies between voices in ensemble playing (Rasch 1979). Deliberate departures from metricality such as rubato seem to be used to emphasize musical structure, as exemplified in the phrase-final lengthening principal formalized by Todd (1985). Alongside these effects, which are collectively called expressive timing, are non-voluntary effects, such as random timing errors caused by the limits in the accuracy of the motor system (Shaffer 1981), and errors in mental time-keeping processes (Vorberg & Hambuch 1978). These non-intended effects are generally rather small, in the order of 10 milliseconds.

To make sense of most musics, it is necessary to separate the discrete and continuous components of musical time. We will call this process quantization, although the term is generally used to reflect only the extraction of a metrical score from a performance. This quantization process transforms incoming time intervals between subsequent note onsets, i.e. *inter-onset intervals*, into discrete note durations (as can be found in the score) and a tempo factor that reflects the deviation from this exact duration. It is solely based on inter-onset intervals: any other information like note offsets, dynamics and pitch is ignored. The output of the quantization process can serve as input for processes extracting higher level structural descriptions like meter.

Apart from its importance for cognitive modelling, a good theory of quantization has technical applications. It is one of the bottle-necks in the automatic transcription of performed music, and is also important for compositions with a real-time interactive component where the computer improvises or interacts with a live performer. It is indispensable in the study of expressive timing of music for which no score exists.

## 2. TRADITIONAL METHODS

The quantization problem has been approached from different directions, the resulting solutions ranging from naive and inept to elegant and plausible. We will describe here first the methods that construct the solution in a straightforward numerical way.

### 2.1. Inter-onset quantization

This simple method rounds the inter-onset intervals of the notes to the nearest note duration on a scale containing all multiples of a smallest duration (time-grid unit or *quantum*). In Figure 1 an architecture for this method with standard signal processing modules is shown. Note that this method runs in *event-time*: one cycle of processing is done for each new incoming inter-onset interval, resulting in a quantized interval. The module divides the input by the smallest allowed value and rounds it to the nearest integer. It also yields a relative error in proportion to the quantum (between -0.5 and 0.5). When given a list of intervals and a value for the



quantum the method will produce a list of quantized intervals with respect to this quantum. Given the inter-onset intervals of the rhythm of Figure 2, and a quantum of 100 ms (32th triplet at tempo 50), it will result in the list of multiples of this quantum (12 6 3 3 4 3 4 4 6 6 3 3 3 12) which does not represent the right quantization: (12 6 3 3 4 4 4 6 6 3 3 3 12). This method, when it makes a round-off error, will shift the absolute onset of all subsequent notes. When used in polyphonic music, an error in one stream of notes will permanently de-synchronize it with respect to the other streams.

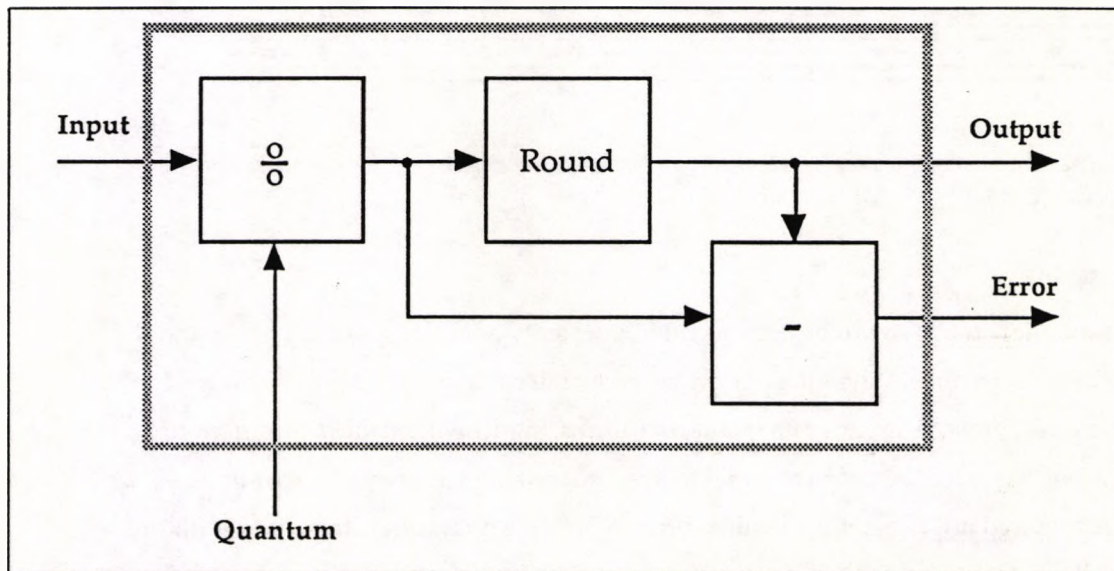


Figure 1. Inter-onset quantizer.

## 2.2. Onset quantization

At first sight, quantizing the absolute onsets of the notes themselves, instead of the inter-onset intervals, will be a solution to the de-synchronization problem. This method simply maps each onset-time to the nearest point in a fixed grid with a resolution equal to the quantum. Small but consistent deviations in the inter-onset intervals, as occur in slight tempo fluctuations, will add-up and produce an onset-time deviation that is the sum of all previous interval deviations. So this method is more sensitive to small tempo fluctuations than inter-onset quantization. Occasionally an onset-time will topple over the boundary between two grid points and the note will not be quantized correctly, but the quantized data will not be permanently de-synchronized.

Commercially available sequencer and transcription software packages use this simple onset quantization method. They cannot notate a non-trivial piece of music without errors (see Figure 2). This is not surprising, considering the large deviations of up to 50% and the ambiguity that has to be dealt with, especially in the case where both binary and ternary divisions are present. Most of these packages force the interpreter to play along with a metronome to give an

acceptable result, or require a precise tuning of parameters (e.g. are triplets allowed) for different sections of the piece.

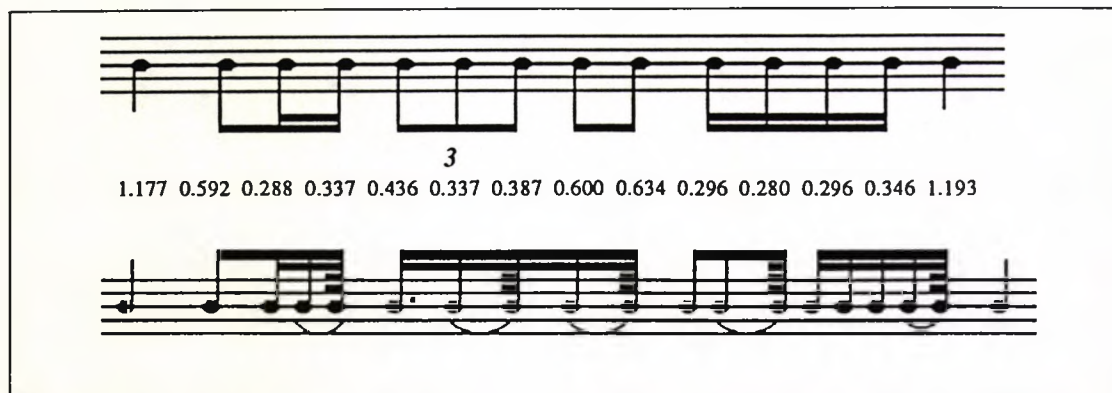


Figure 2. Played score (performance inter-onset intervals in seconds) and its quantization by a commercial package (using a resolution of 1/64 note).

### 2.3. Tempo tracking

The methods mentioned above can be enhanced by repeatedly adapting the duration of the quantum to the performance. When the performer accelerates, the onset times will all tend to fall before the grid points. Adapting the quantum (decreasing it) will enable the system to follow the tempo change of the performer and to keep quantizing correctly. This set-up is shown in Figure 3. A required adjustment is calculated that, when the quantum is increased with this value, would have accounted for the interval perfectly. The fastest response possible for the tempo tracker would be to increase its quantum (one interval later) with that proportion. But such a progressive approach may allow the tempo to stray on the first note that is played imprecise. It is rather difficult to design a good control module that adjusts tempo fast enough to follow a performance, but not so fast that it reacts on every 'wrong' note. A common solution is to build in some conservatism in the tempo tracker by using only a fraction of the proposed adjustment. If this fraction, called the adjustment speed, is set to 0.5 the new tempo will be the mean of the old tempo and the proposed ideal.

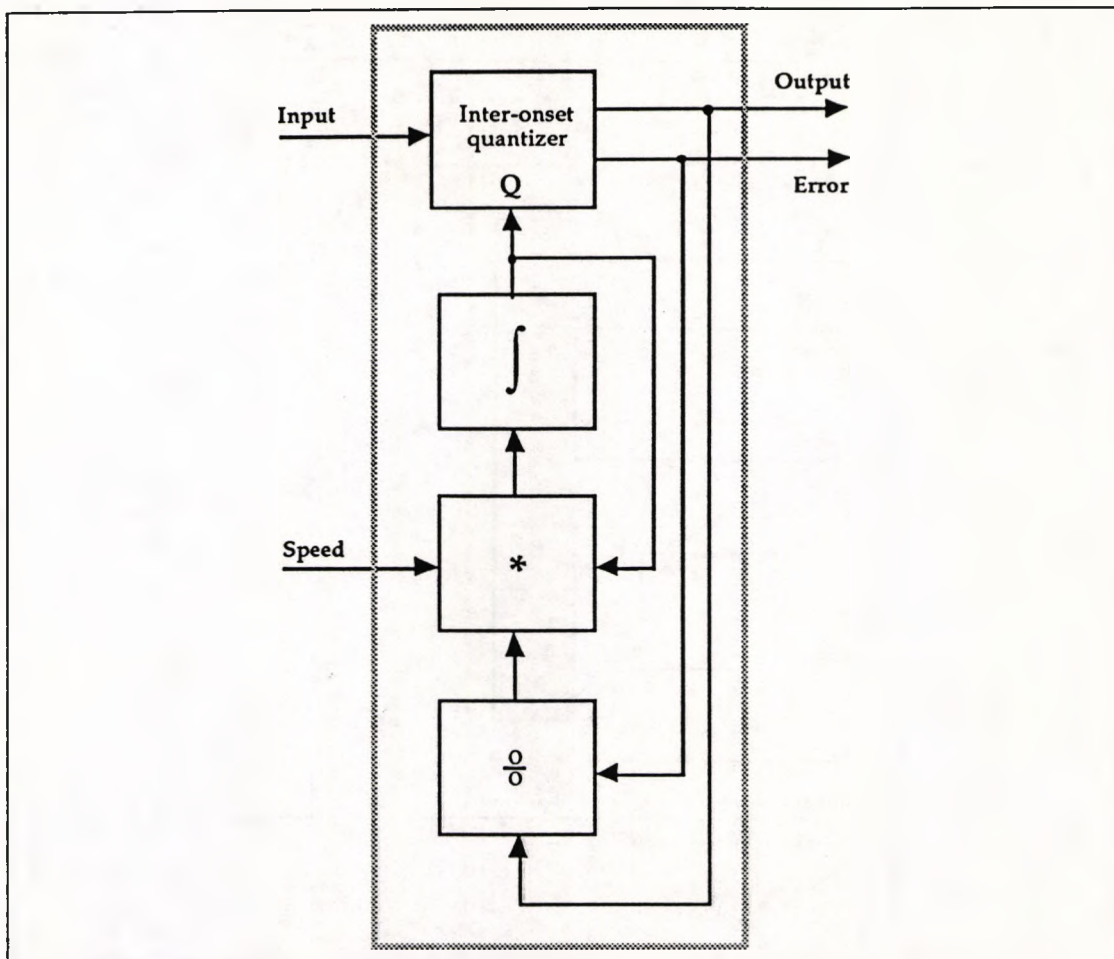


Figure 3. Tempo tracker.

#### 2.4. Tempo tracking with confidence based adjustments

A more sophisticated tempo tracker adapts its tempo only when there is enough confidence to do so. An onset that occurs almost precisely between two grid points will give no evidence for adjusting the tempo (because it is not sure in what direction it would have to be changed). In Figure 4 the details are shown. The quantization error (the difference between the incoming interval and the quantized output of the system) is expressed as a fraction of the quantum. A simple function will calculate a confidence level, on the basis of this error and has a maximum near zero errors. The confidence level also depends on the parameter 'trust', that expresses its sensitivity for errors. If we now use this confidence level as a scale factor for the adjustment speed of the tempo tracker will enhance its performance.

Of course, even this method is vulnerable to errors. Dannenberg and Mont-Reynaud report a 30% error rate for their 'real time foot tapper' which uses a variant of this method (Dannenberg and Mont-Reynaud 1987). This poor performance, considering their careful tuning of parameters and their preprocessing of the musical material (taking only 'healthy' notes into account), is disappointing.



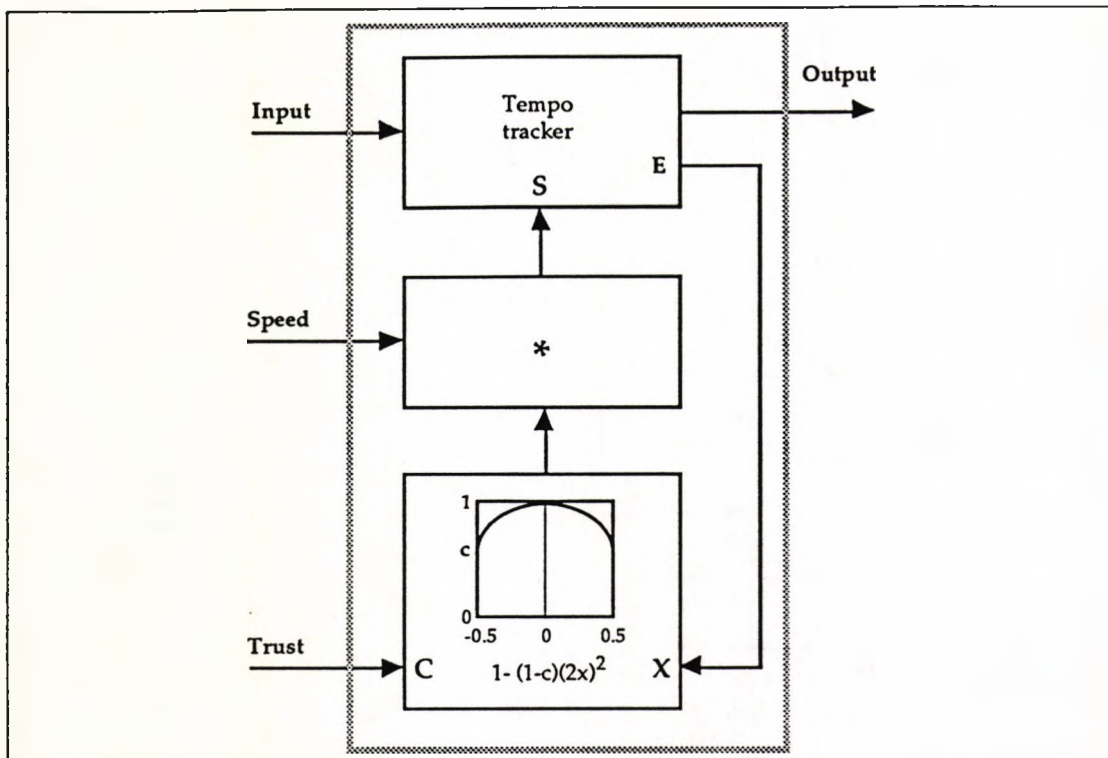


Figure 4. Tempo tracker with confidence based adjustment.

### 2.5. The Algorithm

Since the methods mentioned above can be considered as extensions of each other, the last method can emulate the less sophisticated ones by supplying zero- or one-valued parameters. In Appendix I a micro version of a this general traditional quantizer is given. Experimenting with it, changing parameter values and feeding it with different musical material quickly shows the limitations of these kinds of systems and their lack of robustness.

### 3. USE OF STRUCTURAL INFORMATION

Because of the poor performance of the methods described above, techniques that make use of knowledge of the hierarchical structure of rhythms were proposed for quantization. Longuet-Higgins (1987) describes a hybrid method based on tempo tracking plus the use of knowledge about meter. In this method the tempo tracking is done with respect to a beat (that can span one or more notes). This beat is recursively subdivided in 2 or 3 parts looking for onset times near the start of each part. The best subdivision is returned, but the program is reluctant to change the kind of subdivision at each level. The start and length of the beat or subdivision thereof is adjusted on the basis of the onsets found, just as in the simple tempo tracking method. Next to the quantized results, this program delivers a hierarchical metrical structure. A more detailed study of the behavior of this elegant method can be found in (Desain, 1991b).

### 3.1. The Algorithm

Because Longuet-Higgins published the rather complicated program in POP-2, it seems appropriate to restrict ourselves here to a stripped version (see Appendix II), concentrating only on the essential aspects (see Desain, 1991a). It incorporates the basic ideas about stability of meter, the tolerance with respect to which all decisions on onsets are made, and the beat length that has to be supplied as an initial state of the system. But the analysis of articulation, delivery of metrical structure and the sophisticated tempo tracking is removed. When given the inter-onset intervals of the rhythm in Figure 2, it will result in the correct quantization: (1 1/2 1/4 1/4 1/3 1/3 1/3 1/2 1/4 1/4 1/4 1/4 1).

### 4. KNOWLEDGE BASED METHODS

The automatic transcription project at CCRMA (Chowning et al. 1984) is a particularly elaborate example of a knowledge based system. It prefers simple ratios and uses context dependent information to quantize correctly. This knowledge based approach uses information about melodic and rhythmic accents, local context, and other musical clues to guide the search for an optimal quantized description of the data. Using even more knowledge could possibly contribute to the quantization problem. e.g. harmonic clues could be used to signal phrase endings where the tempo may be expected to decrease at the boundary (phrase final lengthening) and repetition in the music could be used to give more confidence in a certain quantization result. However these knowledge based approaches seem to share the same problems of all traditional AI programs: the better they become, the more domain dependent knowledge (depending on a specific musical style) must be used for further advance, and such programs will break down rapidly when applied to data outside their domain.

### 5. MULTIPLE ALTERNATIVES

All methods above can be enhanced by using them repeatedly on the same data, but with different parameters, searching for the best solution. These analyses could even go on in parallel. Dannenberg and Mont-Reynaud (1987) propose multiple 'foot tappers' all running at the same time. For Chung (1989) the parallel exploration of multiple alternatives is essential. Using Marvin Minsky's paradigm (Minsky 1986) he describes his system as consisting of multiple intelligent agents. These proposals are distributed models with a 'coarse' grain: each part-taking processor consists of a complete traditional symbolic AI program. However, it is possible to use a very fine grained parallelism to tackle the quantization problem, where each processor is very simple, but the interaction between them is crucial.

## 6. CONNECTIONIST METHODS

Connectionism provides the possibility for new models which have characteristics that traditional AI models lack, in particular their robustness and flexibility (see Rumelhart & McClelland 1986). Connectionist models consist of a large number of simple cells, each of which has its own activation level. These cells are interconnected in a complex network, the connections serving to excite or inhibit other elements. The general behavior of such a network is that from a given initial state, it converges towards an equilibrium state. An example of the application of such a network to music perception is given by Bharucha (Bharucha 1987) in the context of tonal harmony, but the connectionist approach has not yet been used for quantization. The quantization model that will be presented now is a network designed to reach equilibrium when metrical time intervals have been achieved, and which converges towards this end point from non-metrical performance data. It is implemented as a collection of relatively abstract cells, each of which performs a complex function compared to standard connectionist models. We will now give a condensed overview of the model.

### 6.1. A Connectionist Quantizer

The proposed network consists of three kinds of cells: the *basic-cell* with an initial state equal to an inter-onset interval, the *sum-cell* to represent the longer time interval generated by a sequence of notes, and the *interaction-cell* that is connected in a bidirectional manner to two neighboring basic- or sum-cells. Figure 5 shows the topology of a network for quantizing a rhythm of four beats, having its three inter-onset intervals set as initial states of the three basic-cells, labeled A, B, and C, and the two summed time intervals A+B and B+C represented by the corresponding sum-cells. There are four interaction-cells connecting cell A to cell B, B to C, A+B to C and A to B+C respectively. Each interaction-cell steers the two cells, to which it is connected, toward integer multiples of one another, but only if they are already close to such a multiple.



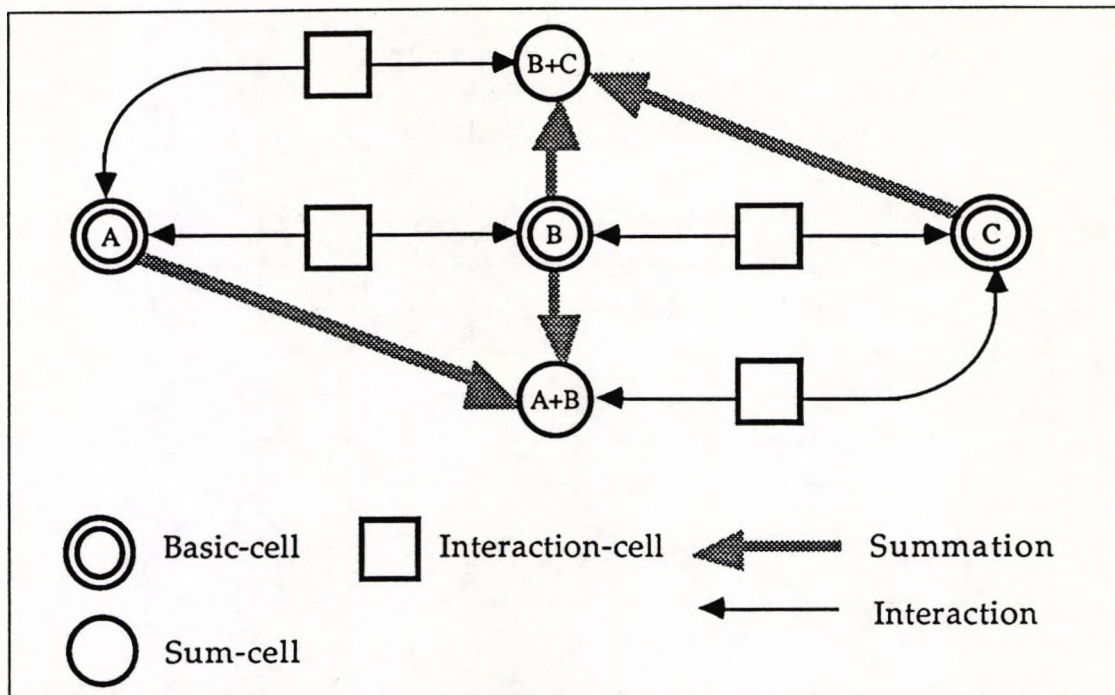


Figure 5. Topology of a connectionist network of a rhythm of three inter-onset intervals.

The two connected cells receive a small change calculated from the application of an interaction function (see Figure 6) to the quotient of their states. One can see that if the ratio is slightly above an integer it will be adjusted downward, and vice versa. The interaction function has two parameters: *peak*, describing how stringent the function requires an almost integer ratio to calculate a correction and *decay*, expressing the decreasing influence of larger ratios. Each cell accumulates the incoming change signals from the connected interaction-cells. The interaction of a sum-cell with its basic-cells is bidirectional: if the value of the sum-cell changes, the basic-cells connected to it will all change proportionally, as well as the other way around. This process is repeated, updating the values of the cells a little bit in each iteration, moving the network towards equilibrium. The system produces promising results. It is *context sensitive*, with precedence of local context. For this reason the example in Figure 2 is quantized correctly (for more details see Desain & Honing, 1991). The system also exhibits *graceful degradation*. When the quantizer breaks down in a complex situation it is often able to maintain musical integrity and consistency at higher levels. The resulting error will only generate a local deformation of the score.

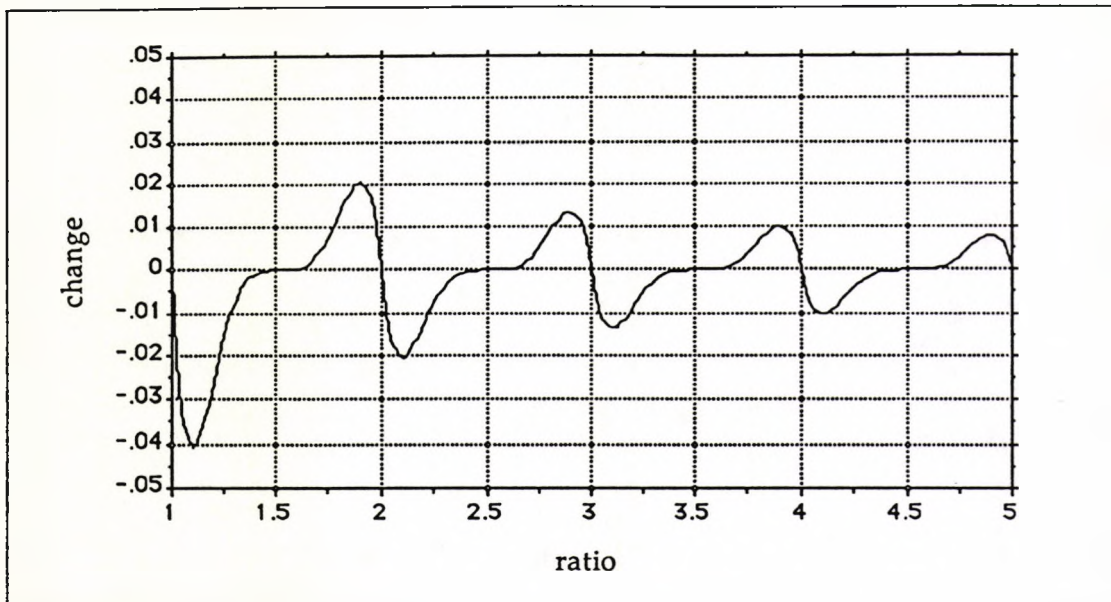


Figure 6. Interaction function.

## 6.2. The Algorithm

A micro version of the program is given in Appendix III. In this program the sum-cells are not represented explicitly, their value is recalculated from the basic-cells. Also the interaction-cells are not represented explicitly. Their two inputs from the connecting sum-cells are calculated in the main loop, as is their final effect on basic-cells. All updates to the basic-cells are collected first, only to be effectuated once per iteration round (i.e. synchronous update).

## 7. RECENT RESEARCH

Since this paper was written we elaborated on several aspects of the model. It has been extended to a process model (Desain, Honing, & de Rijk, 1989), a rigorous mathematical description is given in Desain & Honing (1991), and a detailed comparison with the Longuet-Higgins model and its interpretation as a cognitive model is described in Desain (1991a).

## 8. ACKNOWLEDGEMENTS

We would like to thank Eric Clarke, Jim Grant, and Dirk-Jan Povel, for their help in this research, and their comments on the first version of this paper. This research was partly supported by an ESRC grant under number A413254004.

## 9. REFERENCES

Bharucha, J.J. 1987. Music Cognition and Perceptual Facilitation, A Connectionist Framework. *Music Perception* 5.

- Chowning, J., L.Rush, B. Mont-Reynaud, C. Chafe, W. Andrew Schloss, & J. Smith, 1984. Intelligent systems for the Analysis of Digitized Acoustical Signals. CCRMA Report No. STAN-M-15.
- Chung, J.T. 1989. An Agency for the Perception of Musical Beats or If I Only Had a Foot. Masters Thesis, Department of Computer Science, MIT, Boston.
- Clarke, E., 1987. Levels of Structure in the Organization of Musical Time. *Contemporary Music Review* 2:212-238.
- Dannenber, R. B. & B. Mont-Reynaud, 1987. An on-line Algorithm for Real Time Accompaniment. In Proceedings of the 1987 International Computer Music Conference. San Francisco: Computer Music Association.
- Desain, P. & H. Honing, 1989. Quantization of Musical Time: A Connectionist Approach. *Computer Music Journal* 13(3), also in Todd & Loy (1991).
- Desain, P., H. Honing, & K. de Rijk. 1989 A Connectionist Quantizer. In Proceedings of the 1989 International Computer Music Conference. San Francisco: Computer Music Association.
- Desain, P. 1991a. A Connectionist and a Traditional AI Quantizer, Symbolic versus Sub-symbolic Models of Rhythm Perception. In Proceedings of the 1990 Music and the Cognitive Sciences Conference, edited by I. Cross. *Contemporary Music Review*. London: Harwood Press. (French version, edited by I. Deliège at Brussels: Mardage Editions). (forthcoming).
- Desain, P. 1991b Parsing the Parser, a Case Study in Programming Style. Internal Report. Utrecht: Center for Knowledge Technology. (Submitted to Computer Music Research).
- Longuet-Higgins, H.C., 1987. *Mental Processes*. Cambridge, Mass.: MIT Press.
- Minsky, M. 1986. *Society of Mind*. New York: Simon and Schuster.
- Rasch, R. A. 1979. Synchronization in Performed Ensemble Music *Acustica* 43(2):121-131.
- Todd, P. & Gareth Loy. D. eds. 1991. *Music and Connectionism*, Cambridge, Mass.: MIT Press. Forthcoming.
- Rumelhart, D.E. & McClelland. J.E. eds. 1986. *Parallel Distributed Processing*. Cambridge, Mass.: MIT Press.
- Shaffer, L.H. 1981. Performances of Chopin, Bach, Bartok: Studies in Motor Programming. *Cognitive Psychology* 13:326-376.
- Todd, N.P. 1985. A Model of Expressive Timing in Tonal Music. *Music Perception* 3(1):33-58.
- Vorberg, D. J. & R. Hambuch, 1987. On the Temporal Control of Rhythmic Performance. In: J. Requin (Ed.) *Attention and Performance VII*.
- Vos. P. & Handel, S. 1987. Playing Triplets: Facts and Preferences. In: A Gabrielsson (Ed.) *Action and Perception in Rhythm and Music*. Royal Swedish Academy of Music. No. 55:35-47.



## APPENDIX I, The traditional algorithm.

```
;;; MICRO TRADITIONAL QUANTIZER
;;; (C)1990, Desain & Honing
;;; in Common Lisp (uses loop macro)

;;; utilities

(defun square (x) (* x x))

(defun quantize (intervals &key (speed 0.0) (trust 1.0)
                (quantum (first intervals)))
  "Quantize time intervals in multiples of quantum"
  ;; speed = 0, trust = 1 :inter-onset quantizer
  ;; 0<speed<1, trust = 1 :tempo tracker
  ;; 0<speed<1, 0<trust<1 :tempo tracker with confidence
  (loop for in in intervals
        as out = (quantize-ioi in quantum)
        as error = (quantization-error in out quantum)
        do (incf quantum
              (* (delta-quantum error out quantum)
                 (confidence error trust)
                 speed))
        collect out))

(defun quantize-ioi (time quantum)
  "Return approximation of time in multiples of quantum"
  (round (/ time quantum)))

(defun quantization-error (in out quantum)
  "Return error of quantization"
  (- (/ in quantum) out))

(defun delta-quantum (error out quantum)
  "Return the quantum change that would have given a zero error"
  (* quantum (/ error out)))

(defun confidence (error trust)
  "Return amount of confidence in a possible tempo adjustment"
  (- 1 (* (- 1 trust) (square (* 2 error))))))

;;; example: real performance data: no luck
(quantize '(1.177 0.592 0.288 0.337 0.436 0.337 0.387 0.600
           0.634 0.296 0.280 0.296 0.346 1.193)
          :quantum 0.1 :speed 0.5)
-> (12 6 3 3 4 3 4 6 6 3 3 3 4 13)
```

## APPENDIX II, The Longuet-Higgins algorithm.

```
;;; LONGUET-HIGGINS QUANTIZER
;;; (C)1990, Desain
;;; Stripped version: no articulation analysis, metrical structure or tempo tracking
;;; in Common Lisp (uses loop macro)

;;; utilities

(defun make-onsets (intervals)
  "Translate inter-onset intervals to onset times"
  (loop for interval in intervals
        sum interval into onset
        collect onset into onsets
        finally (return (cons 0.0 onsets))))

(defun make-intervals (onsets)
  "Translate onset times to inter-onset intervals"
  (loop for onset1 in onsets
        for onset2 in (rest onsets)
        collect (- onset2 onset1)))

(defun alternative (metre &rest states)
  "Return alternative metre plus unaltered states"
  (cons (case (first metre) (2 '(3)) (3 '(2)))
        states))

(defun extend (metre)
  "Return alternative metre plus unaltered states"
  (or metre '(2)))

;;; main parsing routines

(defun quantize (intervals &key (metre '(2)) (tol 0.10)
                (beat (first intervals)))
  "Quantize intervals using initial metre and beat estimate"
  (loop with start = 0.0
        with onsets = (make-onsets intervals)
        for time from 0
        while onsets
        do (multiple-value-setq (start figure metre onsets)
            (rhythm start beat metre onsets time 1 tol))
        append figure into figures
        finally (return (make-intervals figures))))
```

```

(defun rhythm (start period metre onsets time factor tol)
  "Handle singlet and subdivide as continuation"
  (singlet
   start (+ start period) metre onsets time tol
   #'(lambda (figure onsets)
       (tempo figure start period metre onsets time factor tol))))

(defun singlet (start stop metre onsets time tol cont)
  "Handle singlet note or rest"
  (if (and onsets (< (first onsets) (+ start tol)))
      (singlet-figure stop metre (list time) (rest onsets) tol cont)
      (singlet-figure stop metre nil onsets tol cont)))

(defun singlet-figure (stop metre figure onsets tol cont)
  "Create singlet figure and subdivide in case of more notes"
  (let* ((onset (first onsets))
         (syncope (or (null onset) (>= onset (+ stop tol))))
         (more? (and onset (< onset (+ stop (- tol))))))
         (if more?
             (apply #'values (funcall cont figure onsets))
             (values (if syncope stop (first onsets))
                     figure metre onsets syncope))))

(defun tempo (figure start period metre onsets time factor tol)
  "One or two trials of subdivision using alternative metres"
  (rest (generate-and-test #'trial
                          #'(lambda (syncope stop &rest ignore)
                              (and (not syncope)
                                   (< (- stop tol)
                                       (+ start period)
                                       (+ stop tol))))
                          #'alternative
                          metre figure start period onsets time factor tol)))

(defun generate-and-test (generate test alternative &rest states)
  "Control structure for metre change"
  (let ((result1 (apply generate states)))
    (if (apply test result1)
        result1
        (let ((result2 (apply generate (apply alternative states))))
          (if (apply test result2)
              result2
              result1))))))

```



```

(defun trial (metre figure start period onsets time factor tol)
  "Try a subdivision of period"
  (loop with pulse = (pop metre)
        with sub-period = (/ period (float pulse))
        with sub-factor = (/ factor pulse)
        repeat pulse
        for sub-time from time by sub-factor
        do (multiple-value-setq
            (start sub-figure metre onsets syncope)
            (rhythm start sub-period (extend metre) onsets
                    sub-time sub-factor tol))
        append sub-figure into sub-figures
        finally
          (return
            (list syncope start (append figure sub-figures) (cons pulse metre)
                    onsets))))

;;; example
(quantize '(1.177 0.592 0.288 0.337 0.436 0.337 0.387 0.600 0.634
            0.296 0.280 0.296 0.346 1.193) :tol 0.15)
->(1 1/2 1/4 1/4 1/3 1/3 1/3 1/2 1/2 1/4 1/4 1/4 1/4 1)

```

### APPENDIX III, The connectionist algorithm.

```
;;; MICRO CONNECTIONIST QUANTIZER
;;; (C)1990, Desain & Honing
;;; in Common Lisp (uses loop macro)

;;; utilities

(define-modify-macro multf (factor) *)
(define-modify-macro divf (factor) /)
(define-modify-macro zerof () (lambda(x) 0))

(defmacro with-adjacent-intervals
  (vector (a-begin a-end a-sum b-begin b-end b-sum) &body body)
  "Setup environment for each interaction of (sum-)intervals"
  `(loop with length = (length ,vector)
    for ,a-begin below (1- length)
    do (loop for ,a-end from ,a-begin below (1- length)
      sum (aref ,vector ,a-end) into ,a-sum
      do (loop with ,b-begin = (1+ ,a-end)
        for ,b-end from ,b-begin below length
        sum (aref ,vector ,b-end) into ,b-sum
        do ,@body))))

;;; interaction function

(defun delta (a b minimum peak decay)
  "Return change for two time intervals"
  (let* ((inverted? (<= a b))
        (ratio (if inverted? (/ b a) (/ a b)))
        (delta-ratio (interaction ratio peak decay))
        (proportion (/ delta-ratio (+ 1 ratio delta-ratio))))
    (* minimum (if inverted? (- proportion) proportion))))

(defun interaction (ratio peak decay)
  "Return change of time interval ratio"
  (* (- (round ratio) ratio)
    (expt (abs (* 2 (- ratio (floor ratio) 0.5))) peak)
    (expt (round ratio) decay)))
```

```

;;; quantization procedures

(defun quantize (intervals &key (iterations 20) (peak 5) (decay -1))
  "Quantize data of inter-onset intervals"
  (let* ((length (length intervals))
         (changes (make-array length :initial-element 0.0))
         (minimum (loop for index below length
                        minimize (aref intervals index))))
    (loop for count to iterations
          do (update intervals minimum changes peak decay)
          finally (return (coerce intervals 'list))))

(defun update (intervals minimum changes peak decay)
  "Update all intervals synchronously"
  (with-adjacent-intervals intervals
    (a-begin a-end a-sum b-begin b-end b-sum)
    (let ((delta (delta a-sum b-sum minimum peak decay)))
      (propagate changes a-begin a-end (/ delta a-sum))
      (propagate changes b-begin b-end (- (/ delta b-sum)))))
  (enforce changes intervals))

(defun propagate (changes begin end change)
  "Derive changes of basic-intervals from sum-interval change"
  (loop for index from begin to end
        do (incf (aref changes index) change)))

(defun enforce (changes intervals)
  "Effectuate changes to intervals"
  (loop for index below (length intervals)
        do (multf (aref intervals index)
                  (1+ (aref changes index)))
          (zerof (aref changes index))))

;;; example (the result is rounded)
(quantize (vector 1.177 0.592 0.288 0.337 0.436 0.337 0.387 0.600
                 0.634 0.296 0.280 0.296 0.346 1.193))
->(1.2 .6 .3 .3 .4 .4 .4 .6 .6 .3 .3 .3 .3 1.2)

```



**Peter Desain and Henkjan Honing**

Centre for Art, Media, and Technology  
Utrecht School of the Arts  
Langestraat 2b  
3511 BK Utrecht  
The Netherlands

Music Department  
City University  
Northampton Square  
London EC1V 0HB  
United Kingdom

**The Quantization of  
Musical Time: A  
Connectionist Approach**

Will be published as: Desain, P. & H. Honing (1991) Quantization of Musical Time: A Connectionist Approach. In *Music and Connectionism*, edited by P.M. Todd and G.J. Loy. Cambridge, Mass.: MIT Press.

---

## Peter Desain and Henkjan Honing

Centre for Art, Media, and Technology  
Utrecht School of the Arts  
Lange Viestraat 2b  
NL-3511 BK Utrecht  
The Netherlands

Music Department  
City University  
Northampton Square  
London EC1V OHB  
United Kingdom

# The Quantization of Musical Time: A Connectionist Approach

## Introduction

Musical time can be considered to be the product of two time scales: the discrete time intervals of a metrical structure and the continuous time scales of tempo changes and expressive timing (Clarke 1987a). In musical notation both kinds are present, although the notation of continuous time is less developed than that of metric time (often just a word like "rubato" or "accelerando" is notated in the score). In the experimental literature, different ways in which a musician can add continuous timing changes to the metrical score have been identified. There are systematic changes in certain rhythmic forms: for example, shortening triplets (Vos and Handel 1987) and timing differences occurring in voice leading with ensemble playing (Rasch 1979). Deliberate departures from metricality, such as rubato, seem to be used to emphasize musical structure, as exemplified in the phrase-final lengthening principle formalized by Todd (1985). In addition to these effects, which are collectively called *expressive timing*, there are nonvoluntary effects, such as random timing errors caused by the limits in the accuracy of the motor system (Shaffer 1981) and errors in mental time-keeping processes (Vorberg and Hambuch 1978). These effects are generally rather small—in the order of 10–100 msec. To make sense of most musical styles, it is necessary to separate the discrete and continuous components of musical time. We will call this process of separation *quantization*, although the term is generally used to reflect only the extraction of a metrical score from a musical performance.

Computer Music Journal, Vol. 13, No. 3, Fall 1989,  
© 1989 Massachusetts Institute of Technology.

## Perception of Musical Time

Human subjects, even without much musical training, can extract, memorize, and reproduce the discrete metrical structure from a performance of a simple piece of music—even when a large continuous timing component is involved. This is surprising, given that the note durations in performance can deviate by up to 50 percent from their metrical values (Povel 1977). Indeed, it seems that the perception of time intervals on a discrete scale is an obligatory, automatic process (Sternberg, Knoll, and Zukofsky 1982; Clarke 1987b). This so-called categorical perception can also be found in speech perception and vision. By contrast, the perception and reproduction of continuous time in musical performance seems to be associated with expert behavior.

Once the discrete and continuous aspects of timing have been separated by a quantization process, each can function as an input to other processes. The induction of an internal clock (Povel and Essens 1985) and the reconstruction of the hierarchical structure of rhythmical patterns (Mont-Reynaud and Goldstein 1985) both rely on the presence of a metrical score, while Todd (1985) has developed a model in which hierarchical structure is recovered from expressive timing alone.

## Applications of Quantization

Apart from its importance for cognitive modeling, a good theory of quantization has technical applications. It is one of the bottlenecks in the automatic transcription of performed music, and is also important for compositions with a real-time, interac-

Fig. 1. Example of a performed score and its quantization by a commercial MIDI Package using a resolution of 1/64 note.



tive component where the computer improvises or interacts with a live performer. Last but not least, a quantization tool would make it possible to study the expressive timing of music for which no score exists, as in improvised music.

## Known Methods

Few computational models are available in the literature for separating a metrical score from expressive timing in performed music (Desain and Honing 1988). Available methods produce a considerable number of errors when quantizing the data. The traditional approach is to expand and contract note durations according to a metrical grid that is more or less fixed—the grid being adjustable to incorporate different, low-level subdivisions (e.g., for triplets). Commercial MIDI software uses this method, which often gives rise to a musically absurd output, as shown in Fig. 1. Better results are obtained when the system tracks the tempo variations of the performer (Dannenberg and Mont-Reynaud 1978), though the system still returns an error rate of 30 percent. More sophisticated artificial intelligence (AI) methods use knowledge about meter (Longuet-Higgins 1987) and other aspects of musical structure. A particularly elaborate system originated at the CCRMA center at Stanford University in the automatic transcription project (Chowning et al. 1984). This knowledge-based method uses information about different kinds of accent, local context, and other musical clues to guide the search for an optimal quantized description of the data. It is entirely implemented in a symbolic, rule-based paradigm. This approach can be seen as the antithesis of our approach, in which all knowledge in the system is represented implicitly. We took the connectionist approach because knowledge-based approaches seemed to offer no real solution to manifest inadequacies of the simplistic metrical grid method. As with the majority of traditional AI programs, the sophisticated knowledge these AI methods use is extremely domain dependent (depending on a specific musical style), causing the systems to break down rapidly when applied to data foreign to this style.

## Connectionist Methods

Connectionism provides the possibility for new kinds of models with characteristics traditional AI models lack, in particular robustness and flexibility (Rumelhart and McClelland 1986). Connectionist models consist of a large number of simple elements, each of which has its own activation level. These cells are interconnected in a complex network, with the connections serving to excite or inhibit other elements. One broad class of these networks, known as *interactive activation and constraint satisfaction networks*, generally converge towards an equilibrium state given some initial state.

An example of the application of these networks to music perception is given by Bharucha (1987) in the context of tonal harmony. These networks have not yet been used for quantization. The quantization model presented in this paper is a connectionist network designed to converge from nonmetrical performance data to a metrical equilibrium state. This convergence is hard wired into the system, and no learning takes place. The model is thought of as a collection of relatively abstract elements, each of which performs a rather complex function compared to standard connectionist models. While it may be possible to express these functions in terms of one of the formalisms for neural networks, this lies beyond the scope of the present article.

## Basic Model

Consider a network with two kinds of cells: the *basic cell*, with an initial state equal to an inter-onset interval, and the *interaction cell*, which is connected in a bidirectional manner to two basic cells. Figure 2a shows the topology of a network for quantizing a rhythm of four beats, having its three



Fig. 2. Topology of a basic network (a) and a compound network (b).

Fig. 2

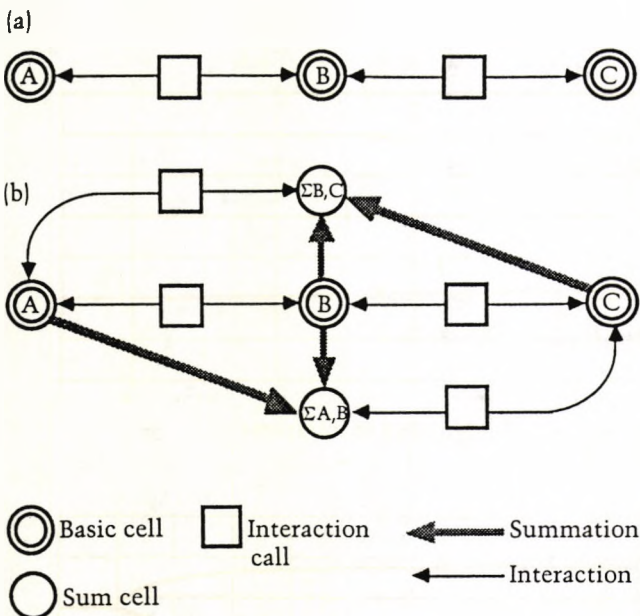


Fig. 3. Interactive time intervals in a basic network (a) and a compound network (b).

Fig. 3

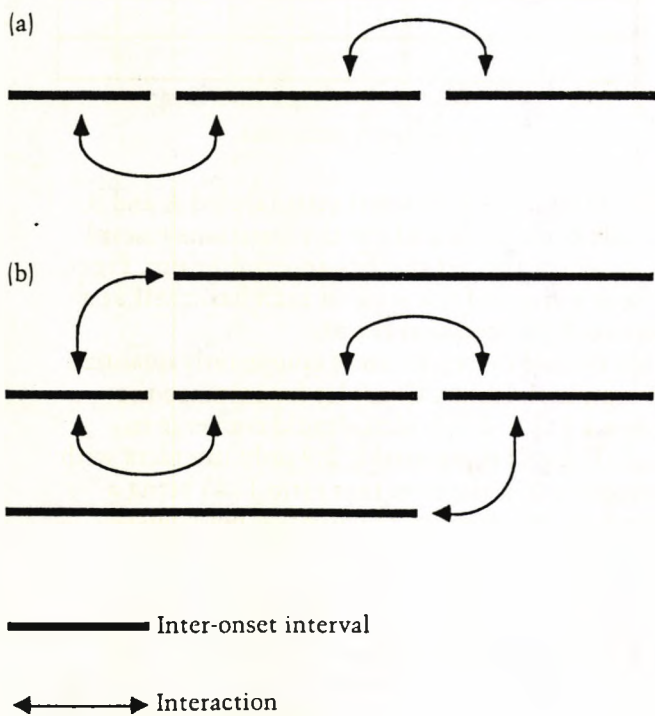
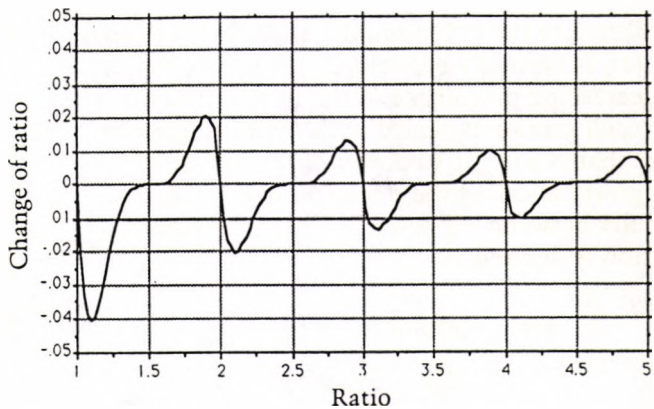


Fig. 4. Interaction function with a peak at 4 and decay equal to -1.

Fig. 4



inter-onset intervals set as states of the three basic cells, labeled A, B, and C. There are two interaction cells connected to the basic cells A and B, and B and C, respectively. Each interaction cell steers the two basic cells to which it is connected toward integer multiples of one another, but only if they are already near this state. It applies the interaction function to the quotient of their states (ratios smaller than 1 are inverted). If this ratio were close to an integer (e.g., 1.9 or 2.1), the interaction function would return a *change of ratio* that would steer the two states toward a perfect integer relation (e.g., 2). Figure 3 illustrates the interactions that are relevant in quantizing the four-beat rhythm. One can see that if the ratio is slightly above an integer, it will be adjusted downward, and vice versa as in Fig. 4.

There are constraints to be taken into account for interaction functions. First, the function and its derivative should be zero in the middle region between two integer ratios. In this region it is not clear if the integer ratio above or below is the proper goal, so no attempt is made to change the ratio. Second, the derivative around integer ratios should be negative to steer the ratio towards the integer, but greater than -1 to prevent overshoot that would result in oscillations. Third, the magnitude of the function should decrease with increasing ratios to diminish the influence of larger ratios. A large class of functions meet these constraints. At present we use a polynomial section around each integer ratio.

Fig. 5. State as a function of iteration count for the rhythm 2, 1, 3 in a basic network (a). State as a function of iteration count for the rhythm 1, 2, 3 in a basic network (b).

The degree of the polynomial, called the *peak* parameter, is typically between 2 and 12. To realize the decreasing magnitude of the interaction function, each section is scaled with a multiplication factor that is a negative power of the integer ratio. This power is called the *decay parameter*, and is typically between  $-1$  and  $-3$ . This interaction function is defined as

$$F(r) = (\text{round}(r) - r) * |2(r - \text{entier}(r) - 0.5)|^p * \text{round}(r)^d,$$

in which the first term gives the ideal change of ratio, the second term signifies the speed of change which is at maximum near an integer ratio (with peak parameter  $p$ ), and the third term scales the change to be lower at higher ratios (with decay parameter  $d$ ). It is simple to prove that this interaction function satisfies the constraints mentioned.

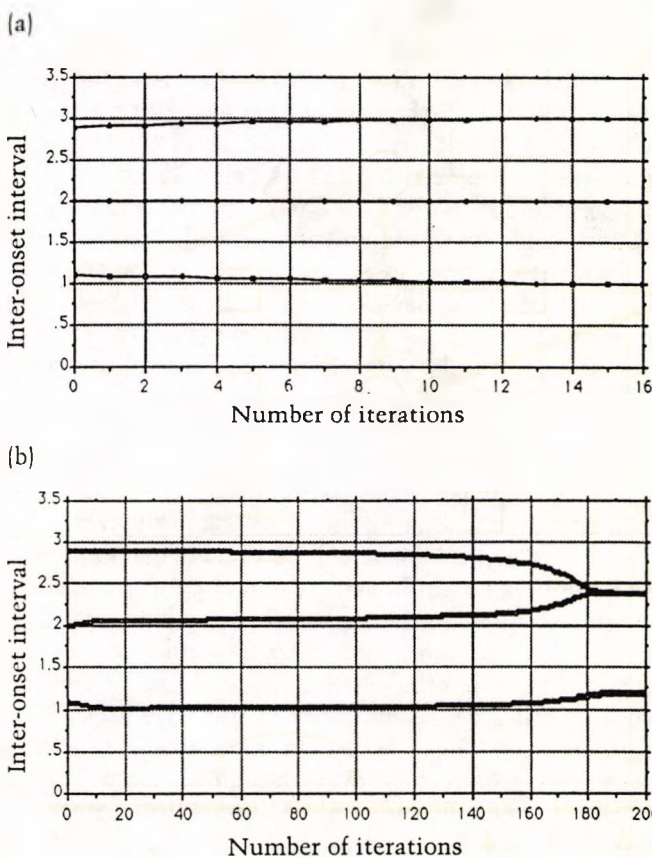
From the change of ratio  $F(a/b)$ , new intervals  $a + \Delta$  and  $b - \Delta$  are calculated without altering the sum of both intervals.

$$\frac{a + \Delta}{b - \Delta} = \frac{a}{b} + F\left(\frac{a}{b}\right)$$

which implies

$$\Delta = \frac{bF\left(\frac{a}{b}\right)}{1 + \frac{a}{b} + F\left(\frac{a}{b}\right)}$$

In simulating the network, each interaction cell updates the states of the two basic cells to which it is connected. This process is repeated, moving the basic cells slowly towards equilibrium. Equilibrium is assumed when no cell changes more than a certain amount between two iterations. For example, let us take a rhythm with inter-onset intervals of 2, 1.1, and 2.9 csec. As the representation of duration is currently unimportant in the model, they are treated as relative values (tempo has no influence on the quantization). This rhythm is represented in a basic network as three cells with the initial states 2.0:1.1:2.9. Iterating the procedure outlined above



for the interactions between cells labeled A and B, and cells B and C will adjust the durations toward 2:1:3, where the net reaches an equilibrium. Figure 5a is a graph of the state of each basic cell as a function of the iteration count.

This type of network can of course only quantize very simple rhythms. Consider for instance the rhythm 1.1:2.0:2.9, which should converge to 1:2:3. The cell representing 2.9 only interacts with its neighbor 2.0, the resultant ratio 1:45 being a long way from an integer. The basic net adjusts these values to 1.2:2.4:2.4, as seen in Fig. 5b.

What the model fails to take account of is the time interval 3.1, the sum of the first two durations. If this interval were incorporated into the model, it would interact successfully with the third interval (2.9) in such a way that the pair of intervals would gravitate toward the ratio 1. This observation leads to a revised model.



Fig. 6. State as a function of iteration count for a complex rhythm in a compound network.

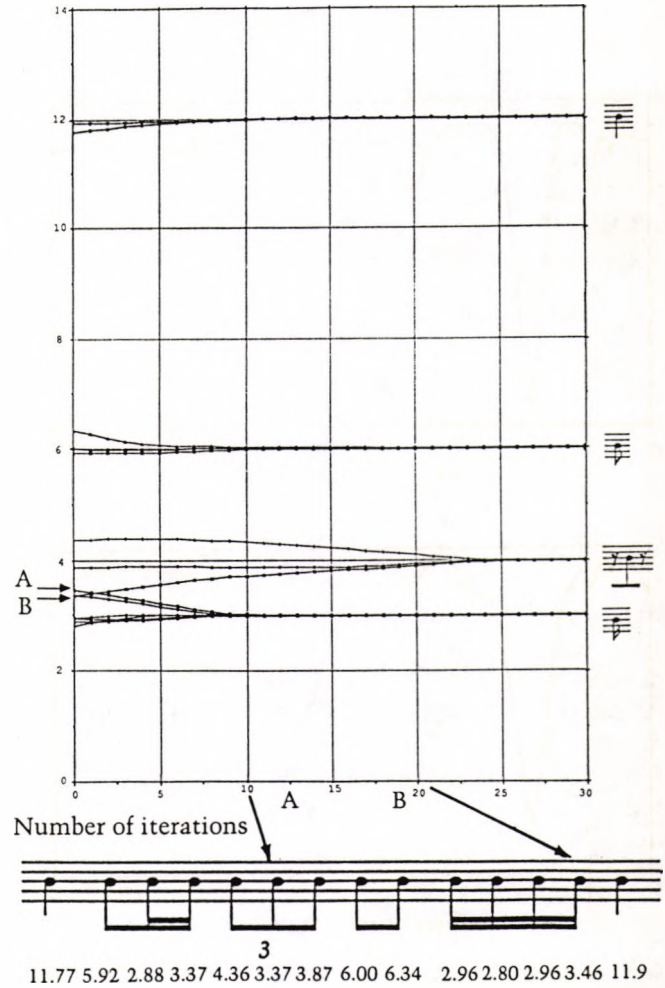
## Compound Model

In order to represent the longer time intervals generated by a sequence of notes, *sum cells* are postulated. These cells sum the activation levels of the basic cells to which they are connected. The interaction of a sum cell with its basic cells is bidirectional; if the sum cell changes its value, the basic cells connected to it will all change proportionally. The sum cells are interconnected to cells representing adjacent intervals by the same interaction cells that are used in the basic model. The function of the interaction cells is once again to try to steer the interconnected cells—which may be sum cells, or a mixture of sum cells and basic cells—toward an integer ratio as was shown in Figs. 2b and 3b.

Our earlier example—a duration sequence of 1.1, 2.0, 2.9—is now quantized correctly due to combined effects of interacting sum cells and the interactions between the basic cells. Let us consider a more complex example using the real performance data shown in Fig. 6. In this rhythm the final sixteenth note is played longer than the middle note of the triplet. Nonetheless the local context of the two intervals steers each note towards its correct value as seen in Fig. 6. The compound model produces promising results, even though the network is rather sparse, allowing only adjacent time intervals to interact. A compound network for a rhythm of  $n$  intervals consists of  $n$  basic cells,  $\lfloor (n + 1) / 2 \rfloor$  sum cells, and  $\lfloor n(n^2 - 1) / 6 \rfloor$  interaction cells.

## Understanding the Model

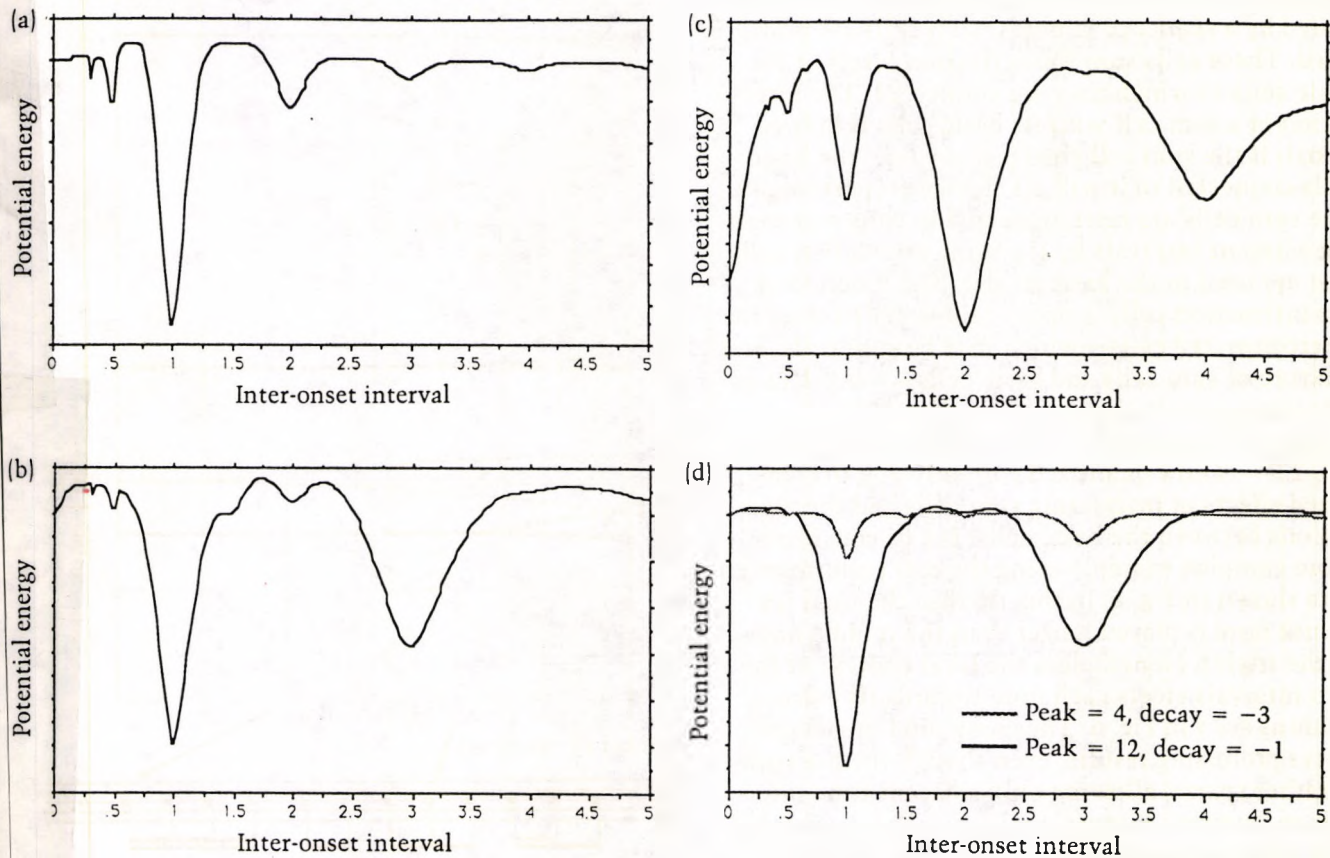
In connectionist systems the global behavior emerges from a large number of local interactions. This makes it very difficult to study the behavior of the network at a detailed level. While it may initially seem attractive to use descriptions like "winning cells," "pulling harder," etc., a better understanding of the patterns of change within the network and of the influence of context requires the development of specialized methods. An approach that has proved very useful is what we call the *clamping method*. This entails the clamping, or



fixing, of the states of all but one of the cells. The remaining cell is given an activation level in a reasonable range (the independent variable). Then the resulting change that would have taken place—after one iteration—if the cell were free to change its activation level is monitored (the dependent variable). In order to facilitate the interpretation of this measure (the amount of change), the function is negated and integrated to give a curve with local minima at stable points. The state of the experimentally varied cell will tend to move towards a minimum, like a rolling ball on an uneven surface. As such, it can be interpreted as a curve of potential energy. These minima and maxima can now be evaluated and judged in light of the context set up



Fig. 7. Clamping curve for a cell with a left context of 1 (a). Clamping curve for a cell with a left context of 2, 1 (b). Clamping curve for a cell with a left context of 2, 1, 1 (c). Clamping curve for a cell with a left context of 2, 1 with different parameters for the interaction function (d).



by the surrounding clamped cells. We call the interval between two neighboring local maxima the *catch range*. A value occurring within this range will move towards the minimum between these two maxima, provided the context does not change. The size of the interval where the potential energy stays close to a minimum is called its *flatness value*. It is a measure of the lack of clarity in the context; simple and clear contexts give rise to sharp minima.

Figure 7a shows the potential energy curve of two cells in a basic network; the first has a state of 1, while the other varies between 0–5. The figure shows prominent local minima at 1, 2, 3, 4 and so on, and at the inverse ratios (.5, .33, and so on). These will be the equilibrium states of the second cell. Note the flatter minima at larger ratios.

A graph of the basic interaction (without sum cells) in a 3 cell net with the first two cells clamped to the values 2 and 1 would yield the same curve,

since the first cell does not interact with the varying third cell. Introducing sum cells, however, gives a different curve as can be seen in Fig. 7b. A minimum is shown at 3 caused by the interaction of the sum of the first and second basic cells with the last cell (3:3 yielding a ratio of 1). The minimum at 3 being strengthened by the interaction of the first cell with the sum of the second two (2:4, yielding a ratio of 2). This interaction also results in a weaker minimum at 1.5 (3:1.5, a ratio of 2). With a left context of 2:1:1 the minimum at 3 almost disappears as in Fig. 7c. There is now a strong minimum at 2 because the sum cell—which combines the durations of the second and third cell—is also 2. The sum of the first three cells give rise to the minimum at 4. This clamping method thus gives a clear picture of the mechanisms involved in the complex interactions through a simplification of the process that assumes fixed values in most of the cells. The

Fig. 8. Clamping curves of two notes in the context of an idealized complex rhythm (a). Clamping curves of two notes in the context of a performed complex rhythm (b).

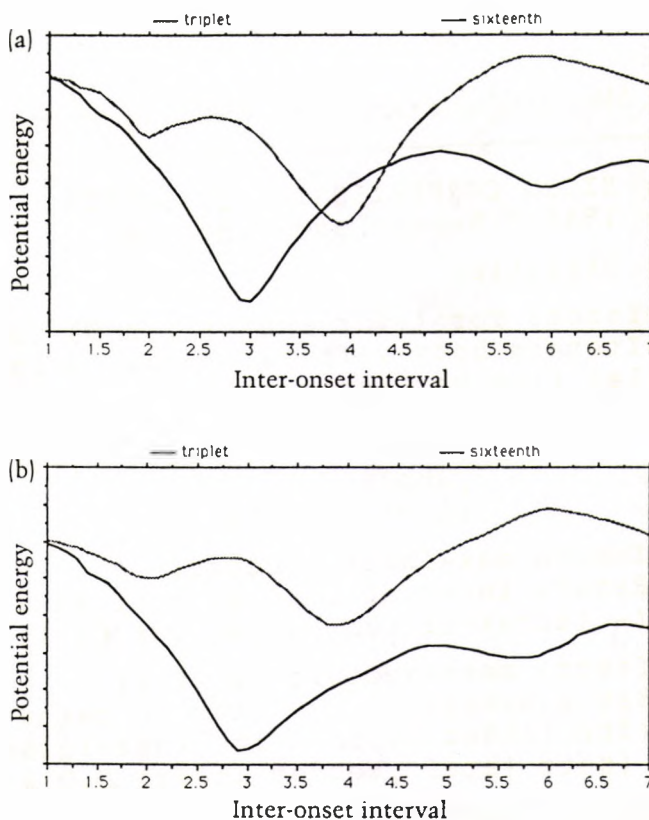
same method can also be used to study the influence of the parameters of the interaction function. In Fig. 7d, which uses the same context as in Fig. 7c, the peak and decay parameters have been changed, showing the effect on the catch range.

If we now return to the more elaborate example shown in Fig. 6, we can study the behavior of the net using the clamping method. Fig. 8a shows the potential energy curves resulting from applying the clamping method to the middle note of the triplet and the final sixteenth note. It shows clearly that the different contexts in which they appear result in different curves and that both will be directed towards the appropriate values. Note the wide catch ranges that allow rather large deviations to be quantized correctly and the smoothness of the curves. This smoothness (the lack of small local minima in the curve) is a result of the large number of interactions (364 and 91 for the triplet and sixteenth notes, respectively), which combine additively to yield each point on the curve. When the clamping experiment is rerun with performance data as context, more complex curves result, with a smaller catch range and a greater flatness, which is shown in Fig. 8b. Nonetheless, the durations still converge towards the correct metrical values.

The position of local maxima in the energy curves constitute the boundaries between the categories into which the data will be quantized. As a result, precise predictions can now be made about the perceptual interpretation of rhythmical sequences with a range of experimentally adjusted durations. It is our intention to compare these predictions with the results of empirical studies.

## Implementation

In simulating a connectionist network, the calculated change in the state of one cell can be effectuated immediately (*asynchronous update*), or can be delayed, effectuating the change of all interactions at once (*synchronous update*). For asynchronous updates, a random order of visiting cells is generally preferred. In Table 1, a simplified implementation of the quantization model is given in Common Lisp (Steele 1984), based on synchronous updates.



The basic cells are represented as a vector of inter-onset intervals. The sum cells are not represented explicitly, but are recalculated, summing the represented interval of basic cells for each interaction. A macro is provided that implements the iteration over adjacent sum intervals. The described interaction function is the one we used for the Figs. 5 and 6. This simplified version requires the minimum inter-onset interval to be around 1. More elaborate versions run in Common Lisp and in C on stock hardware (Macintosh II and Atari ST series machines).

## Further Research

The model we have presented needs high peak values to stabilize accurately. Because this results in smaller catch ranges, we are currently studying the automatic increasing of the peak parameter while

See improved version in Addendum!

Table 1. Micro version of the connectionist quantizer in CommonLISP

```
;;; MICRO CONNECTIONIST QUANTIZER
;;; 1988 P.Desain and H.Honing

;;; Utilities

(defmacro for ((var &key (from 0) to) &body body)
  "Iterate body with var bound to successive values"
  (let ((to-var (gensym)))
    '(let ((,var ,from)(,to-var ,to))
      (loop ,when to '(when (> ,var ,to-var) (return)))
        ,@body
        (incf ,var))))))

(defmacro max-index (vector)
  "Return index of last element in a vector"
  '(- (array-dimension ,vector 0) 1))

(defmacro zero-vector! (vector)
  "Set elements of a vector to zero"
  '(for (index :from 0 :to (max-index ,vector))
    (setf (aref ,vector index) 0.0)))

(defmacro incf-vector-scalar! (a b from to)
  "Increment elements in a range of a vector"
  '(for (index :from ,from :to ,to)
    (incf (aref ,a index) ,b)))

(defmacro incf-relative-vector-vector! (a b)
  "Increment elements of a vector proportionally"
  '(for (index :from 0 :to (max-index ,a))
    (incf (aref ,a index) (* (aref ,a index) (aref ,b index))))))

(defun print-vector (times vector &optional (stream t))
  "Print all elements of vector"
  (format stream "~%-3d: " times)
  (for (index :from 0 :to (max-index vector))
    (format stream "~2,1,5$ " (float (aref vector index)))))

;;; control structure for iteration over intervals

(defmacro with-all-intervals (vector (begin end sum) (start
finish) &body body)
  "Iterating over all intervals contained in [start,finish]"
  '(let (,sum)
    (for (,begin :from ,start :to ,finish)
      (setf ,sum 0.0)
      (for (,end :from ,begin :to ,finish)
        (incf ,sum (aref ,vector ,end))
        ,@body))))))
```

(cont'd)



```

(defmacro with-intervals (vector (begin end sum) (start
finish) &body body)
  "Iterating over intervals"
  '(let ((,sum 0.0)(,begin ,start))
    (for (,end :from ,start :to ,finish)
      (incf ,sum (aref ,vector ,end))
      ,@body)))

(defmacro with-adjacent-intervals
  (vector (a-begin a-end b-begin b-end a-sum
b-sum) &body body)
  "Iterating over interval pairs"
  '(let ((max-index (max-index ,vector)))
    (with-all-intervals ,vector (,a-begin ,a-end
,a-sum) (0 (1- max-index))
      (with-intervals ,vector (,b-begin ,b-end ,b-sum)
        ((1+ ,a-end) max-index)
        ,@body))))

;;; Main quantization procedures

(defun quantize! (durations &optional (peak 4)(decay -1))
  "Quantize data in durations vector"
  (let ((changes (make-array (length durations) :initial-
element 0.0)))
    (for (times :from 0)
      (print-vector times durations)
      (update! durations changes peak decay))))

(defun update! (durations changes peak decay)
  "Update all durations synchronously"
  (zero-vector! changes)
  (with-adjacent-intervals durations
    (a-begin a-end b-begin b-end a-sum b-sum)
    (let ((delta (if (> a-sum b-sum)
                     (delta (/ a-sum b-sum) peak decay)
                     (- (delta (/ b-sum a-sum) peak decay)))))
      (incf-vector-scalar! changes (/ delta a-sum) a-begin
a-end)
      (incf-vector-scalar! changes (- (/ delta b-sum) b-begin
b-end)))
    (incf-relative-vector-vector! durations changes))

(defun delta (ratio peak decay)
  "Return change of time interval"
  (let ((delta-ratio (interaction ratio peak decay)))
    (/ delta-ratio (+ 1 ratio delta-ratio))))

```

(cont'd)

```

(defun interaction (ratio peak decay)
  "Return change of ratio"
  (let ((position (1- (* 2 (- ratio (floor ratio))))))
    (goal (round ratio)))
    (* (- goal ratio)
      (abs (expt position peak))
      (expt goal decay))))

;;; usage examples
;;; minimum element in data should be larger than 1

;(quantize! (vector 1.1 2.0 2.9))
;(quantize! (vector 11.77 5.92 2.88 3.37 4.36 3.37 3.87
                  6.00 6.34 2.96 2.80 2.96 3.46 11.93))

```

the network comes to rest. The dependency of the model on absolute time and absolute tempi is still an open question. The most difficult rhythmic cases for this model are: (1) those that involve additive durations that emerge when rests and tied notes occur in the data and (2) divisive rhythms, such as when a quintuplet is adjacent to a triplet. Our aim is to be able to characterize exactly the limits of the model and to evaluate the computational requirements and the psychological plausibility of the results. A further aim is to develop a robust technical tool for real-time quantization using a process model. Tempo tracking is then an absolute necessity.

## Conclusion

We consider the compound model presented here to be promising. In difficult cases the system undergoes a graceful degradation instead of a sudden breakdown: that is, the range in which rhythms are caught and quantized correctly becomes more and more limited. However, it is a paradoxical problem with connectionist models that their adaptability means that even a rough first implementation, with obvious bugs, may exhibit appropriate behavior. In order to increase an understanding of the process involved, it is necessary to develop specialized tools for diagnosis and investigation. The clamping

method described here seems to have considerable potential, and we are confident that further tools of a similar sort will develop as connectionist modeling gathers momentum.

## Acknowledgments

We would like to thank Dirk-Jan Povel, Steve McAdams, Marco Stroppa, the reviewers of *Computer Music Journal*, and especially Eric Clarke and Klaus de Rijk for their help in this research and their comments on the first version of this paper.

## References

- Bharucha, J. J. 1987. "Music Cognition and Perceptual Facilitation: A Connectionist Framework." *Music Perception* 5(1):1-30.
- Chowning, J., et al. 1984. "Intelligent Systems for the Analysis of Digitized Acoustical Signals." *CCRMA Report* STAN-M-15.
- Clarke, E. 1987a. "Levels of Structure in the Organization of Musical Time." *Contemporary Music Review* 2:212-238.
- Clarke, E. 1987b. "Categorical Rhythm Perception: An Ecological Perspective." In A. Gabrielsson, ed. *Action and Perception in Rhythm and Music*. Stockholm: Royal Swedish Academy of Music. No. 55:19-33.
- Dannenberg, R. B., and B. Mont-Reynaud. 1987. "An On-

- line Algorithm for Real Time Accompaniment." *Proceedings of the 1987 International Computer Music Conference*. San Francisco, California: Computer Music Association, pp. 241-248.
- Desain, P., and H. Honing. 1988. "The Quantization Problem: Traditional and Connectionist Approaches." *Proceedings of the first Artificial Intelligence and Music Workshop*. St. Augustin, West Germany: Gesellschaft für Mathematik und Datenverarbeitung.
- Longuet-Higgins, H. C. 1987. *Mental Processes*. Cambridge, Massachusetts: MIT Press.
- Mont-Reynaud, B., and M. Goldstein. 1985. "On Finding Rhythmic Patterns in Musical Lines." *Proceedings of the 1985 International Computer Music Conference*. San Francisco, California: Computer Music Association, pp. 391-397.
- Povel, D. J. 1977. "Temporal Structure of Performed Music: Some Preliminary Observations." *Acta Psychologica* 41:309-320.
- Povel, D. J., and P. Essens. 1985. "Perception of Temporal Patterns." *Music Perception* 2(4):411-440.
- Rasch, R. A. 1979. "Synchronization in Performed Ensemble Music." *Acustica* 43(2):121-131.
- Rumelhart, D., and J. McClelland, eds. 1986. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol. 1. Cambridge, Massachusetts: MIT Press.
- Shaffer, L. H. 1981. "Performances of Chopin, Bach, Bartok: Studies in Motor Programming." *Cognitive Psychology* 13:326-376.
- Steele, G. L. 1984. *Common Lisp: The Language*. Bedford, Massachusetts: Digital Press.
- Sternberg, S., R. L. Knoll, and P. Zukofsky. 1982. "Timing by Skilled Musicians." In D. Deutsch, ed. *The Psychology of Music*. New York: Academic Press.
- Todd, N. P. 1985. "A Model of Expressive Timing in Tonal Music." *Music Perception* 3(1):33-58.
- Vorberg, D. J., and R. Hambuch. 1987. "On the Temporal Control of Rhythmic Performance." In J. Requin, ed. *Attention and Performance VII*.
- Vos, P., and S. Handel. 1987. "Playing Triplets: Facts and Preferences." In A. Gabrielsson, ed. *Action and Perception in Rhythm and Music*. Royal Swedish Academy of Music. No. 55:35-47.



## Corrections to the original article:

- (1) The name of the first institution mentioned in the article should be changed from *Centre for Art, Media and Technology* into *Center for Knowledge Technology*
- (2) Caption of figure 6 should be changed:
  - a) The arrows should come from the A and B.
  - b) More space is needed between A and B, and the x-axis of the figure.
  - c) The number 3 underneath the triplet was left out. See correction of figure 6 (sent earlier).
- (3) In the text -because of an improvement to the Lisp code (see 4)- the line "*This simplified version requires the minimum inter-onset interval to be around 1*" under the paragraph named Implementation should be removed.
- (4) Lisp code in Table 1 should be replaced with the new code using the loop macro. The enclosed code is considerably improved. See the notes preceeding the Table for remarks on lay-out.
- (5) The reference Desain & Honing (1988) should be replaced by:

Desain, P. & H. Honing (1991) The Quantization Problem: Traditional and Connectionist Approaches. In M. Balaban, K. Ebcioglu & O. Laske, eds. *Musical Intelligence*. Menlo Park: The AAAI Press.

since the AIM Proceedings were never published.

## Addendum

Peter Desain, Henkjan Honing, and Klaus de Rijk

### More Tools for Inspecting the Network

As mentioned previously, the design of special tools and methods to study the network is of great importance, allowing us to explain and predict behavior for particular data, to examine the influence of the parameters on network performance, etc. The clamping method described earlier is one of these tools. A second method visualizes the state space of the system by only taking rhythms of three inter-onset intervals into account. The 3 degrees of freedom are mapped to 2 dimensions by normalizing the total length of the rhythm. Each point  $(x,y)$  represents a rhythm of three inter-onset intervals  $x : y : 1 - x - y$  in a net of interacting cells. Drawing the rhythm after each iteration yields a trajectory towards a stable point in this space: the quantized version of the three intervals.

Plotting the trajectories of different rhythms exhibits the behavior of the network and the stable attractor points in this two dimensional space. They are positioned on straight lines that represent rhythms with an integer ratio of two durations or their sums ( $x=y$ ,  $x+y=z$ ,  $2x=y$ , etc.). Figure 9 shows this state space diagram with a variety of trajectories traced on it. One can see relatively large areas of attraction around the simple rhythms and relatively small areas around more complex rhythms. These so called basins of attraction depend on the parameters of the interaction function; when the peak parameter is set to a higher value (see Figure 9b), more basins of attraction around complex rhythms appear.

Diagrams, as shown in Figure 9, can form the basis for experiments to test the validity of the connectionist quantizing method as a cognitive model for rhythm perception. For example, we can plot the analogous diagram for human listeners performing a categorical perception experiment on part of the rhythm space, and compare it with the output of the quantizer method. The results can be used to adjust the interval-interaction function of the model to more closely match human performance.

++++  
++++  
++++

*Insert Figure 9. around here*

++++

A third method amounts to a systematic exploration of the space of all possible parameter settings. A mapping can be made from this space to the number of correct quantizations of a set of performances. Figure 10 shows this mapping for a set of about 50 relatively simple rhythms, varying in length from 3 to 14 inter-onset intervals, performed by a musical expert. In this way, we defined implicitly what a 'correct' quantization is. The vertical axis shows the percentage of correct quantizations of the system, the other axes show the parameters peak and decay. This visualization brings out specific characteristics of the model. First, it shows the models sensitivity for its parameters. Often connectionist models behave badly in this respect, they need specific parameter settings for different problems. But Figure 10 shows the system behaves quite well with respect to parameter sensitivity. The surface between a peak value of 4 and 6 and a decay value between 0 and -2 is almost flat. Secondly, it shows that the two parameters peak and decay are more or less independent. A decay value between 0 and -1 is most successful, fairly independent of the peak parameter.

Furthermore, families of rhythms with particular characteristics could be made (e.g., rhythms that change meter, syncoped rhythms, rhythms with swing, sloppy performed rhythms) and tested, yielding insights in the limitations of the model for these specific type of rhythms and the musical and cognitive interpretation to the parameters. We did not do any work in this direction yet.

+++++  
 ++++ *Insert Figure 10. around here* +++++  
 +++++

However, the best understanding of such a complex system arises from a mathematical description through which one can search for analytical solutions, prove convergence and stability properties, etc. The present state of the work done on a mathematical description is given below, but much remains to be done.

**Mathematical Model**

Suppose a rhythm is given by a vector  $x$  of durations  $x_i$  with  $1 \leq i \leq N$ . Each update a new duration vector is computed by

$$x^* = x + D(x)$$



Where D in this case is a kind of 'update' function. With a certain initial vector  $x$ , we can construct a set of vectors,  $x^*$ ,  $x^{**}$ , ... hopefully approaching equilibrium. To characterize D we begin by decomposing it into an update of individual basic-cells

$$x_i^* = x_i + D_i(x)$$

An interaction-cell connected to cells with values  $a$  and  $b$  should accomplish an increment of their ratio given by the interaction function.

$$\frac{a^*}{b^*} = \frac{a}{b} + F\left(\frac{a}{b}\right)$$

We convert this change of ratio to a change of time interval  $\Delta(a,b)$  under the constraint that the sum of the intervals stays the same:

$$a^* + b^* = a + b$$

$$a^* = a + \Delta(a,b)$$

$$b^* = b - \Delta(a,b)$$

This combines in the definition of the change effectuated by an interaction-cell

$$\Delta(a,b) = b \frac{F\left(\frac{a}{b}\right)}{1 + \frac{a}{b} + F\left(\frac{a}{b}\right)}$$

In a basic net, each basic-cell (except the left- and rightmost cell) is connected to two interaction-cells. Their change is computed by summing the change from each interaction.

$$D_i(x) = \Delta(x_i, x_{i+1}) - \Delta(x_{i-1}, x_i)$$

This describes the complete behavior of the basic network. In the compound network, the value of the sum-cells is defined as

$$S_{p,q} = \sum_{j=p}^q x_j \quad 1 \leq p \leq q \leq N$$

Suppose a sum-cell  $S_{p,q}$  is changed by an update function  $D_{p,q}$

$$S^*_{p,q} = S_{p,q} + D_{p,q}(x)$$

A sum-cell  $S_{p,q}$  is interacting with a number of sum-cells  $S_{q+1,}$  on the right and a number of sum-cells  $S_{,p-1}$  on the left, yielding the following definition of  $D_{p,q}$

$$D_{p,q}(x) = \sum_{r=q+1}^N \Delta(S_{p,q}, S_{q+1,r}) - \sum_{r=1}^{p-1} \Delta(S_{r,p-1}, S_{p,q})$$

Here, if  $q=N$  the first term vanishes because there are no right neighbors. Likewise if  $p=1$  the second term vanishes.

The change of the sum-cells is propagated proportionally to all the basic-cells connected to it. In each basic-cell the change from all connected sum-cells is summed.

$$D_i(x) = \sum_{p=1}^i \sum_{q=i}^N D_{p,q}(x) \frac{x_i}{S_{p,q}}$$

Summarizing the above and taking care of leftmost and rightmost intervals, gives

$$D_i(x) = \sum_{p=1}^i \sum_{q=i}^{N-1} \sum_{r=q+1}^N \Delta\left(\sum_{j=p}^q x_j, \sum_{j=q+1}^r x_j\right) \frac{x_i}{\sum_{j=p}^q x_j} - \sum_{p=2}^i \sum_{q=i}^N \sum_{r=1}^{p-1} \Delta\left(\sum_{j=r}^{p-1} x_j, \sum_{j=p}^q x_j\right) \frac{x_i}{\sum_{j=p}^q x_j}$$

This describes the behavior of the compound model.

Until now we assume  $a > b$  in the definition of  $\Delta(a,b)$ . We modify it to take care of this.

$$\Delta(a,b) = h(a,b) \frac{F(g(a,b))}{1 + g(a,b) + F(g(a,b))}$$

where  $h(a,b)$  and  $g(a,b)$  are defined by

$$h(a,b) = \begin{cases} b & \text{if } a > b \\ -a & \text{otherwise} \end{cases}$$

$$g(a,b) = \begin{cases} \frac{a}{b} & \text{if } a > b \\ \frac{b}{a} & \text{otherwise} \end{cases}$$

When we implemented these systems, the results were inaccurate or unstable because the change in large sum-cells tended to swamp the influence of smaller, local interactions. Therefore we scaled the interaction with the inverse of the interval  $b$ . This gave a precedence to local interactions that worked well. Because we still want to refrain for the moment from modelling the dependence of quantization on absolute global tempo, which was introduced implicitly by this change, we normalized this factor with the minimum duration. The factor can be incorporated in the definition of  $h(a,b)$ :

$$h(a,b) = \begin{cases} \min_{1 \leq j \leq N} x_j & \text{if } a > b \\ - \min_{1 \leq j \leq N} x_j & \text{otherwise} \end{cases}$$

We have to characterize the equilibrium state for which

$$D_i(x) = 0$$

In the simplified network, it can be proven that this condition only holds when all  $\Delta(x_j, x_{j+1})$  are zero. This implies that the interaction function  $F$  has to be zero for all ratios, which in turn means that all ratios are integers or integers plus 0.5. When the sum cells are introduced the system is much harder to analyze. All equilibrium points of the simplified system are also equilibrium points of the complete system, but there are many additional equilibrium points as well. In fact it is not clear yet what exactly are the (stable) equilibrium points of the complete system.

#### Process model and tempo tracking

A system that takes all of the data into consideration is, of course, not feasible when the aim is to develop a robust technical tool for near real-time quantization of longer pieces, nor



is such an algorithm plausible as a cognitive model. Luckily, it proved quite simple to design a process version of the quantizer which operates upon a limited window of events. In this system, new inter-onset intervals shift into the window and metrical durations shift out, being quantized on the way through. With such a model, tempo tracking becomes an absolute necessity since slow global tempo changes spanning a time lapse larger than the window cannot be operated upon nor corrected for.

The architecture makes use of two main modules, the quantizer and a tempo curve fitter (see Figure 11). They work in mutual corporation, communicating via a window of inter-onset intervals. In phase 1, the quantizer tries to quantize the data in the window. The result is passed together with the original data to the tempo curve fitter. This process tries to explain the difference between the quantized and original data as a global tempo change instead of random fluctuations, by fitting a third order tempo curve to the quantized and original data. With the resulting tempo model the data window is reinterpreted and any consistent global change in tempo is removed from the original data in phase 2. The resulting sequence is now simpler for the quantizer module to operate upon. In phase 3 it is given a chance to remove the remaining deviations. Finally, in phase 4, a quantized inter-onset interval is shifted out of the window and a new interval is shifted in, after being interpreted according to the expected tempo. Then the whole process is repeated.

As a result a rhythm can be quantized differently depending on the context established by the preceding data. Which of course is the same as we would expect from human listeners. For the implementation of the curve fitter special care was taken to use appropriate numerical methods, as numerical inaccuracies build up because of the feedback architecture used in the method, resulting in oscillations.

++++  
+++++ *Insert Figure 11. around here* +++++  
++++

### Polyphony

The system described so far is unable to deal with inter-onset times that should move towards zero (as in chords or music with multiple voices). Although it may be possible to use other means to 'clean' the data before quantizing it, such as rules for recognizing chord chunks, the general connectionist approach used in the quantizer seems a much better

alternative. This is because the context can be taken into account when deciding if for example something is to be considered a chord with some spread or a regular run of notes or an arpeggio that has its own metrical structure. By introducing note durations, the system can distinguish between sequential and simultaneous inter-onset intervals (i.e. overlapping intervals indicate polyphony). We are currently experimenting with multiple interlocking networks that can handle polyphony. The preliminary results seem to be promising.

### **Main Characteristics of the System**

In summary, the connectionist quantization system has three main characteristics: 1) It is *context sensitive*, with precedence of local context, as we demonstrated with the example in Figure 6 and the results of the clamping method. 2) The system has *no explicit musical knowledge*. There is no pre-conceived knowledge of metrical or rhythmical structure used to quantize the performance data other than the notion of "integer ratios". All information is derived from the data itself. 3) The system exhibits *graceful degradation*. When the quantizer breaks down in a complex situation it is often able to maintain musical integrity and consistency at higher levels. The resulting error will only generate a local deformation of the score. Furthermore, this deformation will always be a simplification of the rhythm, not a very complex fragment as produced by some traditional systems (see Figure 1). On the other hand, when more difficult rhythms are fed into the quantizer they imply a smaller range of deviations that can be accurately captured by the system. Thus, they will be quantized correctly when performed with a higher accuracy or consistency. Such behavior could be another possible link to human cognitive performance.

Table 1. Micro version of the connectionist quantizer in Common Lisp.

```
;;; MICRO CONNECTIONIST QUANTIZER
;;; (C)1990, Desain & Honing
;;; in Common Lisp (uses loop macro)

;;; utilities

(define-modify-macro multf (factor) *)
(define-modify-macro divf (factor) /)
(define-modify-macro zerof () (lambda(x) 0))

(defun print-state (time intervals)
  "Print elements of interval vector"
  (loop initially (format t "~%~2D: " time)
        for index below (length intervals)
        do (format t "~2,1,5$ " (aref intervals index))))

(defmacro with-adjacent-intervals
  (vector (a-begin a-end a-sum b-begin b-end b-sum) &body body)
  "Setup environment for each interaction of (sum-)intervals"
  (loop with length = (length ,vector)
        for ,a-begin below (1- length)
        do (loop for ,a-end from ,a-begin below (1- length)
                sum (aref ,vector ,a-end) into ,a-sum
                do (loop with ,b-begin = (1+ ,a-end)
                    for ,b-end from ,b-begin below length
                    sum (aref ,vector ,b-end) into ,b-sum
                    do ,@body))))

;;; interaction function

(defun delta (a b minimum peak decay)
  "Return change for two time intervals"
  (let* ((inverted? (<= a b))
        (ratio (if inverted? (/ b a) (/ a b)))
        (delta-ratio (interaction ratio peak decay))
        (proportion (/ delta-ratio (+ 1 ratio delta-ratio))))
    (* minimum (if inverted? (- proportion) proportion))))

(defun interaction (ratio peak decay)
  "Return change of time interval ratio"
  (* (- (round ratio) ratio)
     (expt (abs (* 2 (- ratio (floor ratio) 0.5))) peak)
     (expt (round ratio) decay)))
```

(cont'd)



```

;;; quantization procedures

(defun quantize (intervals &key (iterations 20) (peak 5) (decay -1))
  "Quantize data of inter-onset intervals"
  (let* ((length (length intervals))
         (changes (make-array length :initial-element 0.0))
         (minimum (loop for index below length
                        minimize (aref intervals index))))
    (loop for count to iterations
          do (print-state count intervals)
              (update intervals minimum changes peak decay))))

(defun update (intervals minimum changes peak decay)
  "Update all intervals synchronously"
  (with-adjacent-intervals intervals
    (a-begin a-end a-sum b-begin b-end b-sum)
    (let ((delta (delta a-sum b-sum minimum peak decay)))
      (propagate changes a-begin a-end (/ delta a-sum))
      (propagate changes b-begin b-end (- (/ delta b-sum)))))
  (enforce changes intervals))

(defun propagate (changes begin end change)
  "Derive changes of basic-intervals from sum-interval change"
  (loop for index from begin to end
        do (incf (aref changes index) change)))

(defun enforce (changes intervals)
  "Effectuate changes to intervals"
  (loop for index below (length intervals)
        do (multf (aref intervals index)
                  (1+ (aref changes index)))
            (zerof (aref changes index))))

;;; examples

;(quantize (vector 1.1 2.0 2.9))
;(quantize (vector 11.77 5.92 2.88 3.37 4.36 3.37 3.87 6.00 6.34
                  2.96 2.80 2.96 3.46 11.93))

```

(Figure 9 left)

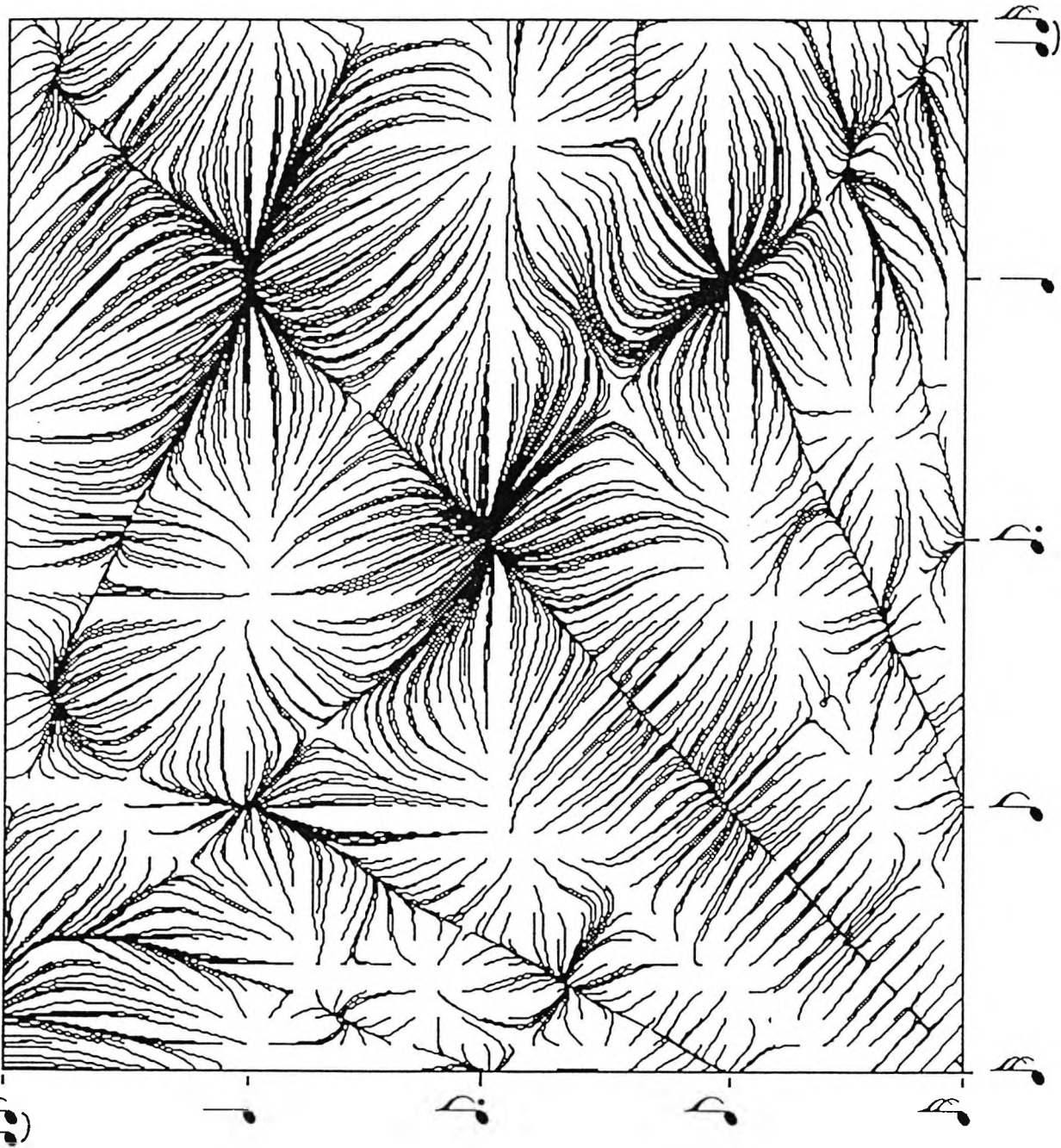
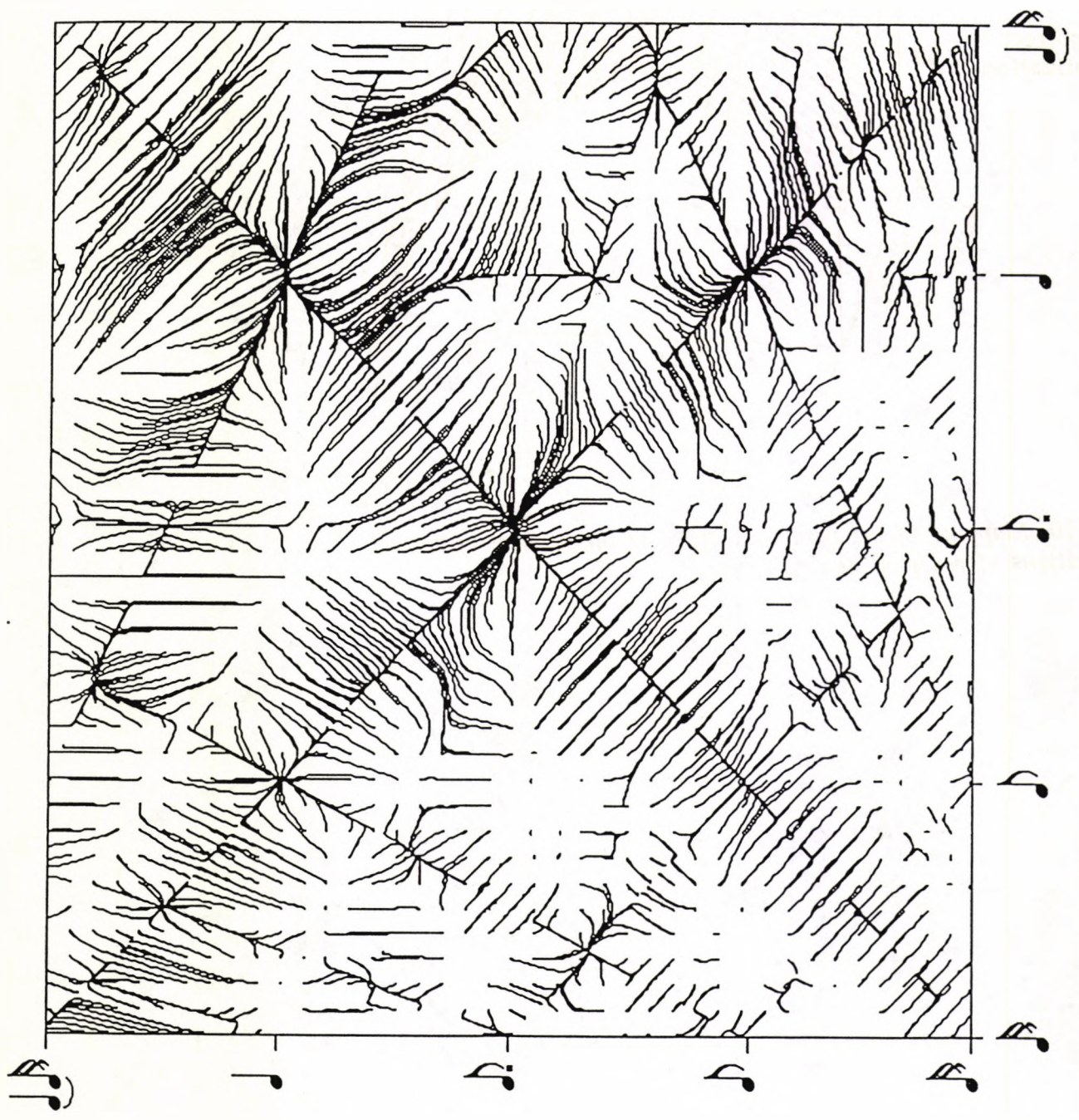


Figure 9. Trajectories in state space of a rhythm of three notes adding up to  $3/4$ . The peak parameter is set to 2 and 6 respectively.



(Figure 9 right)





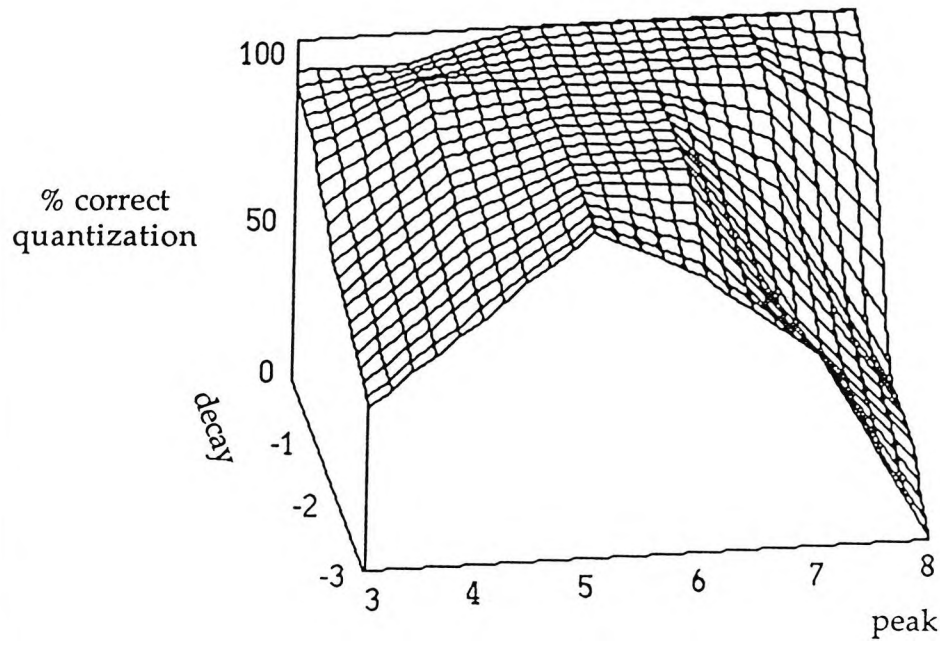


Figure 10. Mapping of the parameter space to the number of correct quantizations of a set of 50 rhythms

5

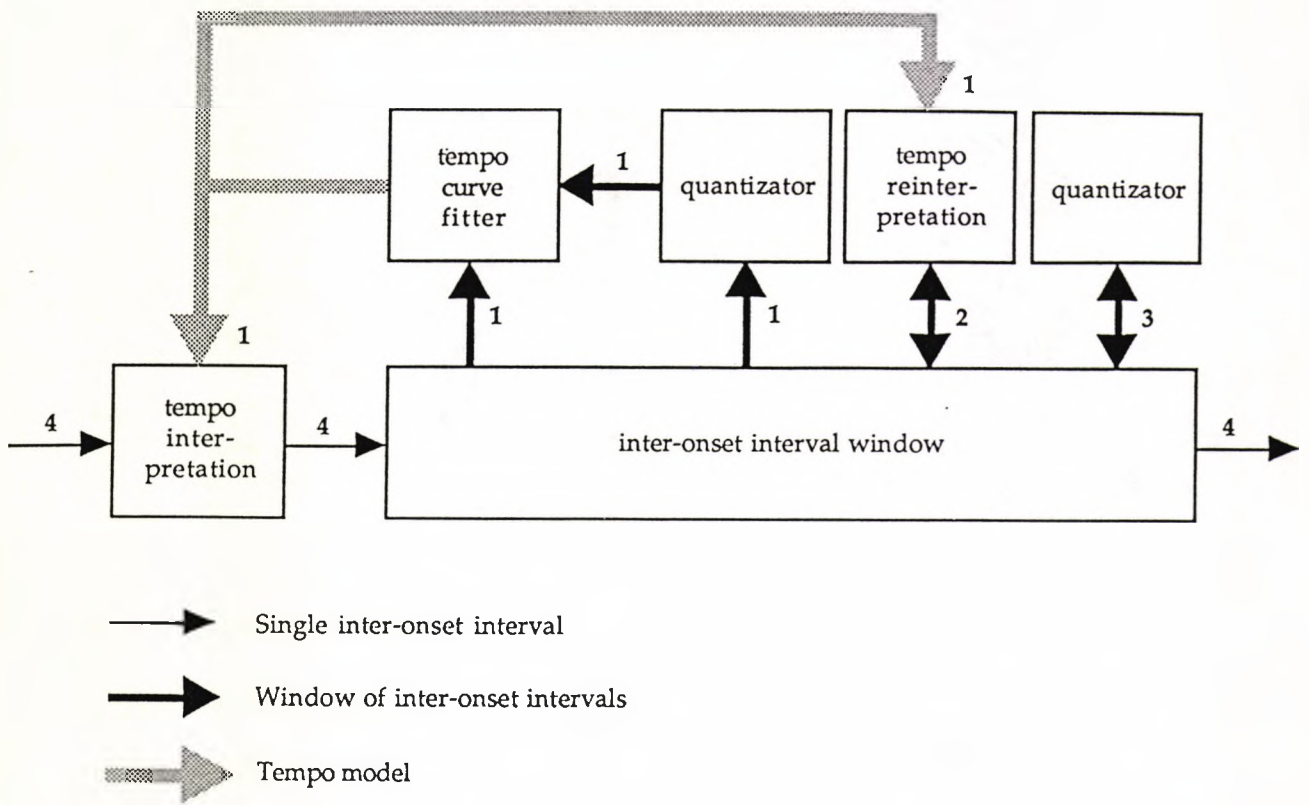


Figure 11. Process model of the connectionist quantizer

# A Connectionist and a Traditional AI Quantizer, Symbolic versus Sub-Symbolic Models of Rhythm Perception

Peter Desain

Center for Knowledge Technology  
Utrecht School of Arts  
Lange Viestraat 2B  
NL-3511 BK Utrecht

Music department  
City University  
Northampton Square  
UK-London EC1V OHB

Will be published as: Desain, P. (forthcoming). A Connectionist and a Traditional AI Quantizer, Symbolic versus Sub-symbolic Models of Rhythm Perception. In Proceedings of the 1990 Music and the Cognitive Sciences Conference, edited by I. Cross. *Contemporary Music Review*. London: Harwood Press.



**Abstract**

The Symbolic AI paradigm and the Connectionist paradigm have produced some incompatible models of the same domain of cognition. Two such models in the field of rhythm perception, namely the Longuet-Higgins Musical Parser and the Desain & Honing connectionist quantizer, were studied in order to find ways to compare and evaluate them. Different perspectives from which to describe their behavior were developed, providing a conceptual as well as a visual representation of the operation of the models. With these tools it proved possible to discuss their similarities and differences and to narrow the gap between sub-symbolic and symbolic models.

**Keywords**

Rhythm Perception, Quantization, (Sub-)Symbolic Processing, Connectionism

## Introduction

The so called Good Old Fashioned Artificial Intelligence has established itself firmly in the past decades as a research methodology. The methods and tools it uses are symbolic, highly structured representations of domain knowledge and transformations of these representations by means of formally stated rules. These rule based theories can function (and are vital) as abstract formal descriptions of aspects of cognition, constraining any cognitive theory. However, some authors go beyond that and claim that mental processes are symbolic operations performed on mental representations of rules (see Fodor, 1975). Until the connectionist paradigm emerged there was no real alternative to that view. But now, in this new paradigm, the departure from reliance on the explicit mental representation of rules is central, and thus the conception of cognition is fundamentally different (Barucha & Olney, 1989). This holds regardless of the fact that the behavior of connectionist models could be formally described in rules. These distributed models consist of a large number of simple elements, or cells, each of which has its own activation level. These cells are interconnected in a network, the connections serving to excite or inhibit others. Connectionism opened the possibility of defining models which have characteristics that are hard to achieve in traditional AI, in particular robustness, flexibility and the possibility of learning (Rumelhart & McClelland, 1986). The connectionist boom brought forth much interesting work, also in the field of music (Todd & Loy, forthcoming). Although many researchers lost their critical attitude, impressed by the good performance of some (prototypical) models, it became soon clear that more study was needed to the limitations of these models. A connectionist model that 'works' well, constitutes in itself no scientific progress, when questions like the sensitivity to parameter changes, the scalability to larger problems and the dependency of the model on a specific input representation, cannot be answered. However, it is possible to describe the behavior of a connectionist model from different abstract perspectives that provide more insight in its limitations and its validity as a cognitive model than simulations or test runs alone. These perspectives are also fruitful for the analysis of traditional AI models. In this article we pursue this approach for a connectionist and a traditional AI model of rhythm perception as a case study for the wider issue how the paradigms themselves relate.

## The Quantization Problem

In performed music there are large deviations from the time intervals as they appear in the score (Clarke, 1987). Quantization is the process by which the time intervals in the score are recovered from the durations in a performed temporal sequence; to put it in another way, it is the process by which performed time intervals are factorized into abstract integer durations representing the notes in the score and local tempo factors. These tempo factors are aggregates of intended timing deviations like rubato and unintended timing deviations like noise of the motor system. This process of separating different discrete and continuous aspects of musical timing, however simple at first sight, and indeed forming a rather basic musical skill, proved to be very hard to model (Desain & Honing, forthcoming). As an example one could try to recover the intended rhythmic interpretation of the following temporal sequence (in milliseconds):

476 : 237 : 115 : 135 : 174 : 135 : 155 : 240 : 254 : 118 : 112 : 118 : 138 : 476

This task, however hard by calculation, yields an obvious and simple answer when the data is converted to an auditory stimulus: a sequence of drumbeats (the solution is given in note 1).

## Known Methods

The simplest method of quantization, used by most commercially available music transcription programs, is the round-off of any point in time to the nearest point on a fixed time grid, with a resolution equal to, or an integral factor smaller than, the smallest duration to be expected. This method is totally inappropriate: even when enhanced with facilities like user control over the grid resolution, it yields results that makes no musical sense, even when the performer is forced to play along with a metronome.

However, this method can serve as the basis of more reasonable models in which the time grid is adapted if consistent deviations (notes being late or early) are detected. In this so called 'tempo-tracking' the design of the control behavior becomes crucial: the extraction of an error signal between time grid and note onsets, and the way in which this error influences the tempo of the

grid. The most elaborate example is the 'real time foot tapper' (Dannenbergh & Mont-Reynaud, 1987 and Boulanger, 1990), but a still a 30% error ratio is reported for this system.

A symbolic, rule based system for quantization was in place at the Stanford automatic transcription project (Chowning, Rush, Mont-Reynaud, Chafe, Schloss & Smith, 1984). It used knowledge about preferable ratios between time intervals as a basis for an optimal quantized description of the data. In such a knowledge based system it is relatively easy to use information from other domains (e.g. dynamic, harmonic) to help the quantization process, but one has to keep in mind that this increases the risk of style dependency and therefore brittleness. Because of its design as an unordered collection of rules it is, like all rule based systems, impossible to characterize its behavior in non-operational terms.

The musical parser (Longuet-Higgins, 1987) comprises another symbolic AI approach to quantization, besides methods of tonal and articulation analysis that will be ignored here. It is highly hierarchical in its music representation and has a reasonable good performance. Furthermore it had the advantage of a published program being available. A Lisp version of this program is published in Desain (1990).

The connectionist quantizer (Desain & Honing, 1989,1991; Desain, Honing & de Rijk, 1989) is a distributed model of fairly simple processing elements. This model displays desirable properties like robustness, graceful degradation and precedence of local context, but as a model it is hard to understand why it works so well, and what its limitations are.

These last two methods will now be described in more detail.

#### **The Longuet-Higgins Musical Parser, a Symbolic Model**

Using just a little knowledge about meter, and exploiting that to the extreme, the Longuet-Higgins Musical Parser builds a metrical tree from performance data, and thus implicitly manages to quantize it. This method is supplied with an initial notion of a time interval called the beat. This interval is subdivided recursively in 2 or 3 parts looking for onset times near the start of each part, until the interval contains no more onsets. The 'best' subdivision is then returned. At each recursive level the interval length is adjusted on the basis of the onsets found, just as in simple tempo-tracking methods.

The output of the system consists of a list of trees, one for every analyzed beat. Each tree is of a combined binary-ternary nature, which means that each node has 0 (in case it is a leaf of the tree) or 2 or 3 sub-trees. During the construction of the tree there is a horizontal flow of information through the layers of the tree, seeking to maintain the same kind of subdivision at a certain level as long as possible. The description of the proposed subdivisions at each level of the tree is called meter. During the construction of the tree a strict left to right order is maintained, and new sub-trees are created on a generate-and-test basis. This means that a proposed (and constructed) binary sub-tree may be rejected in favour of a tertiary one. The generate and test procedure is non-standard in that it may, after checking and rejecting the first alternative, still reject the second in which case as yet the first alternative is chosen.

There is one parameter (called tolerance) identified in the program. It is used in different places as the allowed margin of deviation in deciding if notes start or stop at a certain times. In this way the model does depend elegantly on global tempo by limiting the possibility of further subdivisions when an absolute time span (the tolerance) is reached: onsets that happen within the tolerance interval are considered synchronous.

#### **The Desain & Honing Connectionist Quantizer, a Sub-Symbolic Model**

A class of connectionist models, known as interactive activation and constraint satisfaction networks generally behave so as to converge towards an equilibrium state given some initial state. The connectionist quantization model is designed to converge from non-metrical performance data to a metrical equilibrium. The network topology is fixed (hard-wired) and so is the kind of interaction between cells: no learning takes place.



The net comprises cells for each time interval in a temporal sequence, be it basic (one inter-onset: interval) or compound (spanning several notes). Two cells representing neighboring time intervals may interact and push each other to their 'perfect' values implied by an integer ratio, propagating the changes through the net. After a while the net stabilizes and a quantized temporal sequence can be read out. The interaction between cells, the change of their ratio, depends only on the ratio of their durations, via a so called interaction function. Since the ratio of two time intervals is the only determinant of local behavior, the quantization result does not depend on absolute global tempo, nor can it handle polyphony. The interaction function is a section-wise polynomial with 2 parameters called peak and decay; the first reflects the size of the 'capture' range around an integer ratio, the second represents the decreasing influence of higher ratios. It has to be stressed that all aspects of the global behavior of the system are determined completely by these parameters.

A model that takes a whole temporal sequence into consideration at once is not feasible when the aim is to develop a cognitive model. Luckily, it proved quite simple to design a version of the quantizer which operates upon a window of events. In such a model tempo tracking can handle slow global tempo changes. For reasons of space this part of the connectionist quantizer will not be described here.

### Differences

The models described can be characterized as complete antipoles in a number of aspects. They are summarized roughly in the table in figure 1. The huge differences made comparing them quite hard, but in the end the work was gratifying. Because the systems are prototypical for the two main AI paradigms the results may well generalize to other cases.

\*\*\*\*\*  
\*\*\*\*\* Insert figure 1 around here \*\*\*\*\*  
\*\*\*\*\*

### Different Perspectives

Different perspectives for describing these models will now be given, each at its own level of abstraction. Some perspectives will generalize over sets of inputs or parameters, some will reduce the amount of variability by keeping certain concepts fixed. I hope to show that this search for different representations of the behavior of a computational cognitive model, conceptual as well as visual, is fruitful, even for analyzing traditional symbolic AI programs.

The most direct and raw representation of a computational model is a trace of the computation itself, an overview of how the internal state of the system changes in the course of a complete calculation as a function of the computation-time or the number of computation steps taken. A visualizations of such a trace for the connectionist model is shown in (Desain & Honing, 1989) and for the Longuet-Higgins parser similar graphic representations can be devised.

A deficit of these representations is that they can only be given for one example input at a time, and thus are extremely dependent on the choice of input - in a sense it is easy to 'lie' with these examples by picking one that behaves well. But, on the other hand, these representations show in full detail the ongoing processes and thus enable interpretations and hypothesis forming.

At the other end of the spectrum of possible perspectives is the statistical method, reducing all the information to a number of correct responses. We can assume that, when a skilled performer plays a rhythm, the performed temporal sequence should be quantized as the presented score. Collecting a set of performances and counting the numbers quantized correctly by the model gives us then an indication of its validity. The number of correct quantizations will in general be a function of the possible parameter values given to the model. Visualization of this dependency is useful in the study of the parameter-sensitivity of the models. Often connectionist models behave badly in this respect. They might need specific parameter settings for different problems. Or they might not 'scale-up': for larger problems the model only works for an increasingly smaller range of parameter settings specific to the problem at hand. Parameters might also have no cognitive relevance, and as such could not be used to control the global emerging behavior of the model, or

they might be highly dependent. A visualization of the parameter space can detect such problems. Both models behave well in this respect, but because of space limitations we cannot present the parameter spaces here.

Both perspectives have their drawbacks, one being too specific, the other one too general. If we give up some detail on the speed and order of processing that was available in the computation trace, and give up the free choice of musical material that was available in the parameter space we can characterize the precise behavior of the system for a family of sequences: all possible sequences of a fixed small length. This set can be considered to constitute a rhythm space: the problem space of quantization.

### Rhythm Space Perspective

Let us consider the 3 dimensional space of all possible temporal sequences of 3 inter-onset intervals (four bangs on a drum). Every point in this space represents a unique temporal sequence. One could envisage this space projected in a room, with one corner as the origin. The distance along one wall represents the length of the first time interval, the distance along the second wall represents the length of the second interval, and the height above the floor represents the third. In this space certain points will be perfectly metronomical sequences, other points will represent performed deviations from them. Let us call this space 'rhythm space' although 'temporal sequence space' would be more appropriate. A quantization process maps each point in this space to another; it assigns to each sequence a solution of the quantization. Thus the problem space of a quantization method is the whole rhythm space, the solution space is a set of points within this rhythm space. A further characterization of the solution space (e.g., what constraints limit the set of permissible quantizations - is, for instance, a complex temporal pattern such as 7 : 11 : 2 to be considered allowable?) cannot be given at the moment, which is part of the reason why quantization is a difficult problem to define.

### Trajectories in rhythm space

If the model has intermediate processing states that are temporal sequences themselves, as is the case with the connectionist model, the computation trace becomes a trajectory through this rhythm space. Otherwise a simple straight line can indicate the mapping from problem to solution. Easy visualization requires mapping of this space to 2 dimensions which can be done by assuming that the whole time-length of the temporal sequence is kept constant, the third interval then follows from the first two and the first two durations can be taken as the only independent variables: the x and y axes of a diagram. This normalization, which factors out global tempo, reduces the general applicability of the method if the theory is itself dependent on global tempo. The connectionist quantizer does not (but we admit that to model human rhythm perception accurately, it should). The Longuet-Higgins model does depend on global tempo and for this model the rhythm space can only be shown for one global tempo at a time.

If all three intervals are restricted between a minimum and a maximum time span, the allowed portion of the 2 dimensional projection forms a parallelogram. In figure 2a and 2b the rhythm space for the two models is shown, given an input sequence of three notes between a sixteenth and a double dotted quarter note. The whole sequence has a total duration of three quarter notes. The different solutions are indicated by small circles. Note that in the Longuet-Higgins model some solutions contain inter-onset intervals of length zero. That is because this model interprets two onsets that happen within the tolerance as synchronous.

\*\*\*\*\*  
\*\*\*\*\* Insert figure 2 around here \*\*\*\*\*  
\*\*\*\*\*

### Regions in rhythm space

Because the behavior of the connectionist model is completely determined by a temporal sequence, any point on a trajectory will be mapped to the end point of that trajectory. This means that the connectionist model 'carves up' rhythm space into little compartments around each solution. Each compartment or region contains all the sequences that will be quantized equivalently. Now we can abstract from the trajectory from initial state to solution and only



characterize the compartments. These areas, the so called basins of attraction, can be shown as a partitioning of the rhythm space, as is depicted in figure 3a. The Longuet-Higgins model does not behave such that the solution itself will always lie within the region that will map to that solution. But still the region of all points that map to the same solution (the equivalence classes of the mapping) can be shown as in figure 3b. We can now check the differences between the models. E.g. the region that maps to the rhythm of three quarter notes (marked A in the figures) is much larger in the Longuet-Higgins model than it is in the connectionist model. Another difference is the behavior around the region marked C in figure 3a, which corresponds to a 2:1:2 rhythm. This solution is not present in the Longuet-Higgins model, because it is based on a five-fold division.

\*\*\*\*\*  
\*\*\*\*\* Insert figure 3 around here \*\*\*\*\*  
\*\*\*\*\*

### Influence of context

A good way to understand the influence of context (previously presented musical material) is to consider how these maps change under the influence of it. In figure 4 a context of two dotted quarter notes was presented before the notes shown in the rhythm space. This context heavily biases the behavior of the connectionist model to quantize the following inter-onset intervals in subdivisions thereof as is shown by the enlargement of the area marked B. The area marked C completely disappears in the light of the contextual evidence for these subdivisions. Also in the Longuet-Higgins model the quantization is influenced (or even guided) by context. It does so by propagating the established meter to the processing of the remaining data. Using a duple meter as context, a very similar distortion of the regions in rhythm space can be seen (figure 4 b).

\*\*\*\*\*  
\*\*\*\*\* Insert figure 4 around here \*\*\*\*\*  
\*\*\*\*\*

### Influence of parameters

These maps can also be used to understand the influence of the parameters of the model by interpreting their changes under the influence of different values. In figure 5a the so called peak parameter of the connectionist quantizer is slightly increased. This yields a denser map of smaller regions and adds new regions around 'difficult' rhythms. In the Longuet-Higgins model we can achieve a similar change by decreasing the tolerance. Now the model will behave more 'precisely' and new solutions and small regions around them emerge.

\*\*\*\*\*  
\*\*\*\*\* Insert figure 5 around here \*\*\*\*\*  
\*\*\*\*\*

### Cognitive interpretation

We really need to compare these maps now to the corresponding maps for the human listener to be able to judge the cognitive validity of the models. In principle it is possible to obtain this empirical data in categorical perception experiments, presenting subjects with temporal sequences from the space in a transcription task. But mapping out the whole space will be a paramount task, even for such short sequences. Data about the borderlines between some regions can be found in Schulze (1989) and Clarke (1987).

### Expectancy Space Perspective

The previous representations were based on the abstraction of a whole temporal sequence that served as input of the system. Since the full models work incrementally, a representation that makes explicit how a previously established context influences future decisions would be useful. We have to ignore here any influence of new incoming data back to the previously processed results, which is a reduction for both models. In the full process model of the connectionist quantizer we can 'clamp' the whole of the network state to the partial solution obtained and study what would happen to a new incoming onset. This virtual new onset, acting as a measuring



probe, will be moved by the model to an earlier or a later time. If it is given a positive time shift to a later time, the model clearly had not yet 'expected' an event. If we postulate a measure of expectation of an event, it has to be larger at a later time for this 'early' event. Vice versa: a negative movement, a shift to an earlier time, indicates a dropping expectancy: the event is late. So we can integrate the movement to yield an expectancy measure. It forms a curve with peaks at places where an event, were it happen there, would stay in place. We could also rephrase this explanation in terms of potential and energy. The potential curve projected into the future by the network is then the inverse of the expectancy. But in the context of cognitive models expectancy seems a more appropriate concept. This process of calculating an expectancy can even be done in an incremental way: the expectancy is calculated until a real new event happens, that event is added to the context, and the process starts all over again. In figure 6 this curve is shown for a rhythm in 2/4 and the peaks in between and at the note onsets clearly are positioned at important metrical boundaries. Note that for the sake of clarity the input sequence is already idealized here to a metronomical performance.

\*\*\*\*\*  
 \*\*\*\*\* Insert figure 6 around here \*\*\*\*\*  
 \*\*\*\*\*

To show that these curves capture indeed an abstract property of the input data we can look at the last part of the curve in figure 6 (the last measure between time 16 and 20), and study that for different 2/4 contexts as is done in figure 7. It shows how the different rhythms project a very similar expectancy into the future. This even prompts the challenging thought that these curves constitute a kind of rhythmic 'signature' that can be compared to produce a kind of distance measure, a metric, of rhythms.

\*\*\*\*\*  
 \*\*\*\*\* Insert figure 7 around here \*\*\*\*\*  
 \*\*\*\*\*

One further corroboration of the usefulness of these curves is shown in figure 8. Here the expectancies of two rhythms are compared: one in 6/8 (a division in 2 and then in 3) and the other with time signature 3/4 (a division in 3 and then in 2). The prominent peak in the curve of the first one is located at half the measure length, lesser peaks appear at 1/6 and 2/6, and at 4/6 and 5/6. The curve of the second one has prominent peaks at 1/3 and 2/3 of the measure, and somewhat less pronounced peaks at 1/6, 1/2 and 5/6. These findings clearly correspond with the musical notion of the importance of the different points in time given these meters.

\*\*\*\*\*  
 \*\*\*\*\* Insert figure 8 around here \*\*\*\*\*  
 \*\*\*\*\*

For the Longuet-Higgins model it is a bit difficult to 'clamp' the internal state to a partial solution because of possible backtracking. However, no backtracking can take place across beat boundaries, and after each beat the model only propagates the established meter and the length of a beat (the tempo) to the processing of the next beat, expecting them to apply there too. So given a beat length and a meter, they will determine the points in time where notes will be found and assigned to a metrical level. Together with the resolution of this decision (the tolerance), a comparable kind of expectancy of future onset times can be postulated. Of course the expectancy can only be given on an ordinal scale: onsets at higher metrical levels are expected 'more'. In figure 9 such a measure is shown for a twofold two-division (2/4 meter) and at a beat length of one bar, together with the expectancy curve of the connectionist model from figure 6. It is striking to see how the peaks in both curves now coincide. I feel that this is the point where the two models meet. Meter, a symbolic, structural concept at the very heart of the Longuet-Higgins parser, emerges out of the global and abstracted behavior of the connectionist quantizer. Here we are on the verge of the possibility of 'reading-out' symbolic representations from a sub-symbolic model.

\*\*\*\*\*

## Conclusion

It is possible to represent the behavior of two incompatible models of rhythm perception, the symbolic Longuet-Higgins musical parser and the Desain & Honing connectionist quantizer in different perspectives that make them comparable. These perspectives - the process state trace, the parameter and rhythm space, and the expectancy perspective - highlight different aspects of the models. Visualizations of these representations turned out to be crucial - even if the dimensionality or the flexibility had to be reduced.

These methods also showed the richness of the topic of quantization, a process that lies at the heart of rhythm perception. It is central because it separates two fairly different kinds of timing data: the discrete and the continuous, each of which forms the postulated input of different theories of temporal perception. It is well known that the concept of meter is of great importance in encoding, interpretation and memory in musical tasks (Palmer & Krumhansl, 1990) and it is not surprising that this symbolic concept, even though not represented explicitly in the connectionist model, is still present implicitly and can emerge from the net with the help of an appropriate measuring method.

## Acknowledgements

I would like to thank the colleagues who helped in this research: Eric Clarke for providing a very stimulating research environment at City University. Christopher Longuet-Higgins for encouragement and fruitful discussions about his parser. Steve McAdams for his support. Michel Koenders who automated the categorical perception experiment and Jeroen Schuijt who programmed a test suite for the quantizer, for their work and enthusiasm. Siebe de Vos and Peter van Oosten for commenting on drafts of this text. And especially Henkjan Honing for answering my midnight telephone calls.

## References

- Barucha J.J. & K. L. Olney (1989) Tonal Cognition, Artificial Intelligence and Neural Nets. Contemporary Music Review. 4
- Boulanger R. (1990) Conducting the MIDI Orchestra, part 1: Interviews with Max Mathews, Barry Vercoe, and Roger Dannenberg. Computer Music Journal 14(2)
- Clarke, E.F. (1987) Levels of Structure in Musical Time. Contemporary Music Review. 2(1)
- Clarke, E.F. (1987) Categorical Rhythm Perception: An Ecological Perspective. In A. Gabrielsson (Ed.) Action and Perception in Rhythm and Music. Stockholm: Royal Swedish Academy of Music, vol 55.
- Chowning, J., L.Rush, B. Mont-Reynaud, C. Chafe, W. Andrew Schloss, and J. Smith, (1984.) Intelligent systems for the Analysis of Digitized Acoustical Signals. CCRMA Report No. STAN-M-15.
- Dannenberg, R. B. and B. Mont-Reynaud (1987) An on-line Algorithm for Real Time Accompaniment. Proceedings of the 1987 International Computer Music Conference. San Francisco: Computer Music Association.
- Desain, P. and H. Honing (1989) Quantization of Musical Time: A Connectionist Approach. Computer Music Journal 13(3), also to appear in (Todd & Loy, forthcoming).
- Desain, P., H. Honing, and K. de Rijk (1989) A Connectionist Quantizer. Proceedings of the 1989 International Computer Music Conference. San Francisco: Computer Music Association.



Desain, P. and H. Honing (forthcoming) The Quantization Problem: Traditional and Connectionist Approaches. in Musical Intelligence edited by M. Balaban, K. Ebcioglu and O. Laske. Menlo Park: AAAI book.

Desain, P. (1990) Parsing the Parser, a Case Study in Programming Style. Computers in Music Research. 2.

Fodor, J.A. (1975) The Language of Thought. New York: Crowell.

Palmer C. and C.L.Krumhansl (1990) Mental Representations for Musical Meter. Journal of Experimental Psychology: Human Perception and Performance.

Longuet-Higgins, H.C. (1987) Mental Processes. Cambridge, Mass.:MIT Press.

Rumelhart, D.E. and J.E. McClelland (Eds.) (1986) Parallel Distributed Processing. Cambridge,: MIT Press.

Schulze, H. (1989) Categorical Perception of Rhythmic Patterns. Psychological Research. 51.

Todd, P.M. &D. G. Loy (Eds.) (forthcoming) Music and Connectionism. Cambridge, Mass.: MIT Press.

### Notes

<sup>1</sup> The solution is the rhythm:





Longuet-Higgins musical parser	Desain & Honing connectionist quantizer
symbolic central search hierarchical knowledge based	numerical distributed optimization heterarchical knowledge free

*Figure 1. A summary of the differences of the two models under study.*

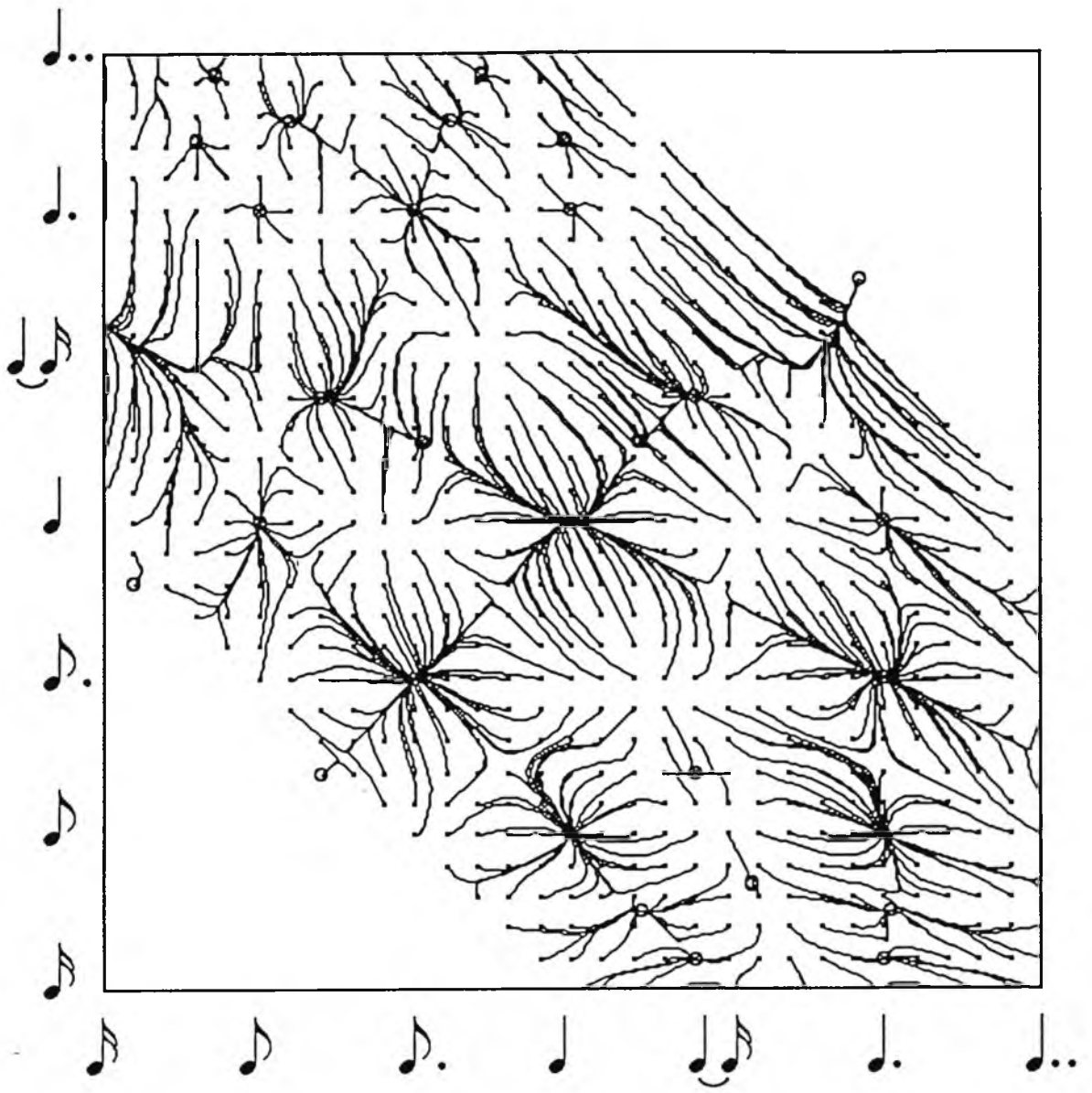


Figure 2a. Trajectories through rhythm space in the connectionist model.

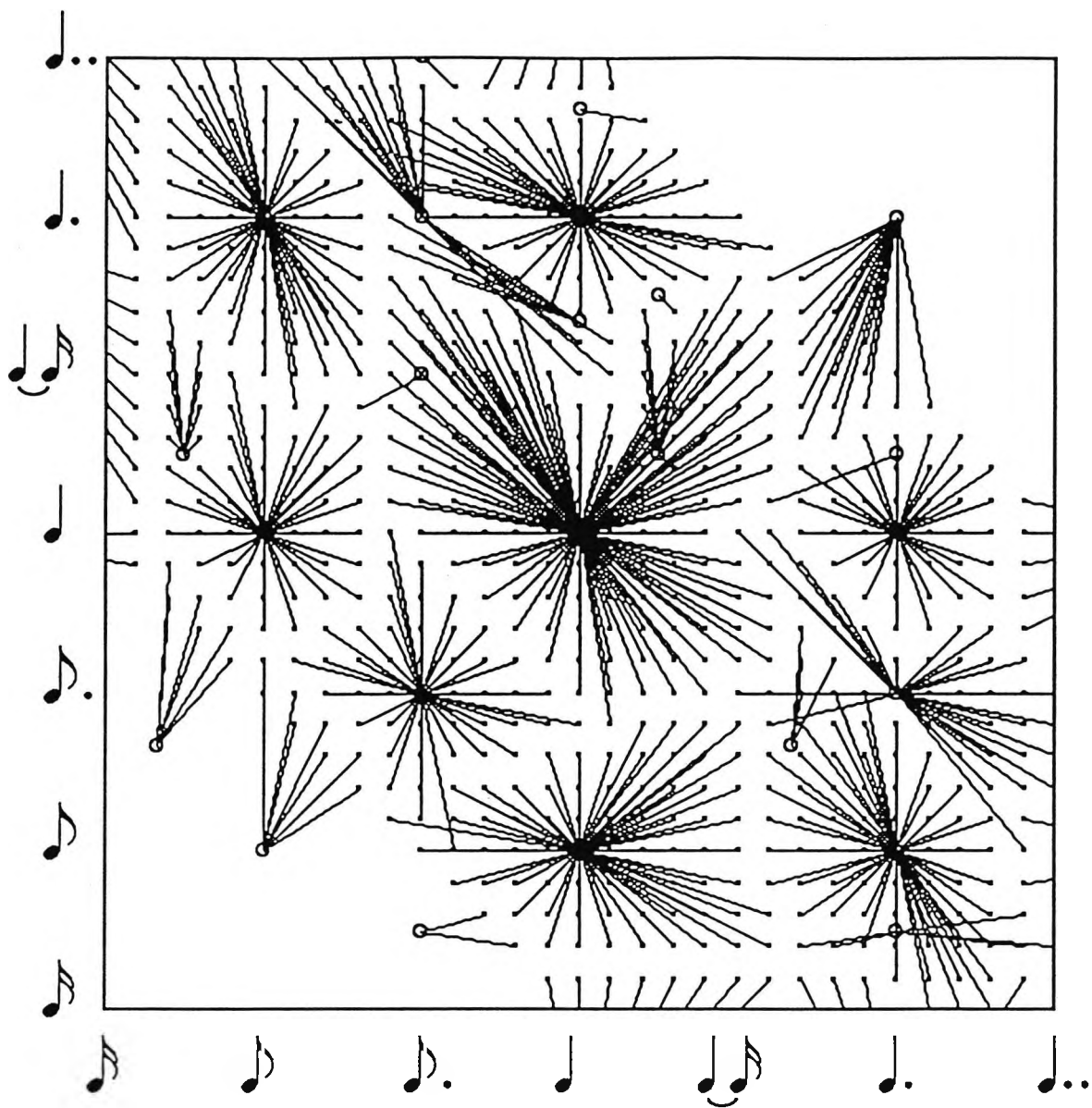


Figure 2b. Mapping in rhythm space in the traditional AI model.



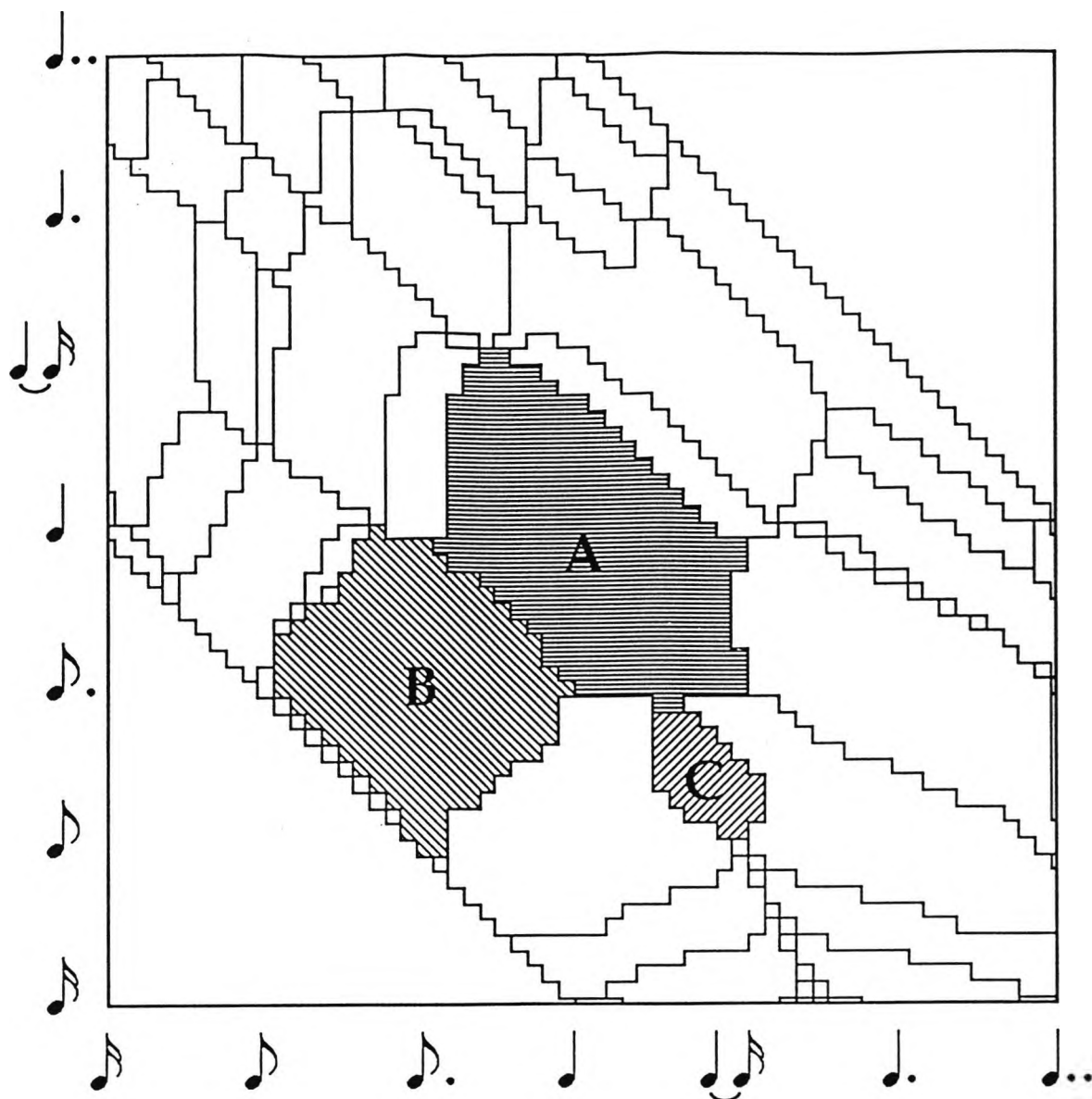


Figure 3a. Regions in rhythm space of the connectionist model.

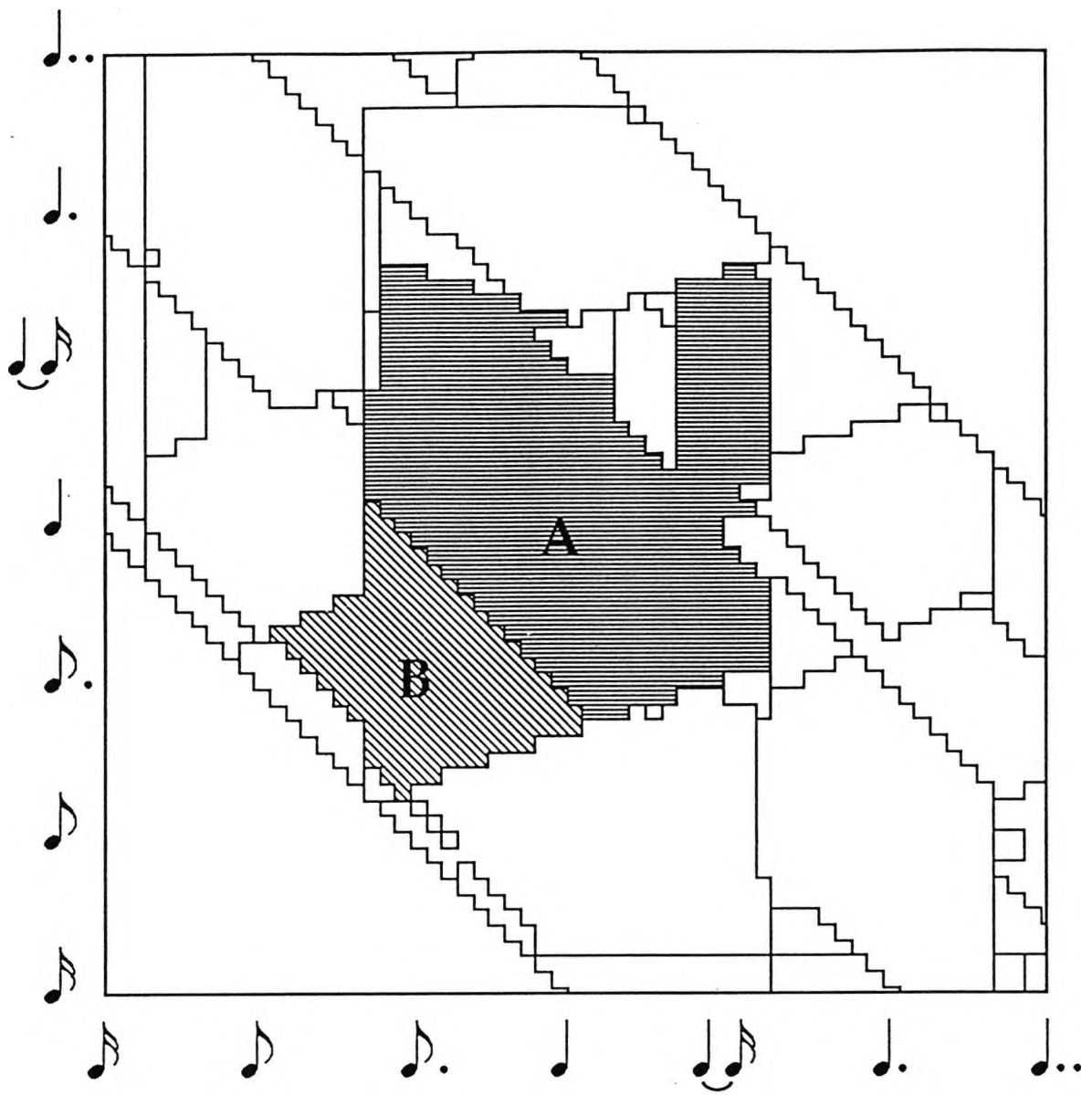


Figure 3b. Regions in rhythm space in the traditional AI model.

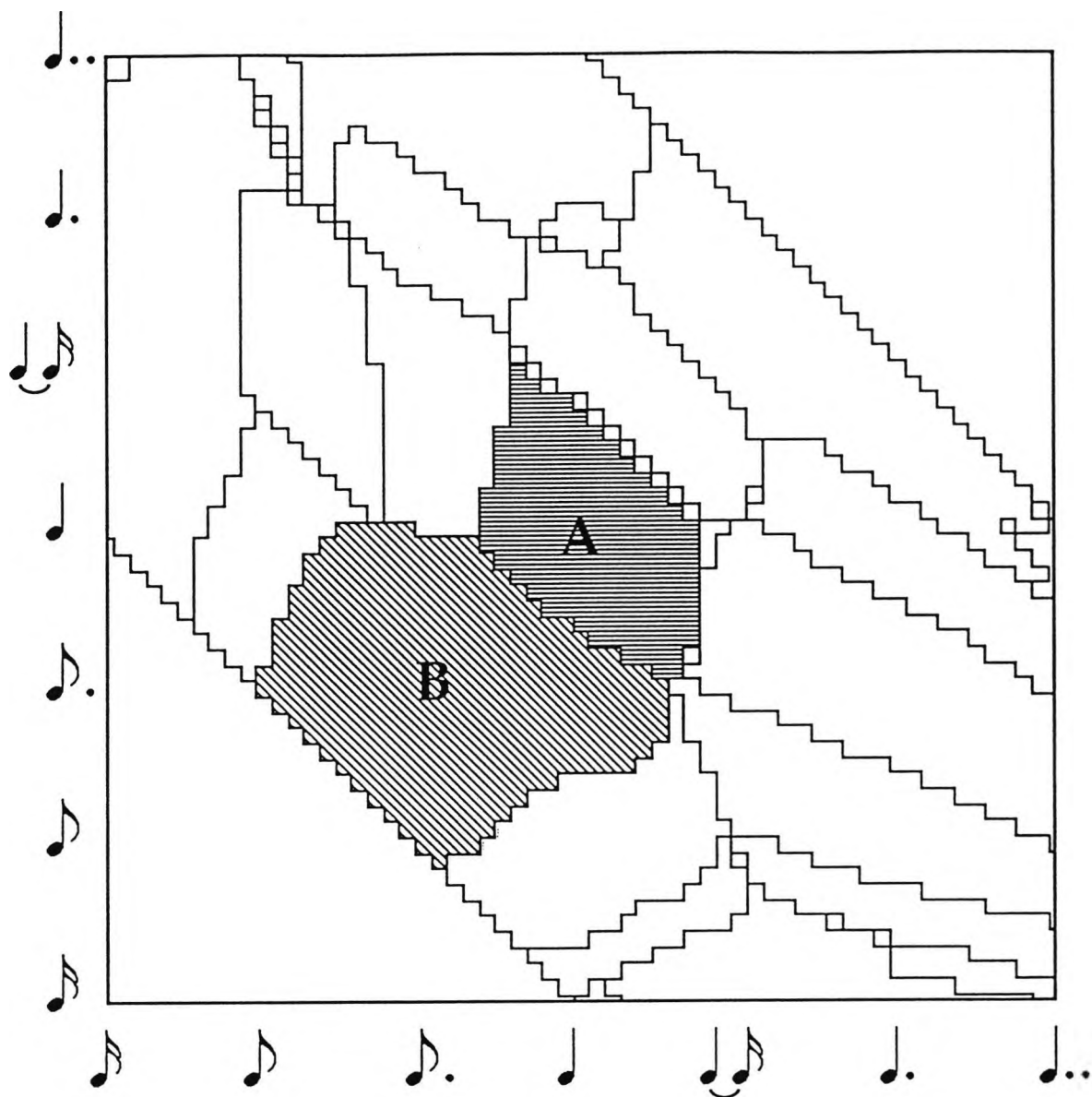


Figure 4a. Influence of context (two dotted quarter notes) in the connectionist model.



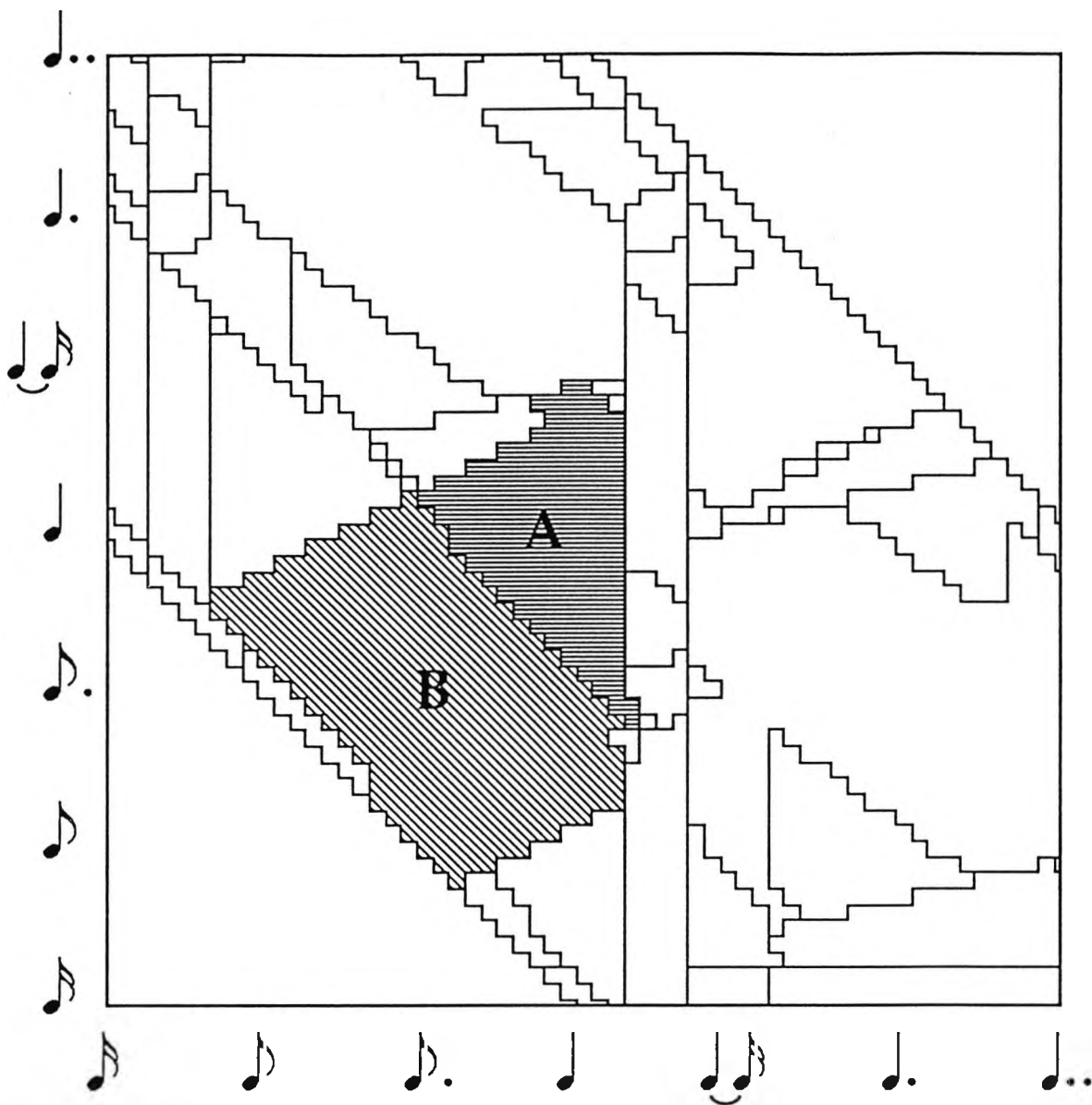


Figure 4b. Influence of context (duple meter) in the traditional AI model.

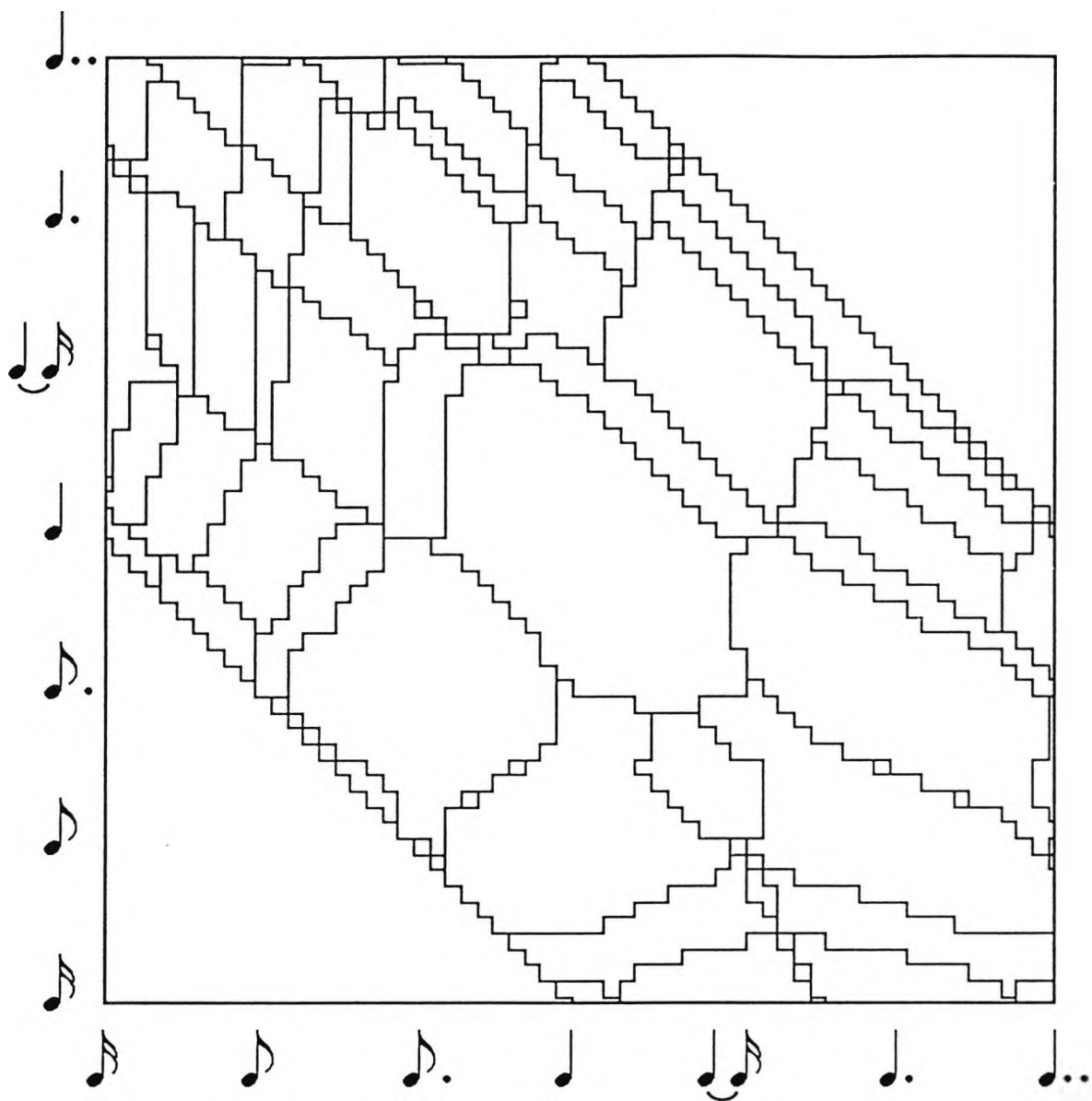


Figure 5a. Influence of the peak parameter in the connectionist model.

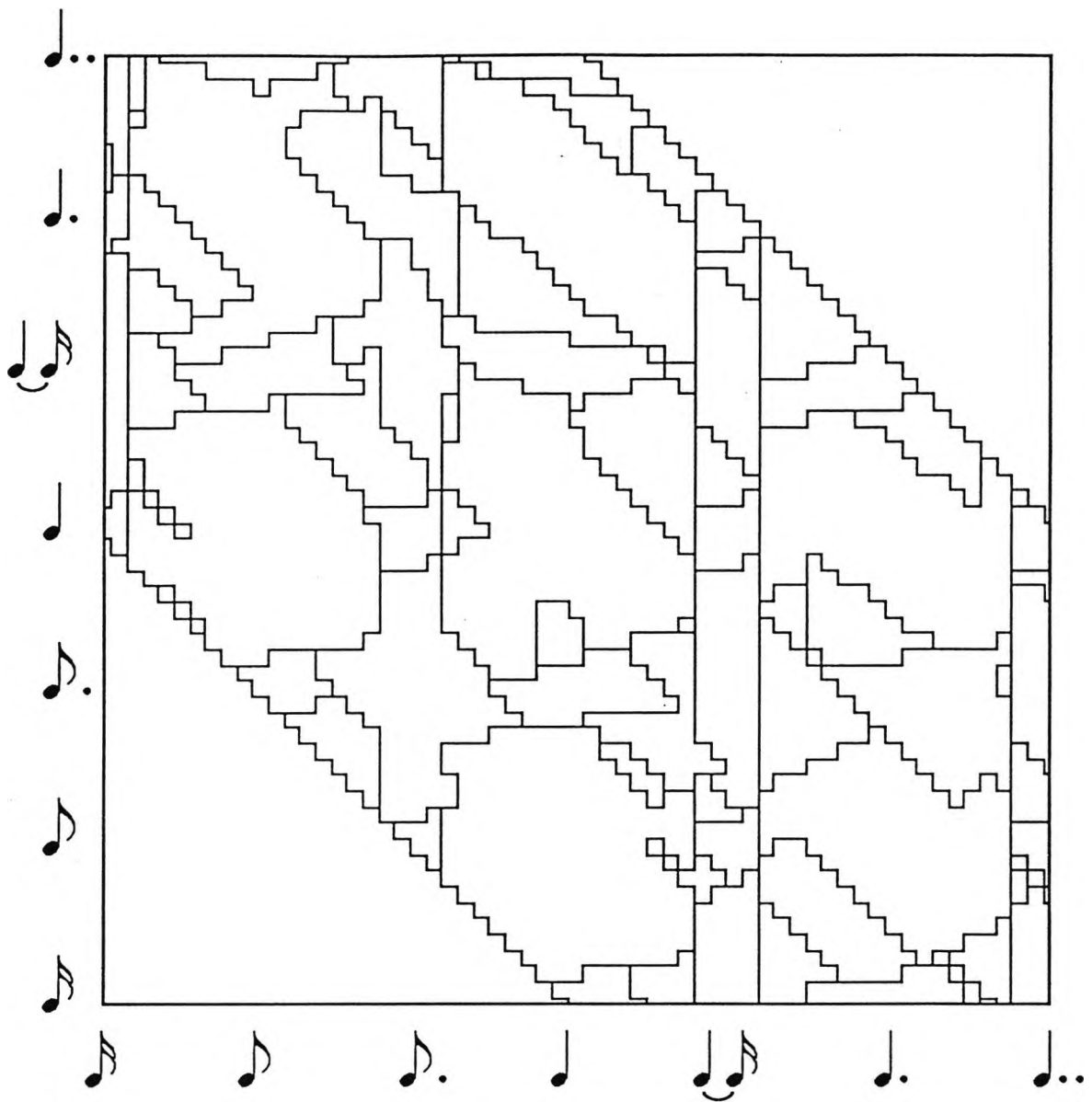


Figure 5b. Influence of the tolerance parameter in the traditional AI model.



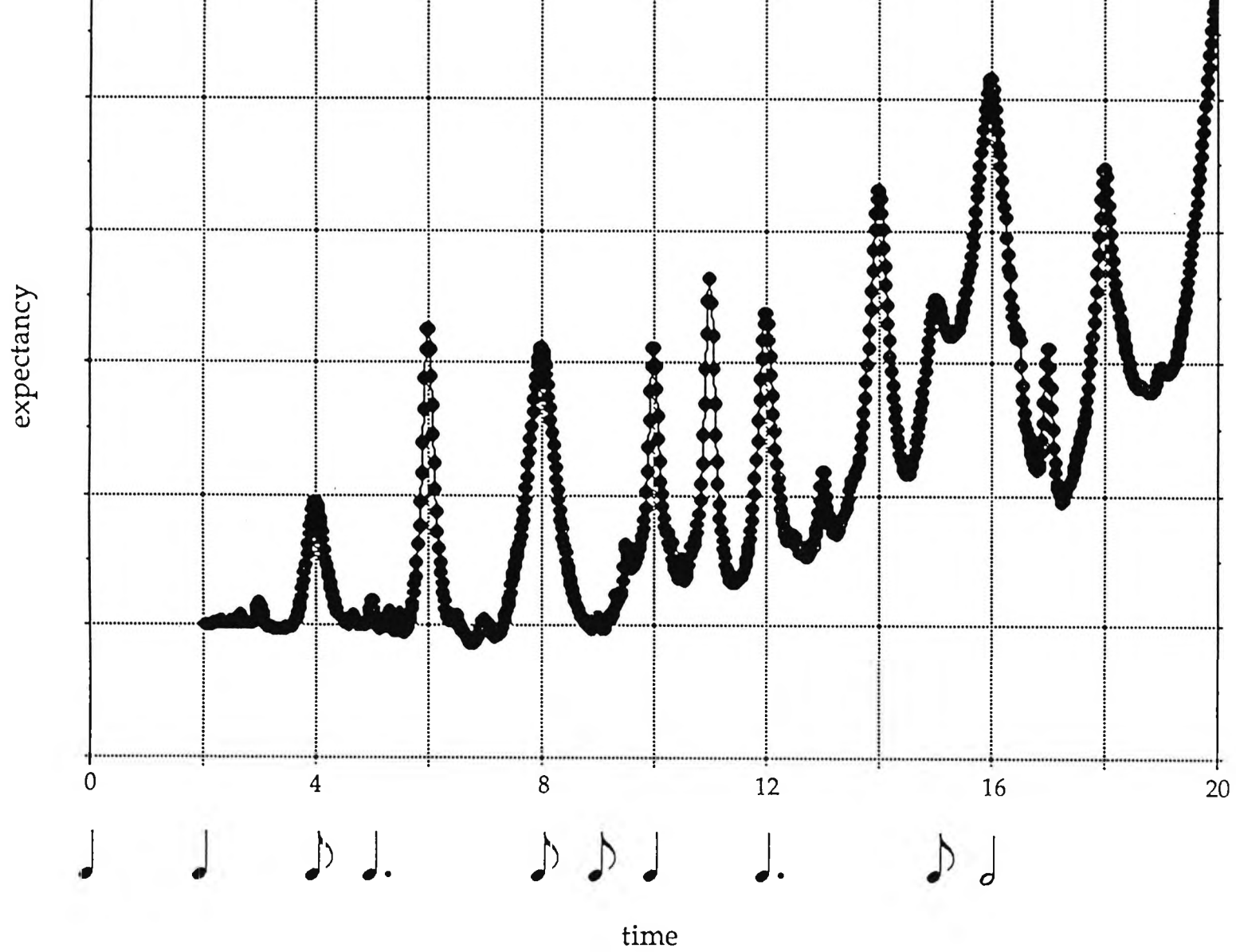


Figure 6. Expectancy of onsets in the connectionist model.

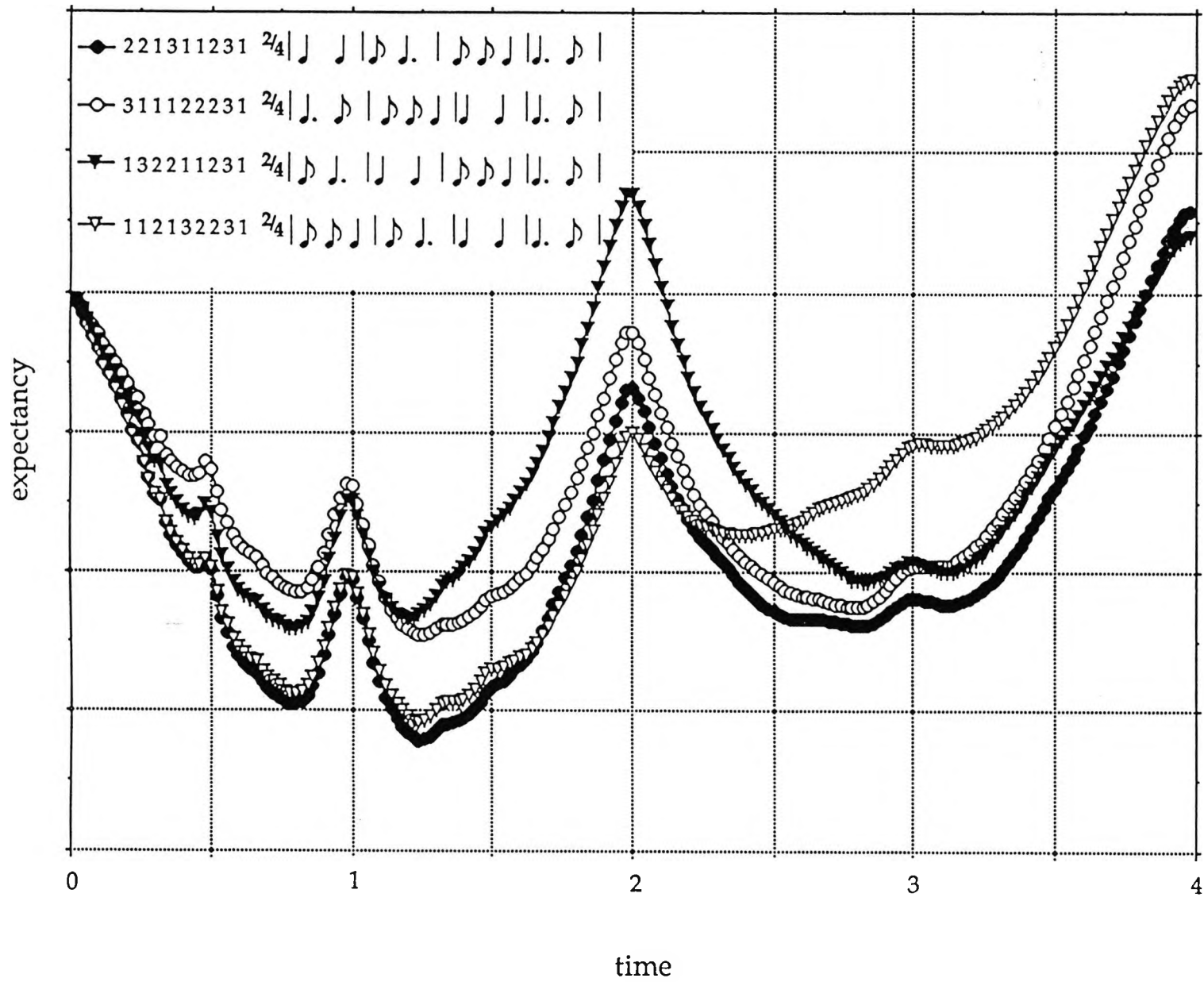


Figure 7. Expectancy of onsets in different 2/4 contexts in the connectionist model

21

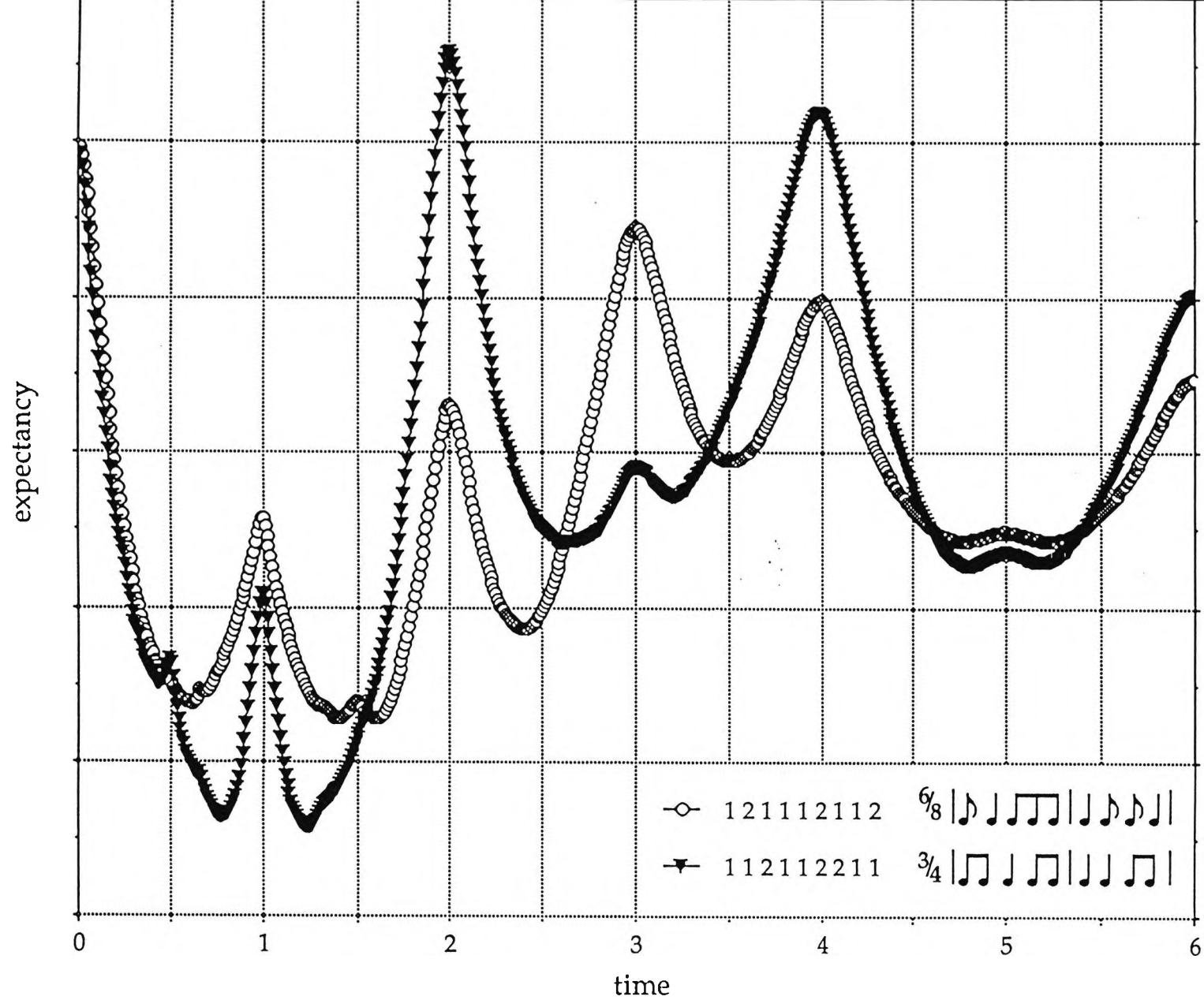


Figure 8. Expectancy of onsets in 6/8 versus 3/4 rhythm in the connectionist model.

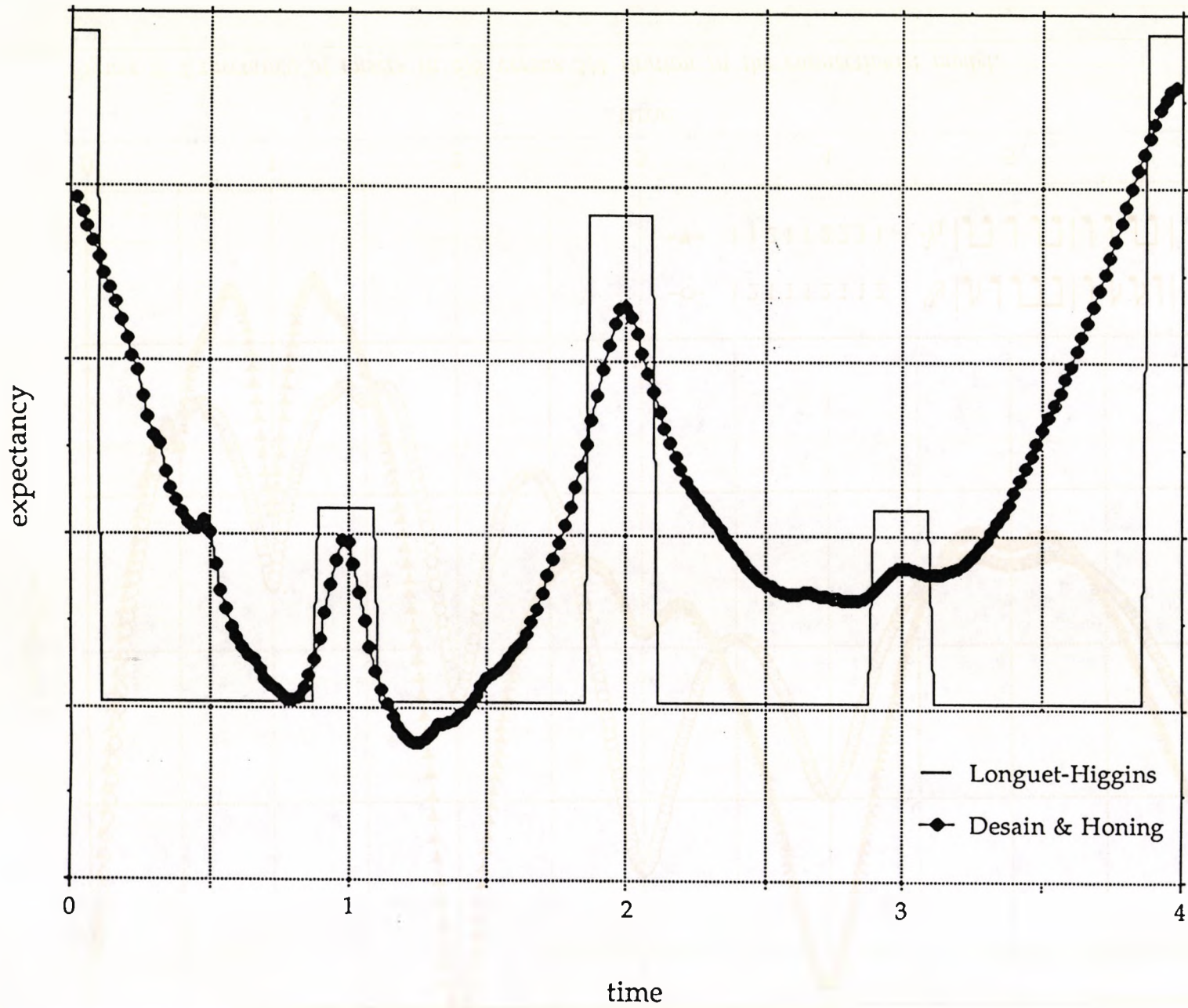


Figure 9. Expectancy of onsets in 2/4 context of the two models.

231



# A (DE)COMPOSABLE THEORY OF RHYTHM PERCEPTION

**Peter Desain**

Center for Knowledge Technology

Utrecht School of the Arts

Lange Viestraat 2B

NL-3511 BK Utrecht

*Will appear in Music Perception.*

**Keywords**

Rhythm perception, temporal patterns, expectancy, connectionism, quantization, categorical perception.

**Abstract**

A definition is given of expectancy of events projected into the future by a complex temporal sequence. The definition can be decomposed into basic expectancy components projected by each time interval implicit in the sequence. A preliminary formulation of these basic curves is proposed and the (de)composition method is stated in a formalized, mathematical way. The resulting expectancy of complex temporal patterns can be used to model such diverse topics as categorical rhythm perception, clock and meter inducement, rhythmicity, and the similarity of temporal sequences. Besides expectancy projected into the future, the proposed measure can be projected back into the past as well, generating reinforcement of past events by new data. The consistency of the predictions of the theory with some findings in categorical rhythm perception is shown.

## Introduction

Many incompatible theories about temporal perception and memory exist, which explain a number of phenomena well, but fail to predict others. A common theoretical basis for such work would be desirable. Connectionism might be an attractive paradigm in the search for such a basis, but most of its models lack compositionality. This means that the model as a monolithic whole might perform well, but it is impossible to decompose its complex behaviour into meaningful smaller parts. Chandrasekaran (1990) argues that the composability is a condition for successful cognitive modelling, even in the connectionist paradigm. In Desain (1990) the behavior of a sub-symbolic (connectionist) model of temporal quantization was described such that it could be compared with an incompatible symbolic model from the traditional AI paradigm. The paper concluded with an abstraction of the behavior of the quantizer in the form of an 'expectancy of events' with a temporal pattern as prior context. Expectancy turned out to be (de)composable which makes it possible to base a theory of perception of complex stimuli on a simple model for the perception of their constituting components. Because the expectancy concept seems to explain the dependency of perception of rhythmic structure on global tempo, the influence of context on categorical perception and other complex phenomena I propose to use it as a common basis for theories about temporal perception and memory. It is noteworthy that Povel (1984, 1985) has already remarked that high level cognitive judgements like rhythmicity might be based on, or be a byproduct of, low level rhythm perception processes that deal with quantization and tempo tracking.

*"The experience of rhythmicity is supposed to result from the process that updates the internal clock in the light of the incoming stream of events. It may be noted that this process makes part of the normal process of listening to music in which the listener*

*constantly adjusts his internal clock (metrical frame) to local temporal irregularities and tempo variations" (Povel,1984)*

In this paper I will focus on the explanation and formalization of the theory and the composability of the definition of expectancy. The interpretation of the resulting curves, their possible use, and their relations to other research aimed at a higher level of rhythm perception, will be dealt with in Desain (in preparation). Although the theory looks attractive enough, I have to warn the reader that this paper is an account of recent work and it has yet to be empirically verified.

#### **The Connectionist Quantizer**

To illustrate how the theory developed from a connectionist approach to the quantization of temporal sequences Desain & Honing (1989) a brief overview of that system will be given here. One by one the inter-onset intervals of a performed sequence are passed to a network of cells. The network acts as a complex shift-register: new inter-onset intervals shift in, are processed on the way through, and then shift out of the network as quantized durations (rhythmic categories). Besides cells for performed time intervals there are cells for intervals spanning several basic intervals. Two cells interact if they represent neighboring time intervals: one ending where the other starts. Their interaction is such that the ratio of the two intervals is adjusted towards an integer, if the ratio is already close to this goal. The change in length of each interval is proportionally propagated to all the basic intervals that form part of it. This interaction proceeds until a new inter-onset interval enters the network. All data is then shifted one position and the interaction process resumes.



## Expectancy

To make the link from the updating of time intervals, as is done in the connectionist quantizer, to expectancy, we can study what would happen to an imagined new incoming event whose corresponding time interval has just been shifted into the network. Its change is completely determined by the context of a temporal pattern that was presented before. When all but the last new interval in the network are clamped to a fixed state so that we can study the change of the new interval effectuated by the context while ignoring the influence of the new interval on the already perceived, but not yet fully processed data. The quantizer then can only effect a change in duration of this new basic interval. The imagined new onset ending this interval, now acting as a kind of measuring probe, will be moved to an earlier or a later time by the interactions. If the interval is increased, moving the onset to a later point in time, the model clearly has not yet 'expected' an event. If we postulate a measure of expectancy for this 'early' event, it will be larger at a later point in time. Conversely, a decrease of that time interval, a movement of the new onset to an earlier point in time, indicates a falling expectancy: the expectancy at an earlier time was larger. Finally, if the context inflicts no change to the new onset, expectancy is constant at that time. We can thus view the change of a imagined new onset time as the slope of an expectancy measure of an onset at that time. The pattern of expectancy forms a curve with extremes at places where an event, were it to happen there, would stay in place, because the derivative of the curve at this point (which is the change) is zero. The local maxima form the expected, 'perfect' places of onsets and the local minima form points of maximal confusion, places of unexpected events. Take e.g. the pattern [3, 1, 2, x]. If x is just below 1, the quantizer network would effectuate a positive change. If x is equal to 1, the change would be zero. If x is just above 1, the quantizer would adjust it downwards, a negative change. Around 2 a similar situation occurs, but the changes are more pronounced because of the strong 1:1 interaction between the last to intervals. Around 3, 6 and even 12 the pattern

is similar as well: positive below, zero on, and negative above that value. If we now integrate the change over  $x$ , an expectancy curve results with local maxima at the values 1,2,3 etc. Figure 1 shows some expectancy curves that were measured with longer temporal patterns represented as prior context in the network. They all have the same global characteristic, 2/4 meter, and produce very similar curves. Note the prominent peaks at the half-bar and bar boundary (at time 2 and 4, counting in eighth notes) and lesser peaks dividing these intervals further.

Figure 1 about here: *Expectancy of onsets after presentation of different 2/4 patterns.*

### Decomposition

Expectancy is defined as the integral of the change generated by the sum of all interactions in the network. We can exchange integration and summation in this formulation and redefine the expectancy, given a prior complex pattern, as the sum of all the expectancies generated by each interval in that pattern. This effectively decomposes the theory for processing complex rhythms into a set of simple components, one for each time interval implicit in the pattern. Figure 2 is an attempt to depict this kind of decomposition. At the bottom left the presented temporal pattern is shown. Above it all the intervals implicit in this pattern are indicated (by heavy lines). To the right of each interval the pattern of expectancy projected by that interval is shown (light lines). This basic expectancy is a function of two parameters: the length of a time interval, and the time elapsed after the end of that interval. It peaks when the second parameter is an integer divisor or an integer multiple of the first. All the projected basic expectancies are summed and yield the curve at the lower right: the global expectancy curve projected by the complex temporal pattern. It has peaks at time points that can be considered as 'good continuations' of the pattern presented.

Figure 2 about here: *(De)composition of expectancy.*

This concept of expectancy seems closely related to ideas of Jones, and could function as a formalization of these.

*"[...] It is such psychological trajectories that rhythmically guide our attentional energies along ideal paths. Attention is cast from some reference event at one point in time toward a target event scheduled for a later time. This approach demonstrates that attention itself is a dynamic, many levelled affair based upon nested internal rhythms. We continually cast ourselves forward by rhythmically anticipating future events that may occur within small and larger time intervals. These paths form the patterns of mental space and time and so can establish for us that sense of continuity and connection that accompanies comprehension."* Jones (1981).

A more precise formulation of this principle of decomposition, which constitutes the core of the theory presented in this paper, will be given later. But first it is interesting to consider what happens if the grain of analysis is made a bit more coarse by lumping together the expectancy contributions of intervals that end at the same point in time. This gives a decomposition based on events instead of intervals. In Figure 3 the contributions of each new event are shown, incrementally building the total expectancy. The expectancy evolves during the presentation of the pattern and not only extends from the end of the pattern into the future, as was shown in figure 2. This kind of curve can show how a temporal pattern fails to realize a high expectancy (a syncope), or comes up with an unexpected event. It also enables one to see how a new event reinforces the already existing pattern of future expectancy or introduces new elements in it.

Figure 3 about here: *Expectancy contributions of events in a temporal pattern.*

Now we will make a slight detour to the concept of memory. According to Mari Jones expectancy and memory are closely related:

*"[...] Paradoxically, a third implication of including time as a part of subjective structure results in an alternative view of memory. This new view casts remembering as a dynamic attentional process unfolding in negative time. That is, we can conceive of both expectancy and remembering as activities tied to the time dimension. [...] Expectancy and remembering then are opposite sides of the same coin."* Jones (1981).

These rather puzzling remarks become clearer when one studies the influence that a new incoming event might have on the prior context. The new event can support one or the other of the previous interpretations and in retrospect contribute to a limited extent to the salience of already perceived stimuli. To visualize this we can simply construct all the new intervals created by a new incoming event and project their expectancy of events into the past (see Figure 4). This gives the amount of reinforcement given to each event in the pattern by the new incoming event. Temporal patterns that behave in a well-formed way, with high support of events by later ones, might be remembered better. This model thus predicts how a later event can facilitate or inhibit the memory of past ones, a rather spectacular feature. An example of this phenomenon is the often encountered 'closure' of a temporal pattern which concludes with an event in a highly expected place: an important metrical position.

Figure 4 about here: *Relation of expectancy and memory.*



Concluding, we can state that the concept of expectancy as presented here has no time direction in itself. It is determined completely by two time intervals, but whether each of the time points marking the intervals was presented as stimulus in the past, or has yet to happen in the future, is irrelevant to the theory. One can thus speak about expectancy of an event at a future time generated by two time points in the past, the reinforcement of an event in the past by two time points happening later, or even the expectancy of an event at a certain time in between two time points. All these notions are equivalent at this level of the theory, and yield the same numerical values if the distance between the first and second time point is the same as the distance between the second and third time point in the three cases. This does not imply that the theory is symmetric with respect to time: swapping the distances between the first and second and between the second and third time point might yield different values, because the perception of a time interval followed by a multiple of its length might be different from the perception of that interval followed by a division of it by the same factor.

Besides the ratio of the two time-intervals, the basic expectancy function is supposed to depend on the absolute time duration of both of its parameters as well. This makes the theory sensitive to the time scale used (the absolute tempo). In Figure 5 the same pattern as in Figure 2 is used, but at half the tempo (note that for the sake of easy comparison the horizontal axis is 're-normalized' such that the figures have the same size). It is clear that what is often called 'the shift of level of attention through the levels of metrical hierarchy, prompted by different tempi' can be found here in the shift in relative importance of expectancy peaks.

Figure 5 about here: *Expectancy at slow tempo.*

We now get back to the details of the model. The shape of the basic expectancy curves is the part of the model that still has to be 'plugged in' to yield the full theory. The theory is 'generic': given any method for calculating the basic expectancy of a time interval pair, it defines the method to calculate expectancies for any complex temporal pattern. I will discuss a first approximation of these basic expectancy curves and the possibilities of deriving them empirically.

### Preliminary theory of basic expectancy

Before its mathematical formulation I will first give a graphic representation of the proposed basic expectancy. We can depict the basic expectancy of a time interval pair  $(A,B)$  for fixed  $A$  as a curve (see figure 6).

Figure 6 about here: *Basic expectancy of interval pair  $A,B$ .*

One can see clear peaks in expectancy when  $B$  equals  $A, 2A, 3A...$  and when  $B$  equals  $A/2, A/3$  etc. The shape of the expectancy curve is determined by our capacity to perceive serial duration ratios, with higher ratios being more difficult and less expected (Jones & Boltz, 1989). This may also be true for more complex ratios in terms of their prime divisors. These curves, projecting expectancy into the future, were used for Figure 2. Another visualization is given in Figure 7. It shows the expectancy of a subdivision of a unit time interval into the interval pair  $A,1-A$ . One can see here that the time-reversed interval pairs are still assigned the same expectancy in this preliminary theory (e.g. the pair  $1/3, 2/3$  and the pair  $2/3, 1/3$ )

Figure 7 about here: *Basic expectancy of interval pair  $A,(1-A)$*

Sternberg, Knoll and Zukofsky (1982) showed that perceptual judgement is dependent both on the ratio of the intervals involved and on their absolute duration. Very long and very short time intervals are difficult to perceive accurately. The maximum in sensitivity occurs at about 600 ms, which is in the preferred tempo range (Fraisse, 1982). The total length of the interval pair will be used as a second determinant of the expectancy to model this dependency on the absolute time scale.

Because the sectionwise polynomials used in Desain & Honing (1989) are a bit difficult to treat mathematically, basic expectancy is defined as a sum of several Gaussian curves, one around each relevant ratio.

$$E_b(A,B) = \sum_{R \in \{\frac{1}{n}, \dots, \frac{1}{2}, 1, 2, \dots, n\}} \text{GAUSS}\left(\frac{A}{B} - R, R, \frac{A+B}{T_{\text{pref}}}\right) \quad (1)$$

$$\text{GAUSS}(x,R,S) = C(R,S) e^{-D(R,S) x^2} \quad (2)$$

The parameters of the Gaussians are determined by the ratio and the absolute tempo (the size of the sum interval in proportion to the preferred tempo). These last two values (R and S) determine the height (via function C) and the width (via function D) of the expectancy curve peak at each integer ratio.

### Measuring basic expectancy

To be able to proceed from a theory to a tested cognitive model we need a way to operationalize and measure basic expectancy. Although the material under consideration is very simple (just two successive temporal intervals), collecting empirical data on perceived expectancy might still be quite difficult. Carolyn Drake (personal communication) has

proposed a measure of accuracy in an adjustment task as used in her work on accents (Drake, Botte & Gérard, 1989). Goodness-of-fit judgements and memory confusion in discrimination judgments as used by Palmer and Krumhansl (1990) should also be considered. A more indirect measure might be derived from calculating the probabilities of time interval pairs A,B in a body of musical pieces. This is not the same as the frequency counts approach used by Palmer and Krumhansl (1990) as the latter makes use of a-priori knowledge of meter.

### Complex expectancy

When the basic expectancy function  $E_b(A,B)$  is given, either by measurement or construction, the expectancy generated by an interval in a complex temporal pattern can be derived. Let  $X$  be a vector of basic time intervals  $X_i$  ( $1 \leq i \leq N$ ) and  $S(X,p,q)$  the time period spanned by intervals  $p$  through  $q$ .

$$S(X,p,q) = \sum_{i=p}^q X_i \quad \text{with } p \leq q \leq N \quad (3)$$

When  $X$  contains the interonset intervals of a temporal fragment presented from time 0, we can define the interval expectancy  $E_i$  of a new onset at time  $T$ , generated by the interval spanning inter-onset intervals  $p$  to  $q$  in  $X$ , by applying the basic expectancy function  $E_b$  to the relevant time intervals.

$$E_i(X,p,q,T) = E_b(S(X,p,q), T-S(X,1,q)) \quad \text{with } T \geq S(X,1,q) \quad (4)$$

The complex expectancy  $E(X,T)$  of an event at time  $T$ , generated by the temporal pattern  $X$  presented from time 0, is then a sum of the interval expectancies  $E_i$  over all intervals implicit in that pattern:



$$E(X,T) = \sum_{p=1}^N \sum_{q=p}^N E_i(X,p,q,T) \quad \text{with } T \geq S(X,1,N) \quad (5)$$

Figure 8 has the same structure as Figure 2, but it labels the relevant time intervals according to the formalism given above.

Figure 8 about here: *Time intervals used in calculating expectancy.*

The event-based expectancy of Figure 3 and the concept of reinforcement of Figure 4 may be formalized in an analogous way.

Because the basic expectancy  $E_b(A,B)$  will be small for large  $A$  or  $B$  there is a natural limit to the size of the context that will contribute to the total expectancy  $E(X,T)$ , and the vector  $X$  can function automatically as a short-term memory construct.

A nice consequence of the bidirectionality of the expectancy concept is that the sum of the corroborations of each event in a pattern by a virtual new onset is the same as the expectancy of that onset generated by the pattern.

#### Related work

The measure of expectancy presented here can be related to a number of different theories and concepts. The Gestalt principle of Good Continuation, which is one of the underlying assumptions in much research in grouping mechanisms (Deutsch, 1982), can be linked closely to the expectancy construct. It is tempting to interpret the relative height of the peaks in the expectancy curve directly as a measure of metrical boundary strength. However, instead of deriving a symbolic notion of meter from these curves, it might be more productive to re-think the concept of meter as a continuous concept, an idealized expectancy curve, as discussed in Desain (in preparation). This allows the construct to be applied directly to the difficult areas of ambiguous rhythms, change of meter and amount of metricality. The

expectancy curves are also promising for the study of the perceived rhythmicity of temporal patterns and their degree of syncopation (Povel, 1985). This is because it is easy to formalize the violation of a maximum in expectancy by the absence of an event at that time.

Given the numerous links with different aspects of the literature on rhythm perception, I will restrict myself here to some remarks on the predictions of the model for categorical rhythm perception.

### Categorical rhythm perception

In general, categorization occurs when objects, on the basis of some continuously variable attribute, are placed in a small number of groups (see Repp, 1984). Sloboda argues for the existence of categorical perception in rhythm by noting how different the perception of rhythm and the perception of expressive timing are:

*"[...] Identification of intended rhythm is a commonplace accomplishment for listeners, who are continually faced with the potentially confusing phenomenon of rubato and gradual changes in speed. In contrast, accurate perception of deviations from metricallity is difficult, and requires much specific training. It is almost impossible for one performer to imitate another exactly. All this strongly suggests that listeners achieve a categorization of the duration of the notes they hear into crotchets, quavers etc. [...] one would not wish to claim that categorical perception makes finer temporal discriminations impossible. We can hear rhythmic imprecision and rubato with appropriate training, but fine differences in timing are more often experienced not as such, but as differences in the quality (the 'life' or 'swing') of a performance. "* (Sloboda 1985, p. 30)

Thus, in quantization the deviations from a strict metrical performance are not thrown away, but timing is separated into structural and expressive components and then handled by different processes.

A discussion of the use of expectancy curves for a categorical perception model (a quantizer) will be left aside here since it involves many technical points about the architecture of static vs. process models, the use of global tempo tracking etc. The models proposed in Desain & Honing (1991, addendum) use hill-climbing in an expectancy landscape, but they vary in the extent to which the different dimensions ('older' and 'newer' time intervals) are allowed to vary. The following material will discuss the expectancy curves themselves, assuming that if local maxima and minima emerge they can be used somehow to segment the continuous time axis into discrete regions, one for each rhythmic category.

Although categorical perception is a well established phenomenon in speech research, its existence is much harder to demonstrate in the rhythm domain. In a well known set of experiments, skilled musicians were asked to judge the length of temporal intervals in rather simple integer ratios (Sternberg, Knoll & Zukofsky, 1982). Surprisingly, they were not able to do this accurately and boundaries between different categories turned out to be rather vague. Schulze (1989) was somewhat more successful in showing the existence of a categorical boundary in interpolations between patterns of three inter-onset intervals. In Clarke (1987) subjects were given a context of five or six inter-onset intervals in different metrical contexts, before the two experimentally manipulated durations were presented. He was able to show clear identification and discrimination curves, that also showed the predicted shift for the different context conditions. The general line in these findings thus seems to be that categorical perception is facilitated by context. This is in agreement with the expectancy theory presented above: the expectancy curves become more pronounced if more context is available. In Figure 6 one can see that in an impoverished context the perception of simple patterns like [3,2] is not possible anyhow: there is no peak in

expectancy at  $B = 2/3A$ . A bit more context will allow correct perception of a  $2/3$  ratio as is shown in Figure 9, where the expectancy of an onset after the context [2,3,3] is given. Note how the local maximum at  $B = 2$  emerges here. This general idea is consistent with the findings of Povel (1981) concerning an imitation task.

Figure 9 about here: *Facilitation of the perception of the ratio 2/3 by context.*

### Conclusion and discussion

The presented theory of expectancy seems a promising candidate for a common basis for many incompatible theories of rhythm perception and memory. Its decomposability into simple components that model perception of time interval pairs is attractive, not in the least because empirical results for simple stimuli can be 'plugged' into the theory to yield predictions for more complex temporal patterns. The theory elegantly links expectancy projected into the future and reinforcement of past events by new data. Predictions following from the theory are consistent with some findings in categorical rhythm perception.

Empirical verification of the theory will be the next step that is needed to further the research in this direction. Another field in which work needs to be done is the formalization of the use of expectancy in the different cognitive processes mentioned above, such as quantization, meter and beat inducement, rhythmicity and similarity of rhythms. It is clear that a full theory of rhythm perception cannot be based on time alone but has to take other musical parameters into account as well. A possible approach could be the use of a notion of salience to weigh expectancy contributions of events.

It still is an open question whether an expectancy pattern is available as a whole for input to other processes, or whether expectancy is merely a changing sense of present anticipation and no access to future expectations is possible.



### Acknowledgements

I would like to thank Piet Vos for organizing the Rhythm Perception and Production Workshop in Horssen and Ian Cross for his Music and Cognition Conference in Cambridge where many of these ideas were born. The research done with Eric Clarke at City University started my interest in issues of expressive timing, it was very stimulating to work with him. Johan den Biggelaar of the Utrecht School of the Arts did his best to provide me with research facilities there. I also would like to thank Carolyn Drake, Mari Jones, Jeroen Schuit, Marie-jose Tienhooven, Siebe de Vos and Caroline van der Wal who gave many helpful comments on this paper. As always, Henkjan Honing prevented me from sidetracking the main issues. Without him, I would have had less than half the fun in contriving these ideas.

## References

- Chandrasekaran, B. What Kind of Information Processing is Intelligence? A Perspective on AI Paradigms and a Proposal. In T. Partridge and Y. Wilks (Eds.), *The foundations of artificial intelligence, a sourcebook*. Cambridge: Cambridge University Press, 1990.
- Clarke, E. Categorical Rhythm Perception, an Ecological Perspective. In A. Gabrielsson (Ed.), *Action and Perception in Rhythm and Music*. Royal Swedish Academy of Music, 1987, No. 55:19-33.
- Desain, P. & Honing, H. Quantization of musical time: a connectionist approach. *Computer Music Journal*, 1989, 13(3) to be reprinted in P.M. Todd and D. G. Loy (Eds.), *Music and Connectionism*, Cambridge, Mass.: MIT Press, 1991.
- Desain, P. A Connectionist and a Traditional AI Quantizer, Symbolic versus Sub-symbolic Models of Rhythm Perception. In I. Cross (Ed.), *Proceedings of the 1990 Music and the Cognitive Sciences Conference*, *Contemporary Music Review*. London: Harwood Press.
- Desain, P. Meter as a Continuous Concept. Report of the Center for Knowledge Technology. Utrecht. (in preparation) .
- Deutsch D. Grouping Mechanisms in Music In D. Deutsch (Ed.), *The Psychology of Music*. Orlando: Academic Press, 1982.
- Drake, C. , Botte, M. C. & Gerard, C. A perceptual Distortion in Simple Musical Rhythms. *Proceedings of the International Society for Psychophysics Fifth Annual Meeting*, Cassis, France, 1989.

- Fraisse, P. Rhythm and Tempo In D.Deutsch (Ed.) *The Psychology of Music*. Orlando: Academic Press, 1982.
- Handel, S. *Listening, An Introduction to the Perception of Auditory Events*. Cambridge Ma: MIT Press, 1989.
- Jones, M.R. & Boltz, M. Dynamic Attending and Responses to Time. *Psychological Review*. 96(3), 1989, 459-491.
- Jones, M.R. Only Time Can Tell: On the Topology of Mental Space and Time. *Critical Inquiry*, 1981 .
- Palmer C. & Krumhansl, C.L. Mental Representations for Musical Meter. *Journal of Experimental Psychology: Human Perception and Performance*. 16(4) 1990, 728-741.
- Povel D.J. Internal Representation of Simple Temporal Patterns. *Journal of Experimental Psychology: Human Perception and Performance*. 1981 , 7(1), 3-18.
- Povel D.J. Time, Rhythms and Tension: in search of the determinants of rhythmicity. Internal Report 84FU11, University of Nijmegen, 1984 .
- Povel D.J. Time, Rhythms and Tension: In Search of the Determinants of Rhythmicity In: Michon J.A. & Jackson, J.L. (eds.) *Time, Mind and Behavior*. Berlin: Springer Verlag. 1985.

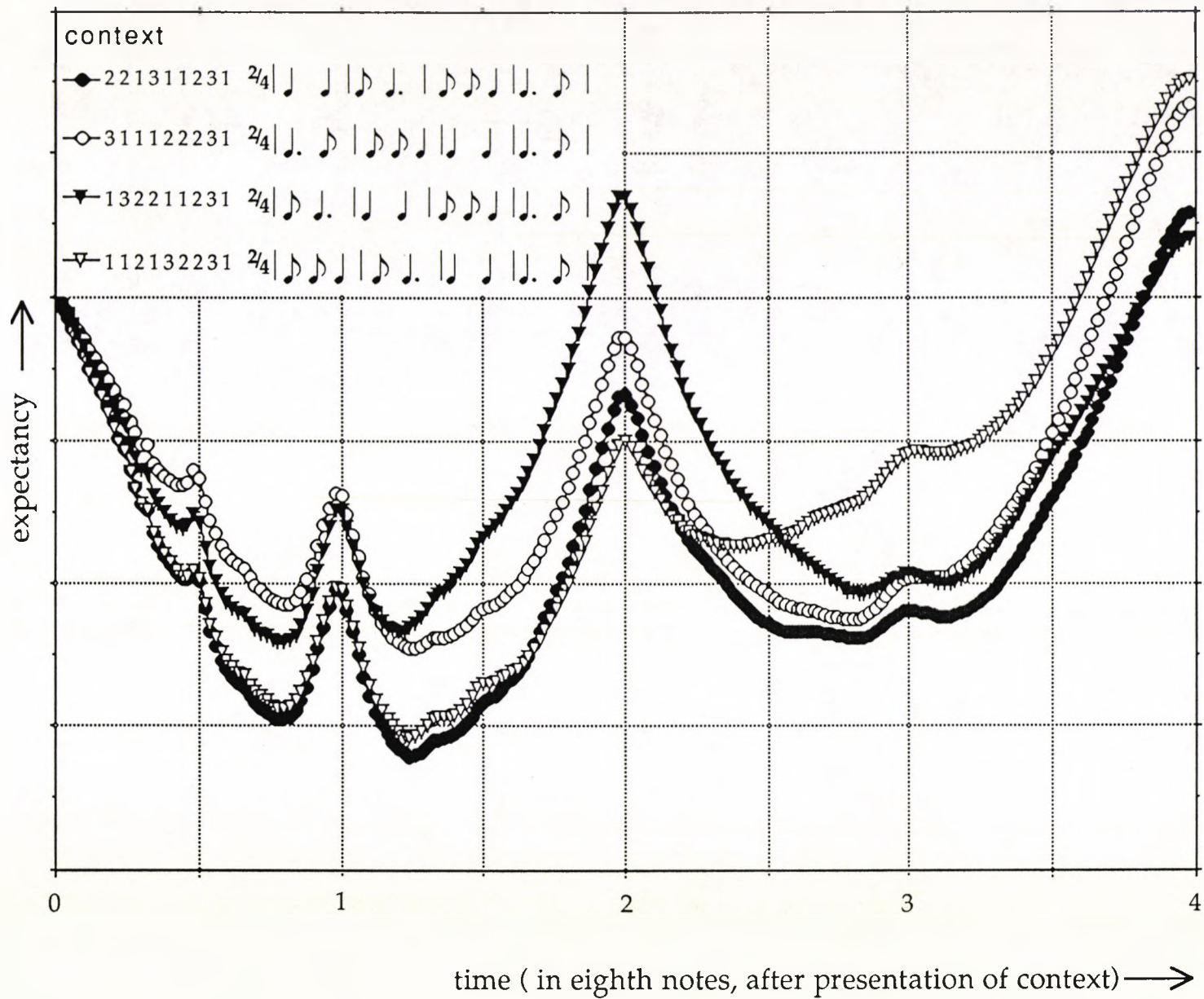
Repp, B.H. Categorical perception: Issues, methods and findings. In N. Lass (Ed.) *Speech and Language*. Vol 10: Advances in basic research and practice. Orlando Fla: Academic Press. 1984 .

Schulze, H. Categorical Perception of Rhythmical Patterns. *Psychological Research*, 1989, 51.

Sloboda, J.A. *The Musical Mind: The Cognitive Psychology of Music*. Oxford: Oxford University Press, 1985.

Sternberg, S. Knoll, R.L. and P. Zukofsky Timing by Skilled Musicians. In D.Deutsch (Ed.) *The Psychology of Music*. Orlando: Academic Press, 1982.





20

Figure 1. Expectancy of onsets after presentation of different 2/4 patterns.

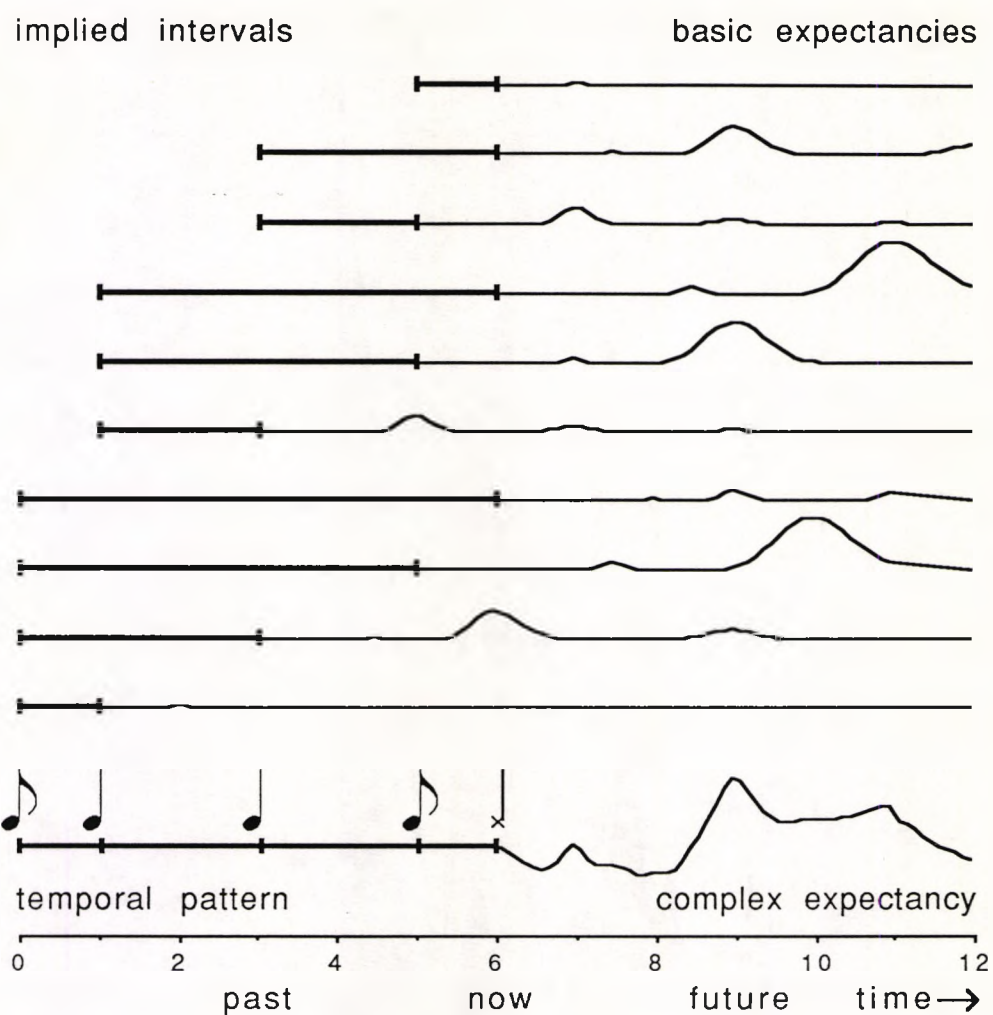


Figure 2. (De)composition of expectancy.

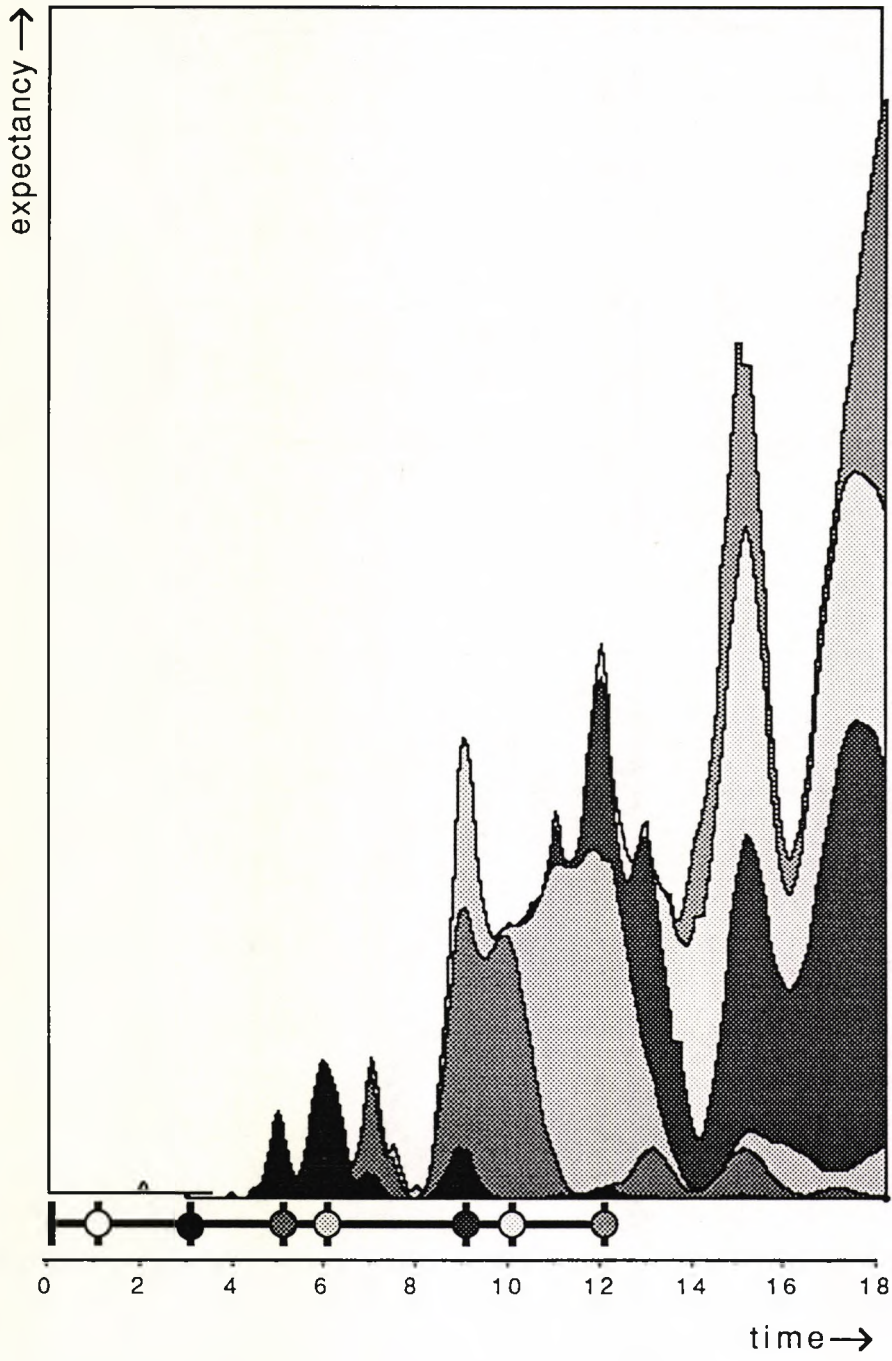


Figure 3. Expectancy contributions of events in a temporal pattern.



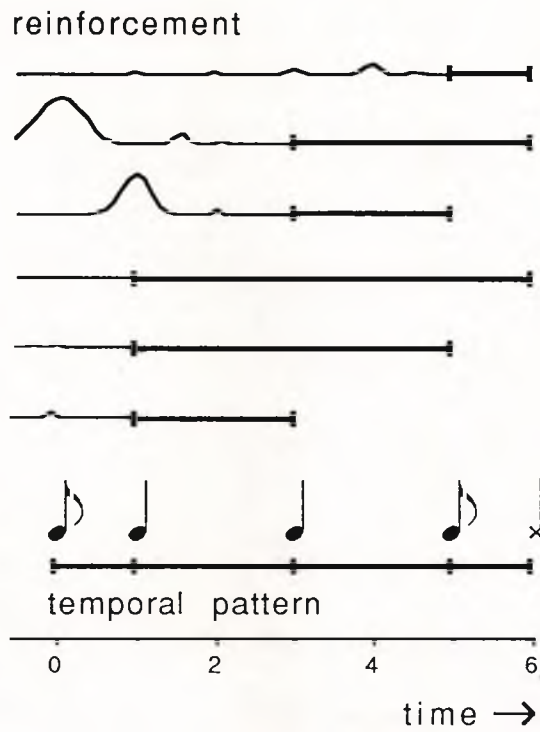


Figure 4. Relation of expectancy and memory.



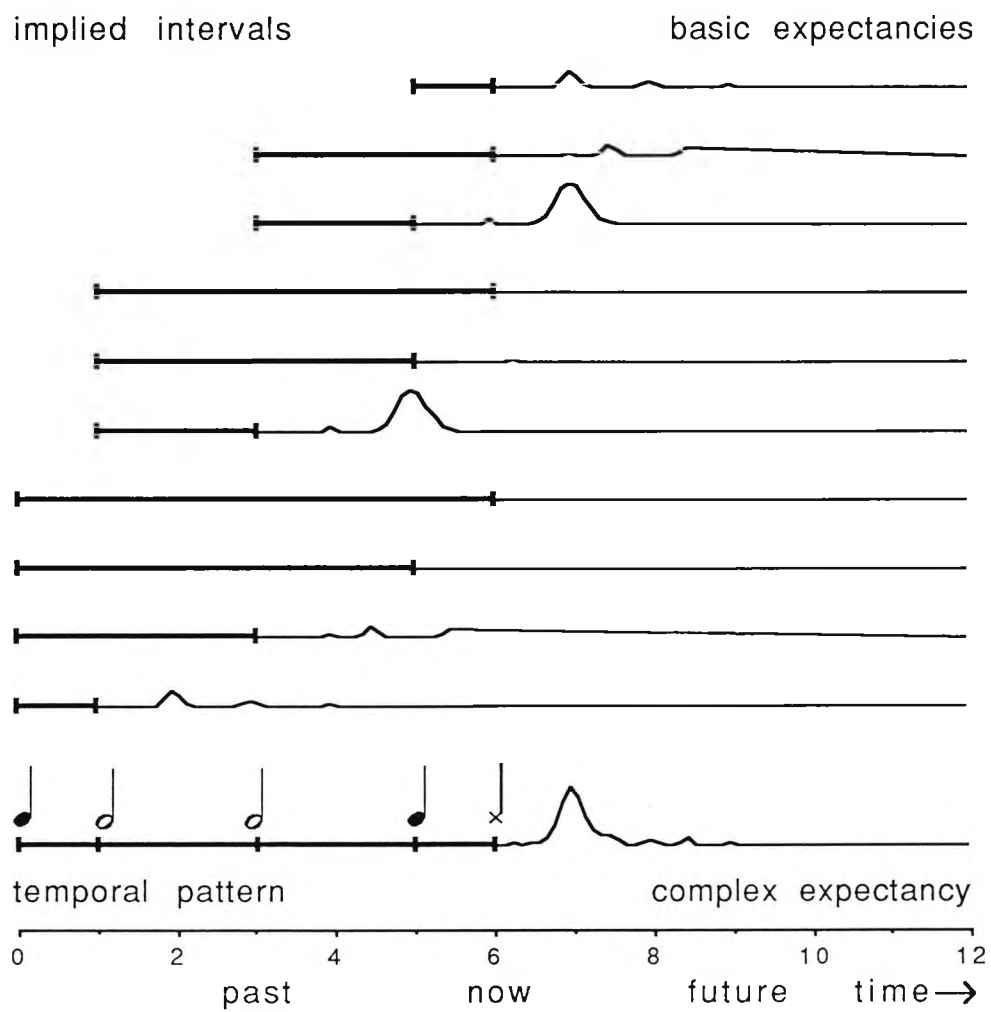


Figure 5. Expectancy at a slow tempo.

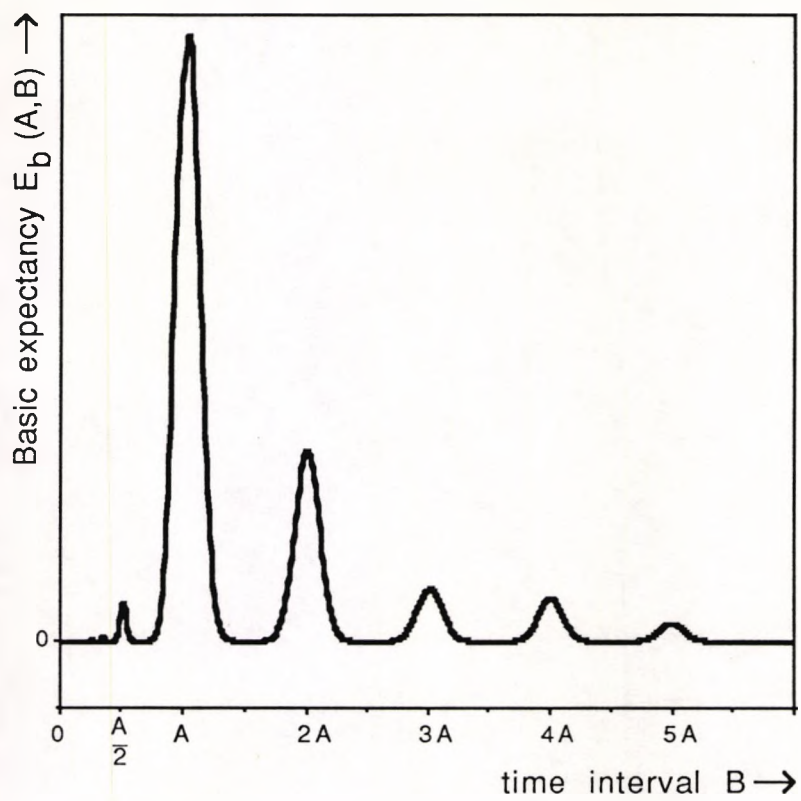


Figure 6. Basic expectancy of interval pair  $A, B$

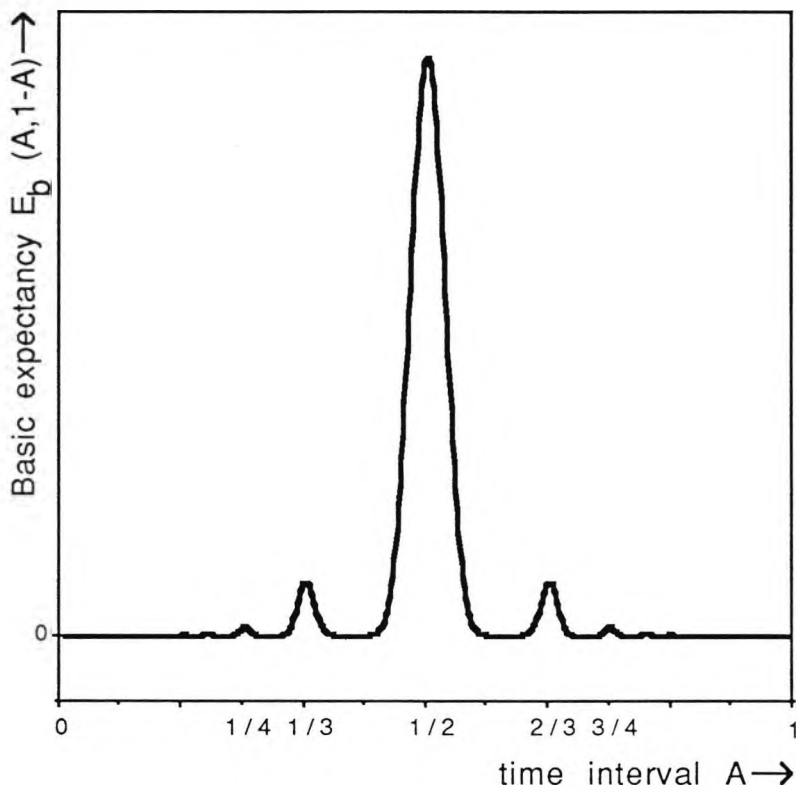


Figure 7. Basic expectancy of interval pair  $A, 1-A$

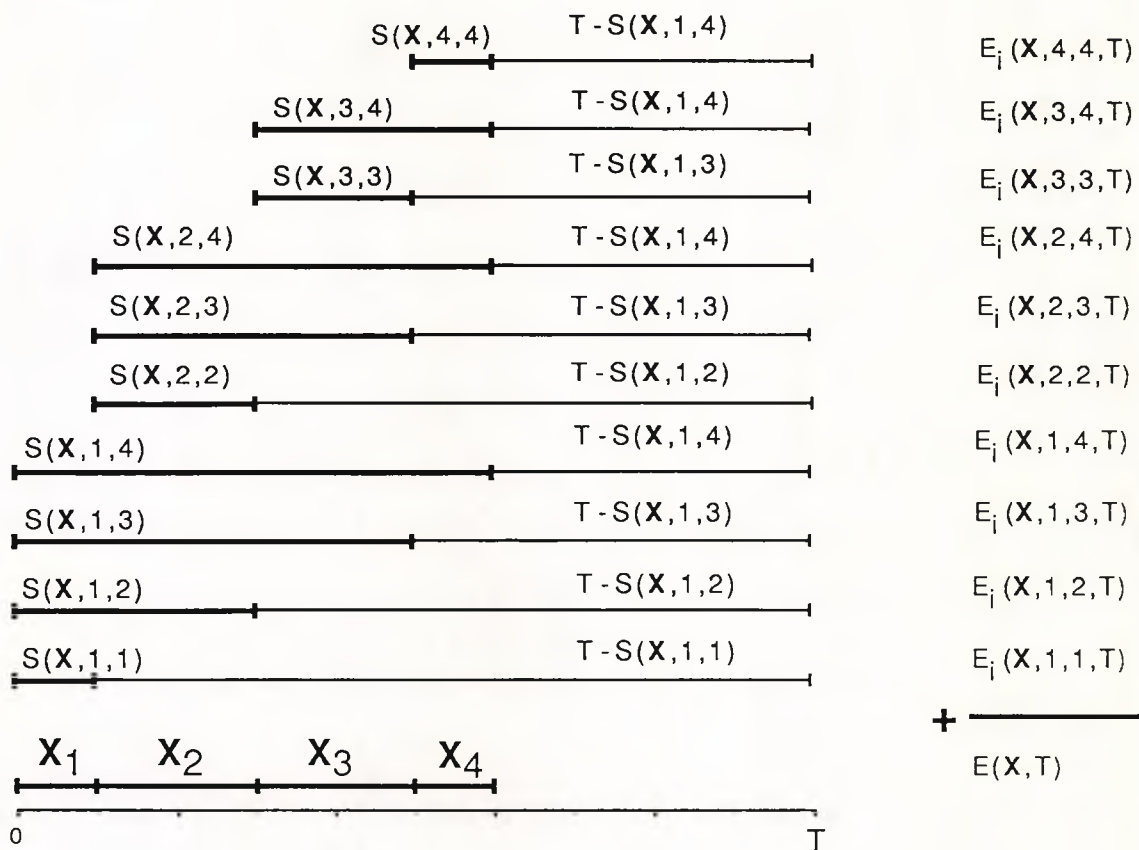


Figure 8. Time intervals used in calculating expectancy.



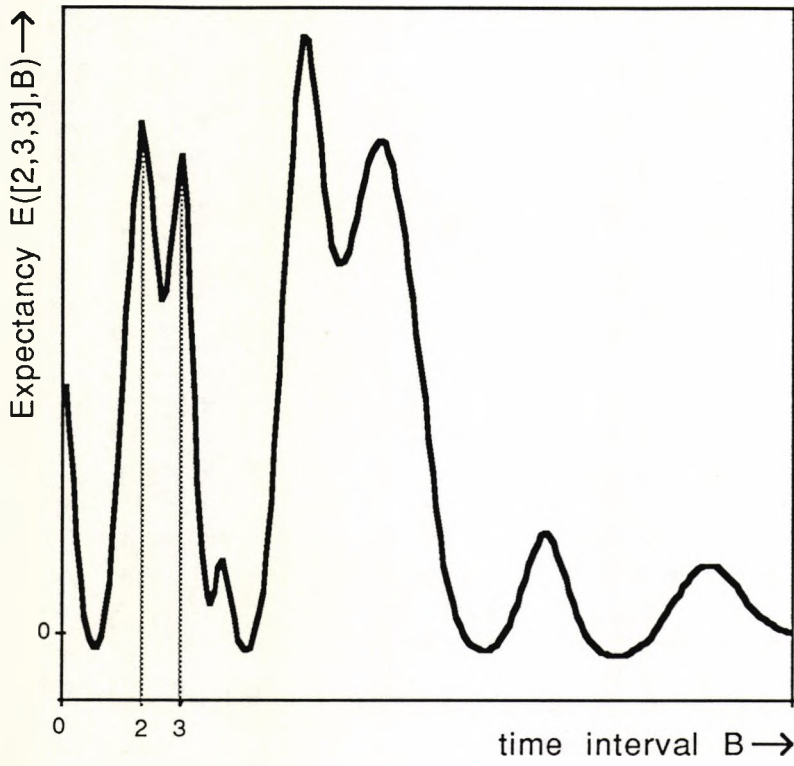


Figure 9. Facilitation of perception of the ratio  $2/3$  by context.

## LISP AS A SECOND LANGUAGE: FUNCTIONAL ASPECTS



PETER DESAIN

### MOTIVATION AS PREFACE

LISP, WHICH WAS designed as early as 1960 by McCarthy (McCarthy 1960) took a long time to be accepted by the computer music community as a suitable language for expressing their problems and solutions (Boynton et al. 1986, Kornfeld 1980), but has now become the preeminent language for symbolic programming in research fields such as Computational Linguistics and Artificial Intelligence. Computer music, and in particular composition, is another field which can benefit from symbolic computation. This poses the interesting question of why the advantages LISP offers have been neglected for so long. Perhaps it is because the mainstream of computer-music research has been sound generation, which doesn't involve many symbolic applications. In this field even present-day

technology is barely fast enough, so it is only natural to reject the extra layer of a high-level language in favor of more efficient low-level ones.

However, nowadays there is a renewed interest in computer-music composition, interactive composition systems, user interfaces for programming synthesizers and the use of AI techniques in computer music. Fundamental music research also uses algorithmic models and in all of these fields symbolic computation is pervasive and so the use of languages like FORTRAN or C is absurd. For the people entering the field of LISP programming from a background of these languages, the transition process will be painful. This is not the result of the many parentheses in LISP, but because it is difficult to come to grips with a whole new style of programming (thus a new way of thinking) and to unlearn the old stereotypical solutions. The change will often be made with a kind of "minimal effort" approach which involves using the old programming style in the new language. And indeed looking at the LISP programs emerging from the computer-music community, the *imperative style* can often be seen between the lines of LISP code. It is a pity to neglect the elegant ways of expressing algorithms in LISP, and doing so will often result in a disappointing performance and maintainability.

In this article I will try to make clear the functional aspects of LISP that cannot be found in the "old" languages. I hope this will result in more "LISPish" LISP programs and will give computer-music composers better techniques to express their personal constructs directly in the form of a working program.

### SOME REMARKS AS INTRODUCTION

This text comprises many examples. They were constructed for the sake of clarity, which means that they are not intended as a computer-music composition system. They rather show programming techniques that can be used in writing your own. The *dialect* chosen for the examples is Common LISP, but any LISP dialect with *lexical scoping* will do. Common LISP is likely to become a widely accepted standard although it yields less elegant programs compared to the purer LISP dialects like Scheme. The examples can be converted to Scheme by deleting the funcalls and function-quotes and adapting the function definition syntax. Readers not familiar with LISP may find the references to good introductory texts in the last paragraph useful. This article starts with a very condensed introduction to the basics of LISP in which the functions needed for the following sections are introduced as examples. Then the functional use of LISP is treated. More advanced topics such as *continuations* and *coroutines* will be treated in a subsequent paper.

## LISTS AS THE MAIN DATA STRUCTURE

The primary *data type* of LISP is the list, which is notated as an open parenthesis followed by zero or more elements and a closing parenthesis. To give an example, the following list could be used as a representation for a MIDI note of two time-units' duration, middle C (key number 60) and full amplitude (velocity 1.0).

```
(note 2 60 1)
```

Consider the choice of this primitive musical object here as an arbitrary one: we need just one such object in our examples. Of course the particular choice in a real program should be made on esthetic grounds—it expresses your idea of the atomic musical object and its parameters.

There is one special list, the empty list, notated as `()` or `nil`. The elements of a list can be symbols, called *atoms* (like `note` or `60`), or they can be lists. Thus a collection of notes can be represented as a list of lists.

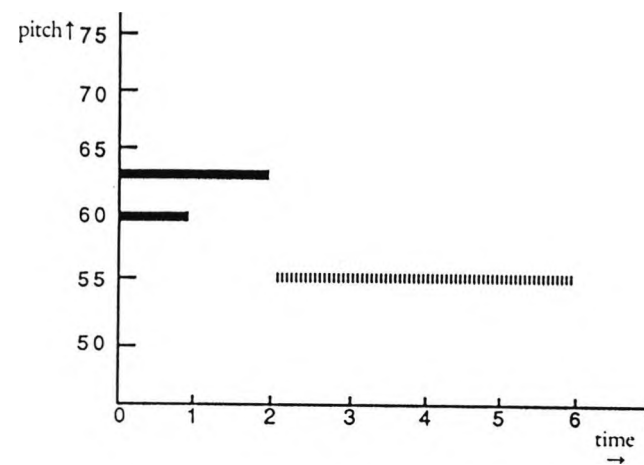
```
((note 1 60 1)(note 2 61 .7)(note 1 55 1))
```

If we wish to express control over the timing structure in the representation, ordering a collection of notes, we could form sequential and parallel structures. This way of specifying the time relations between objects is an alternative to the use of absolute start times found in most composition systems.

```
(sequential (note 1 60 1)(note 2 61 .7))
```

```
(sequential (parallel (note 1 60 1)
                      (note 2 63 1))
            (note 4 55 .7))
```

For a discussion of these time relations see Desain and Honing 1988. A piano-roll notation of the second musical object is given in Example 1. Note that the words like `note`, `sequential`, and `parallel` do not have any intrinsic meaning here, since they are not built-in LISP functions. They are just used as arbitrary symbols, signalling our intention with the data in that list. In Common LISP there are of course other data types available, such as strings, arrays, hash tables, and so on, which sometimes are more appropriate than lists. For the sake of clarity I will not use them here.



EXAMPLE 1: PIANO-ROLL NOTATION OF THE EXAMPLE SCORE

## ABSTRACTION AND APPLICATION AS DUAL MECHANISMS

The very heart of any functional programming language consists of a pair of dual constructs, the first of which is called *application*. It is the action of “calling” a *function* on one or more arguments. In the example below the function `first` is applied to a representation of a note (the argument). The syntactical form of an application is the name of the function followed by its arguments, together enclosed by parentheses (*prefix notation*).

```
(first '(note 1 60 1)) → note
```

In the example the arrow ( $\rightarrow$ ) points to the result of the evaluation of an expression. A constant data list is preceded by a *quote* (`'`) to distinguish it from an application. In the example above it prevented the LISP interpreter from recognizing `(note 1 60 1)` as an application of a function `note` to the arguments `1`, `60` and `1`.

There are two *selector functions* that take lists apart: `first` and `rest`. (Their former names, `car` and `cdr`, are considered obsolete.) There is one *constructor function* for building lists, called `cons`.

```
(rest '(note 1 60 1)) → (1 60 1)
```

```
(first (rest '(note 1 60 1))) → 1
```

```
(cons 'note '(1 60 1)) → (note 1 60 1)
```

Other functions for the construction of lists are also supplied as LISP primitives (e.g. `append`, which concatenates several lists, and `list`, which builds a list out of several elements at once), but they could be written by the programmer using only `cons`.

```
(append '(note 1) '(60 1)) → (note 1 60 1)
```

```
(list 'note 1 60 1) → (note 1 60 1)
```

```
(cons 'note (cons 1 (cons 60 (cons 1 nil)))) →
```

```
(note 1 60 1)
```

The second central construct is called *functional abstraction* and transforms an expression (a piece of program text) into a function. Our first function definition will define (by means of `defun`) a *constant function* without arguments called `example` that will just return, as a result, a simple constant musical structure. We will use this function a lot in the following paragraphs.

```
(defun example ())
```

```
  '(sequential (parallel (note 1 60 1)
                        (note 2 63 1))
              (note 4 55 .7)))
```

```
(example) →
```

```
(sequential (parallel (note 1 60 1)
                    (note 2 63 1))
            (note 4 55 .7))
```

Moving on, we will define functions that have one argument and select a specific element out of a list.

```
(defun second-element (list)
```

```
  (first (rest list)))
```

```
(defun third-element (list)
```

```
  (first (rest (rest list))))
```

```
(defun fourth-element (list)
  (first (rest (rest (rest list)))))
```

```
(second-element '(c d e f g a b)) → d
```

```
(third-element '(c d e f g a b)) → e
```

```
(fourth-element '(c d e f g a b)) → f
```

Now we can introduce some selector functions that take a note representation apart and a constructor function that builds one.

```
(defun duration-of-note (note)
```

```
  (second-element note))
```

```
(defun pitch-of-note (note)
```

```
  (third-element note))
```

```
(defun velocity-of-note (note)
```

```
  (fourth-element note))
```

```
(defun make-note (duration pitch velocity)
```

```
  (list 'note duration pitch velocity))
```

```
(duration-of-note '(note 3 60 1)) → 3
```

```
(pitch-of-note '(note 3 60 1)) → 60
```

```
(make-note 3 60 1) → (note 23 60 1)
```

In the first example above a function called “duration” is defined which has one parameter called `note`. Its body is the application of the function `second-element` on this parameter. In the definition the parameter `note` is said to be abstracted from the body. Only when `duration-of-note` is applied to an actual argument does the body becomes “concrete” in the sense that it “knows” what the parameter `note` stands for, so that its value can be calculated. In the same way we could program a set of selector and constructor functions as a *data abstraction* layer for sequential and parallel structures.

```
(defun make-sequential (structure-1 structure-2)
```

```
  (list 'sequential structure-1 structure-2))
```



```
(defun make-parallel (structure-1 structure-2)
  (list 'parallel structure-1 structure-2))

(defun structure-1-of (complex-structure)
  (second-element complex-structure))

(defun structure-2-of (complex-structure)
  (third-element complex-structure))

(defun structural-type-of (complex-structure)
  (first complex-structure))

(structural-type-of (example)) → sequential
(structure-2-of (example)) → (note 4 55 .7)
```

The use of complex data structures (sequential and parallel) of two components does not make them less general in use because they can always be nested:

```
(sequential (sequential (note 1 60 1)
                        (note 2 61 1))
            (note 1 63 1))
```

However, Common LISP provides means of control for passing arguments to functions and we could use the so called *lambda-list keyword* &rest to signal LISP to collect all the arguments of the function in a list.

```
(defun make-sequential (&rest elements)
  (cons 'sequential elements))

(defun make-parallel (&rest elements)
  (cons 'parallel elements))

(make-sequential (make-note 1 60 1)
                 (make-note 1 62 1)
                 (make-note 1 63 1)) →

(sequential (note 1 60 1)(note 1 62 1)(note 1 63 1))
```

Also, optional parameters to a function, that will be assigned a default value when missing in the function call, can be defined by means of the &optional lambda-list keyword.

```
(defun make-note (&optional (duration 1)(pitch 60)(loudness 1))
  (list 'note duration pitch loudness))

(make-note 2 61) → (note 2 61 1)
(make-note) → (note 1 60 1)
```

For the sake of clarity we will use only the make-sequential and make-parallel functions which have two arguments and the make-note function which has three arguments. Using the data abstraction layer provided by the selector and constructor functions for notes to implement transformations on notes produces very readable LISP code.

```
(defun transpose-pitch (pitch interval)
  (+ pitch interval))

(defun mirror-pitch (pitch center)
  (- center (- pitch center)))

(defun transpose-note (note interval)
  (make-note (duration-of-note note)
             (transpose-pitch (pitch-of-note note)
                              interval)
             (velocity-of-note note)))

(defun mirror-note (note center)
  (make-note (duration-of-note note)
             (mirror-pitch (pitch-of-note note) center)
             (velocity-of-note note)))

(defun transpose-note-semitone (note)
  (transpose-note note 1))

(defun mirror-note-around-middle-c (note)
  (mirror-note note 60))
```

```
(transpose-note-semitone '(note 1 60 1)) →
(note 1 61 1)

(mirror-note-around-middle-c '(note 1 57 1)) →
(note 1 63 1)
```

Note that we first defined some general note-transforming functions and then used these in turn to define the dedicated ones required in the following chapter. The utility functions for the pitch arithmetic isolate the calculation of pitches from the note-transforming functions.

Application should be used as the only means to pass information to a function. This ensures the behavior of functions is not dependent upon the context of its call. This obviates the use of *global variables*. They are to be used only in cases where they represent truly constant values.

#### RECURSION AS MAIN CONTROL STRUCTURE

From the beginning the use of *recursion* in LISP programs was pervasive. Consider the transposition of a collection, i.e. a list, of notes. When the list is empty the task is simple: the empty list has to be returned. Otherwise we cons the transposition of the first note in the result of transposing the rest of the list. The function required for transposing this smaller list is precisely the function we are writing at this moment, so it only has to call itself. This process of self-reference is called recursion.

```
(defun transpose-note-list-semitone (notes)
  (when notes
    (cons (transpose-note-semitone (first notes))
          (transpose-note-list-semitone (rest notes))))))

(transpose-note-list-semitone
 '( (note 1 60 1) (note 2 59 .7) (note 1 65 .7) )) →
((note 1 61 1) (note 2 60 .7) (note 1 66 .7))
```

This simple form of recursion (called *tail recursion*) is recognized by any reasonable compiler, and internally transformed into plain *iteration*, thus overcoming the overhead usually associated with recursion, i.e. extra function calls and increased *stack* space. In LISP the empty list: () and the truth value false are defined equivalent and called nil. Conversely, everything

that is not nil is considered as true: t. This is used here in two ways. Firstly, the condition in the when clause will only be considered true when the note list is not empty, when there are notes left. Secondly, the result of the when clause when the condition is false will be nil, and this will function as the empty starting list for *consing* in transposed notes.

While newly designed languages accepted recursion as a control structure, LISP was augmented with "down-to-earth" and well-known iterative control structures, since it was recognized that in some cases these are simpler for humans to use than recursion. For complex cases however, the recursive form is often the more elegant and easier to read. For example, if we wish to define a transposition on a complex musical structure (built from parallel and sequential) we must first dispatch on the type of structure (using a case construct) and then apply the transformation recursively on the component structures, and finally reassemble the transposed parts into their parallel or sequential order. The resultant program would look very messy when written iteratively.

```
(defun transpose-semitone (structure)
  (case (structural-type-of structure)
    (note      (transpose-note-semitone structure))
    (sequential (make-sequential
                  (transpose-semitone (structure-1-of structure))
                  (transpose-semitone (structure-2-of structure))))))
    (parallel  (make-parallel
                (transpose-semitone (structure-1-of structure))
                (transpose-semitone (structure-2-of structure))))))

(transpose-semitone (example)) →
(sequential (parallel (note 1 61 1)
                     (note 2 64 1))
            (note 4 56 .7)))
```

case will evaluate its first argument, then select a subsequent clause starting with that value, followed by an evaluation of the second part of that clause. In general it can be said that recursion is the natural control structure for hierarchical data. And hierarchical structures are common in music.

## FUNCTIONS AS FIRST-CLASS OBJECTS

In any good programming language all possible objects are allowed to appear in all possible constructs: they are all *first-class citizens*. However, in many programming languages this rule is often violated. In LISP even exotic objects such as functions can be passed as an argument to a function (in an application construct) or yielded as a result from a function. At first sight this may not seem unusual. PASCAL, for example, allows the name of a procedure to be passed to another one using an ad hoc construction. And in C, pointers to functions can be passed around. However, in LISP all functions are uniformly considered as data objects in their own right, and functions operating on them can be used. This provides an abstraction level that is really a necessity but that is lacking in many other languages. For composers of computer music it is quite natural to think in terms of abstract transformations on objects like time-mappings, which are functions themselves.

## FUNCTIONS AS ARGUMENTS

Suppose we want to write a function *mirror-around-middle-c* which would look similar to *transpose-semitone* defined above but only uses *mirror-note-around-middle-c* instead of *transpose-note-semitone* as the bottom level transformation. Instead of just writing a new function for that purpose, it is better to abstract from the bottom transformation and write a general transform function. This function is now given the note transformation as an extra *functional argument*, which enables it to deal with all kinds of note transformations. Wherever it needs the result of the application of the note transformation function to a specific note, it calculates that with the LISP *funcall* construct.

```
(defun transform (structure transform-note)
  (case (structural-type-of structure)
    (note      (funcall transform-note structure))
    (sequential (make-sequential
                  (transform (structure-1-of structure)
                             transform-note)
                  (transform (structure-2-of structure)
                             transform-note))))
    (parallel  (make-parallel
                (transform (structure-1-of structure)
                           transform-note)
                (transform (structure-2-of structure)
                           transform-note))))))

(transform '(sequential (note 1 60 1)(note 2 63 1))
          #'transpose-note-semitone) →
(sequential (note 1 61 1)(note 2 64 1))

(transform '(sequential (note 1 60 1)(note 2 63 1))
          #'mirror-note-around-middle-c) →
(sequential (note 1 60 1)(note 2 57 1))
```

Note the use of the *#'* construct (called the *function quote*), which is used to signal to LISP that the following expression is to be considered as a function.

The possibilities of the chosen representation of musical objects and transformations on it are illustrated by the following example. Here each note is transformed into a sequence of two notes with half the original duration. This transformation, called *double-note*, is built from two other transformations. The first one, *half-note*, divides the duration of its argument by two. The second, *twice-note*, makes a sequence of two identical copies of its argument.

```
(defun twice-note (note)
  (make-sequential note note))
```

```
(defun half-note (note)
  (make-note (/ (duration-of-note note) 2.0)
             (pitch-of-note note)
             (velocity-of-note note)))

(defun double-note (note)
  (twice-note (half-note note)))

(transform (example) #'double-note)→
(sequential
 (parallel (sequential (note .5 60 1)(note .5 60 1))
           (sequential (note 1 63 1)(note 1 63 1)))
 (sequential (note 2 55 .7)(note 2 55 .7)))
```

The use of functions as arguments (so-called *downward funargs*) seems to give so much extra power that we might begin to wonder what good the passing of functions as results (so-called *upward funargs*) could give us.

#### FUNCTIONS AS RESULTS

If we wanted to apply an octave transposition to a structure we would have to write a new function, `transpose-note-octave`, and use it as an argument for `transform`.

```
(defun transpose-note-octave (note)
  (transpose-note note 12))

(transform (example) #'transpose-note-octave)→
(sequential (parallel (note 1 72 1)
                    (note 2 75 1))
           (note 4 67 .7)))
```

This means we always have to define the possible transformations in advance. This is not satisfactory and instead we could use *anonymous* functions as an argument to the `transform` function. Anonymous functions are not as bad as they look. They are merely a consequence of the rule of first class citizens. For example, it is perfectly normal for objects like numbers, lists, and strings to have a notation for the constant values. Functions should also have this property. The anonymous function of one argument that will transpose a note by an octave is notated like this:

```
(lambda (note)
  (transpose-note note 12))
```

This kind of function can be used as argument to the `transpose` function (remember the function-quote).

```
(transform (example)
          #'(lambda (note)
              (transpose-note note 12)))→
(sequential (parallel (note 1 72 1)
                    (note 2 75 1))
           (note 4 67 .7))

(transform (example)
          #'(lambda (note)
              (mirror-note note 72)))→
(sequential (parallel (note 1 84 1)
                    (note 2 81 1))
           (note 4 89 .7)))
```

Still, this is a little tedious to do, and we define a function `transpose-note-transform` that will calculate these transposition functions when given the correct number of semitones.

```
(defun transpose-note-transform (interval)
  #'(lambda (note) (transpose-note note interval)))

(defun mirror-note-transform (center)
  #'(lambda (note) (mirror-note note center)))

(transform (example) (transpose-note-transform 2))→
(sequential (parallel (note 1 62 1)
                    (note 2 65 1))
           (note 4 57 .7)))
```



```
(transform (example) (mirror-note-transform 67)) →
(sequential (parallel (note 1 74 1)
                    (note 2 71 1))
            (note 4 79 .7))
```

#### GENERALITY AS AIM

Functional programming makes it possible to construct very general programs that are customized for specific purposes. These are the tools that are badly needed in software design. They deserve to be supplied in software libraries, so that programmers can stop reinventing the wheel each time. As a tool for composers these programs may aim to be as “empty” as possible, only expressing general abstract knowledge about musical structure and leaving open details about the specific material and relations used.

The transformations we have so far designed are not yet that general. They were *structure preserving*, and thus a transformation of a sequence would always yield a sequence. Only at the note level could the structure expand into a bigger one (c.g. when using double-note). Bearing this in mind, we are going to develop a more general transformation device. Our new transformation is like the old one, except that it takes two more arguments to calculate what a sequential and what a parallel structure transforms into.

```
(defun transform
  (structure sequential-transform parallel-transform note-transform)
  (case (structural-type-of structure)
    (note (funcall note-transform structure))
    (sequential (funcall sequential-transform
                        (transform (structure-1-of structure)
                                sequential-transform
                                parallel-transform
                                note-transform)
                        (transform (structure-2-of structure)
                                sequential-transform
                                parallel-transform
                                note-transform))))
```

```
(parallel (funcall parallel-transform
                  (transform (structure-1-of structure)
                          sequential-transform
                          parallel-transform
                          note-transform)
                  (transform (structure-2-of structure)
                          sequential-transform
                          parallel-transform
                          note-transform))))
```

After this rather tedious definition we have available a very powerful transformational device. Let me first give a rather stupid example of its use: the no-transform transformation function which just rebuilds its argument.

```
(defun no-note-transform (note) note)

(defun no-transform (structure)
  (transform structure
            #'make-sequential
            #'make-parallel
            #'no-note-transform))

(no-transform (example)) →
(sequential (parallel (note 1 60 1)
                    (note 2 63 1))
            (note 4 55 .7))
```

The results passed upward by no-note-transform, make-parallel, and make-sequential are musical structures (see Example 2). Using the same idea, but substituting the identity note transformation by another one, gives us the second example. It supports all structure-preserving transformations.

```
(defun our-old-transform (structure transform-note)
  (transform structure
    #'make-sequential
    #'make-parallel
    transform-note))
```

Now some more useful transformations can be constructed. The results passed upward will be numbers. The first transformation calculates the duration of a complex musical structure by adding or maximizing the duration of substructures. Similarly it is possible to calculate the duration of the longest note, the number of notes in a piece, and the maximum number of parallel voices of a complex structure. Note how easily the transform function is adapted to these different purposes by "plugging in" different functional arguments (see Example 2).

```
(defun duration-of (structure)
  (transform structure #'+ #'max #'duration-of-note))

(defun longest-note-duration (structure)
  (transform structure #'max #'max #'duration-of-note))

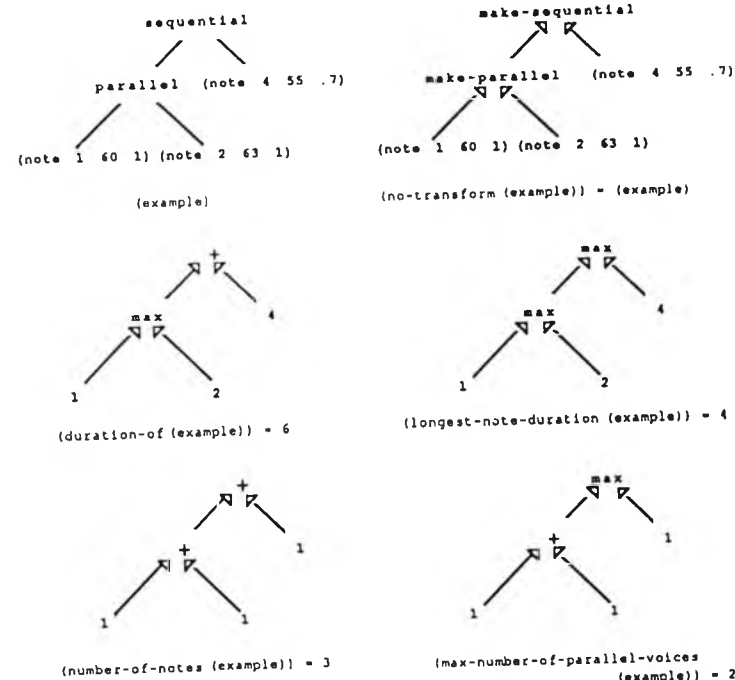
(defun count-one (note) 1)

(defun number-of-notes (structure)
  (transform structure #'+ #'+ #'count-one))

(defun max-number-of-parallel-voices (structure)
  (transform structure #'max #'+ #'count-one))

(duration-of (example)) → 6
(longest-note-duration (example)) → 4
(number-of-notes (example)) → 3
(max-number-of-parallel-voices (example)) → 2
```

To demonstrate again the generality of the transform, we will now write a program to draw a piano-roll notation of a musical structure as shown in Example 1. This program was inspired by a tree-drawing program of Joop Ringelberg. To draw a note at the correct place we need to know the



EXAMPLE 2: TRANSFORMATIONS OF THE EXAMPLE STRUCTURE

absolute start time of a musical object, information that the transform function itself does not supply. When context information is missing, it is a well-known trick in AI programming to calculate a function of the (not yet known) context as temporary result. Indeed such a solution is possible here. The draw-note function can return a function that will draw a graphical representation of a note when given a start time. As the drawing is done as a *side effect*, this function can then return the end time of the note as context to use in further drawing. The draw-sequential function just receives two such draw functions as arguments and constructs the draw function that will pass its start time to the first and pass the end time returned by the first to the second, returning its end time as the result. The function draw-parallel will pass its start time to both substructure draw functions returning the maximum end time they return. Thus neither numbers nor musical structures are passed upward as result of the transformation on substructures, but functions that can draw the substructure

when given a start time. At the top level we will just have to apply the draw function resulting from the call to transform to time 0. An alternative draw-note function is given for a quick test of the code without implementing the graphic procedure draw-horizontal-block.

```
(defun draw (structure)
  (funcall (transform structure
    #'draw-sequential
    #'draw-parallel
    #'draw-note)
    0))

(defun draw-note (note)
  #'(lambda (time)
    (draw-horizontal-block      ; assumed graphical primitive
      time                    ; left x-position
      (pitch-of-note note)     ; y-position
      (duration-of-note note)  ; width
      (velocity-of-note note)) ; grey shade
      (+ time (duration-of-note note)))) ; end time

(defun draw-note (note)          ; alternative for testing without
                                ; graphics
  #'(lambda (time)
    (print (list 'time time
      'pitch (pitch-of-note note)
      'duration (duration-of-note note)
      'velocity (velocity-of-note note))
      (+ time (duration-of-note note))))

(defun draw-sequential (a b)
  #'(lambda (time)
    (funcall b (funcall a time))))

(defun draw-parallel (a b)
  #'(lambda (time)
    (max (funcall b time)
      (funcall a time))))

(draw (example)) → 'example 1'
```

Having shown this general solution for dealing with context information, it is clear that this will not be always the best solution. When information

like start time is used a lot, it may be simpler to adapt the transform function itself so that it passes this information as well to the note transformation function:

```
(defun time-transform
  (structure
    sequential-transform parallel-transform note-transform
    &optional (time 0))
  (case (structural-type-of structure)
    (note (funcall note-transform structure time))
    (sequential
      (funcall sequential-transform
        (time-transform (structure-1-of structure)
          sequential-transform
          parallel-transform
          note-transform
          time)
        (time-transform (structure-2-of structure)
          sequential-transform
          parallel-transform
          note-transform
          (+ time
            (duration-of
              (structure-1-of structure))))))
      (parallel
        (funcall parallel-transform
          (time-transform (structure-1-of structure)
            sequential-transform
            parallel-transform
            note-transform
            time)
          (time-transform (structure-2-of structure)
            sequential-transform
            parallel-transform
            note-transform
            time))))))
```

The time is made into an optional parameter to facilitate the omission of the start time zero in the call of time-transform at the top level. We now can build transformations, such as a fade-out (decrescendo), that are time dependent.

```
(defun fade-out-transform (begin end)
  #'(lambda (note time)
    (if (< begin time end)
        (make-note (duration-of-note note)
                   (pitch-of-note note)
                   (* (velocity-of-note note)
                     (- 1.0 (/ (- time begin)
                               (- end begin))))))
      note)))

(time-transform (make-sequential (example) (example))
  #'make-sequential
  #'make-parallel
  (fade-out-transform 0 10))→
(sequential (sequential (parallel (note 1 60 1)(note 2 63 1))
                                (note 4 55 .56))
            (sequential (parallel (note 1 60 .4)(note 2 63 .4))
                        (note 4 55 .14)))
```

Sometimes we wish to transform our musical objects to note lists where each note has as an extra first parameter an absolute start time, e.g. to play them using a system like John Rahn's LISP Kernel (Rahn 1988, 1990). We can do that now easily. All that is required is a function to transform a note to a list of one note in the new format, and a function that will merge two sorted parallel note lists. An alternative for this function using the Common LISP merge primitive is given as well.

```
(defun add-absolute-start-time (note time)
  (list (list 'note
             time
             (duration-of-note note)
             (pitch-of-note note)
             (velocity-of-note note))))
```

```
(defun merge-note-lists (list-1 list-2)
  (cond ((null list-1) list-2)
        ((null list-2) list-1)
        ((<= (second-element (first list-1))
              (second-element (first list-2)))
         (cons (first list-1)
               (merge-note-lists (rest list-1) list-2)))
        (t (cons (first list-2)
                  (merge-note-lists list-1 (rest list-2))))))

(defun merge-note-lists (list-1 list-2) ; alternative
  (merge 'list list-1 list-2 #'< :key #'second-element))

(defun musical-object-to-note-list (object)
  (time-transform object
    #'append
    #'merge-note-lists
    #'add-absolute-start-time))

(musical-object-to-note-list (example))→
'((note 0 1 60 1)(note 0 2 63 1)(note 2 4 55 .7))
```

#### COMBINATORS AS FUNCTION BUILDERS

Since it turned out to be so useful to be able to talk about functions as objects which are passed to and from other functions, we are now going to examine the possibilities of a special kind of these "other" functions, called *combinators*. A combinator is a *higher order function* that has only functions as arguments and returns a function as a result. The first one we will show is the combinator called *twice*. It can double the action of any function, and therefore (*twice #'rest*) will be a function that removes the first two elements from a list, (*twice #'double-note*) will yield a four-fold splitting of notes, and (*twice #'mirror-note-around-middle-c*) will be an absolutely useless transformation (the identical transformation).

```
(funcall (twice #'rest) '(1 2 3 4 5))→ (3 4 5)
```



```
(our-old-transform (example) (twice #'transpose-note-octave)) →
(sequential (parallel (note 1 84 1)
                      (note 2 87 1))
            (note 4 79 .7)))
```

Here is the definition of twice.

```
(defun twice (transform)
#' (lambda (object) (funcall transform
                             (funcall transform object))))
```

The second combinator is the “function-composition” combinator. It can combine the actions of two transformations into a new one. It is important not to confuse *function composition* and musical composition.

```
(defun compose (transform-1 transform-2)
#' (lambda (object)
    (funcall transform-1 (funcall transform-2 object))))
```

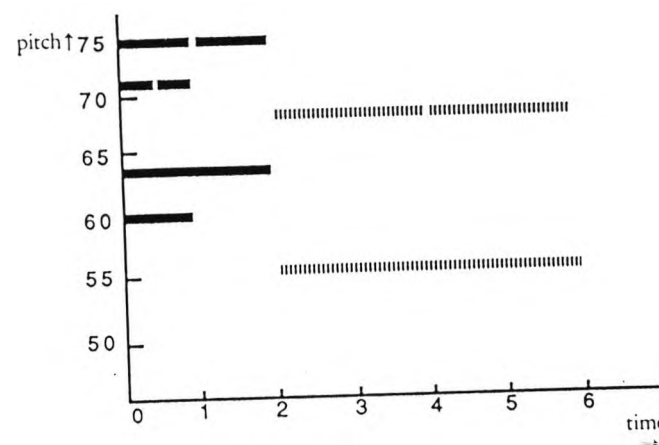
```
(funcall (compose #'first #'rest) '(c d e f'g a b)) → d
```

To construct a transformation that is a doubling applied to the result of an octave transposition we could use this combinator to build it.

```
(our-old-transform
 (example)
 (compose #'double-note #'transpose-note-octave)) →
(sequential
 (parallel (sequential (note .5 72 1)(note .5 72 1))
           (sequential (note 1 75 1)(note 1 75 1)))
 (sequential (note 2 67 .7)(note 2 67 .7)))
```

To show the usefulness of these constructions we will write a function that calculates a complex melody from a simple one by adding a parallel melody that is the doubling of the original one transposed one octave. This is a transformation often used in Javanese Gamelan music. The score of the add-doubled transformation on the example object is shown in Example 3.

```
(defun add-doubled (structure)
 (make-parallel
  structure
  (our-old-transform structure
   (compose #'double-note
            #'transpose-note-octave))))
```



EXAMPLE 3: RESULT OF THE ADD-DOUBLED TRANSFORMATION

Note that we could have defined twice as a composition of a transform with itself.

```
(defun twice (transform)
 (compose transform transform))
```

#### PARAMETERS AS SUPERFLUOUS

When defun defines a function, it creates an anonymous function using the argument list and the body, and stores it in the function cell of the symbol that is the name of the function. This means that the next two expressions are equivalent.

```
(defun duration-of-note (note)
  (second-element note))

(setf
  (symbol-function 'duration-of-note); assign to the function slot
  #'(lambda(note)                    ; of the symbol
      (second-element note)))      ; duration-of-note
                                   ; this anonymous function
```

defun can be considered as a device that makes the definition of a function easier to read, but it assumes the name of the function and the function body itself to be constants. Let us first make a similar construct that gives a little bit more power than defun, and then use it to define duration-of again.

```
(defun define-function (name function)
  (setf (symbol-function name) function))

(define-function 'duration-of-note
  #'(lambda(note)
      (second-element note)))
```

Now we can calculate a function body instead of using a constant anonymous function. In the first example below the calculation merely finds the function definition of a symbol, to create a synonymous function. In the second it is really calculated by one of the transformations we defined above.

```
(define-function 'premier (symbol-function 'first))



```

Note that in the definitions above, the formal parameters which appeared in the argument list of the defun form are no longer needed. If we have access to enough combinators like twice and compose, we can even do completely without parameters.

```
(define-function 'double-note (compose #'twice-note #'half-note))

(define-function 'double-and-raise-note
  (compose #'double-note #'transpose-note-octave))

(define-function 'four-times (twice #'twice))

(define-function 'but-first-four (four-times #'rest))

(but-first-four '(c d e f g a b)) → (g a b)
```

Note how twice is applied to itself to yield the four-times function.

Languages built on combinators using *parameter-free programming* are very useful in domains centered around one type of object and many transformations on this object (like the musical structures in our examples). In these domains they facilitate the definition of higher levels of abstraction whereby transformations are considered objects in their own right, so that they can be manipulated, combined, and modified. However, when dealing with functions of many arguments we need a lot of combinators for juggling with the order of arguments, leading to programs that are difficult to read. For humans, an extra hook into our memory, by means of a mnemonic parameter name, is often indispensable. In addition, more heterogeneous domains consisting of different sorts of objects, all subjected to transformations that are conceptually more or less the same, can be modelled better using another style of programming in LISP. In this style, named *object-oriented programming*, it is straightforward to express, e.g., how both a melody and a synthesizer can have their own definition of “transposition.”

#### SOME WORDS AS CONCLUSION

Exaggerating my standpoint, I will give a summary of a “good” style of LISP programming.

1. No function is longer than five lines.
2. Programs are written functionally.
3. No global variables are used.
4. Names of functions and parameters are long and descriptive.
5. Flow of control is done with recursion.

Such a clean and elegant style will result in programs that are easy to construct and maintain. One advantage that languages like FORTRAN and C still have when compared to LISP is their speed. Luckily good industrial Common LISP compilers make the difference quite small.

#### FRIENDS AS GUINEA PIGS

I am very grateful that the following friends volunteered to read and comment on the first draft of this paper: Paul Berg, Edwin Bos, Wim Claassen, Jim Grant, Margriet Hoenderdos, Koen De Smedt, John Rahn, Joop Ringelberg, Dick Rijken, Huub van Thienen and Theo Vosse, and especially Henkjan Honing because of his stimulating attitude and good ideas.

I would like to thank the colleagues and students of the Language Technology group of the University of Nijmegen and the Centre for Art and Media Technology in Utrecht and for the stimulating environment they provided while I was working on this article.

#### LISP AS A BOOK TOPIC

Since there are so many books on LISP it is difficult to select the one that will be of most use. A quick rule of thumb that can be used is: if a book on LISP starts by explaining SETQ (or SETF) it is unsuitable.

Abelson & Sussman 1985 is an almost perfect introduction to all the wonders of LISP programming and programming in general. There are extensive examples, but it is a pity they are all of the numerical and engineering type. It uses the Scheme dialect. Anderson et al. 1987 is a introductory textbook with lots of exercises. Friedman & Felleisen 1986 is a very funny (and good) programmed instruction course. If you want to learn to write recursive list programs in LISP without a teacher and without prerequisites, try this. Watch out for the differences between editions. There is an MIT Press version as well. Henderson 1980 is a good and clear introduction to functional programming, with many examples. Steele 1984 is the defining report on Common LISP, not intended for learning LISP but indispensable as a reference. If you have the opportunity, take a look into the Symbolics manuals to gain a feeling for the size and power of the programming environment of a LISP machine. Winston and Horn 1981 is a general introduction to LISP.

#### GLOSSARY

- ATOM. In LISP: any symbol, number or other non-list.
- ACCESS FUNCTION. A function which is part of a data abstraction layer (a selector or constructor function).
- ANONYMOUS FUNCTION. A function whose "pure" definition is given, not assigning it a name at the same time.
- APPLICATION. Obtaining a result by supplying a function with suitable arguments.
- COMBINATOR. A function that has only functions as arguments and returns a function as result.
- CONS. A LISP primitive that builds lists. Sometimes used as verb: to add an element to the beginning of a list.
- CONSTANT FUNCTION. A function that always returns the same value.
- CONSTRUCTOR FUNCTION. A function that, as part of the data-abstraction layer, provides a way of building a data structure from its components.
- CONTINUATION. A way of specifying what a function should do with its result.
- COROUTINES. Parts of the program that run in alternation, but remember their own state of computation in between switches.
- DATA ABSTRACTION. A way of restricting access and hiding detail of data structures.
- DATA TYPE. A class of similar data objects, together with their access functions.
- DIALECT. A programming language can be split up into dialects that only differ (one hopes) in minor details. LISP dialects are abundant and may differ a lot from each other even in essential constructs.
- FIRST-CLASS CITIZENS. Rule by which any type of object is allowed in any type of programming construct.
- FUNCTION. A program or procedure that has no side effects.
- FUNCTION COMPOSITION. The process of applying one function after another.
- FUNCTIONAL ABSTRACTION (OR PROCEDURAL ABSTRACTION). A way of making a piece of code more general by turning part of it into a

parameter, creating a function that can be called with a variety of values for this parameter.

FUNCTIONAL ARGUMENT (FUNARG). A function that is passed as argument to another one (downward funarg) or returned as result from other one (upward funarg).

FUNCTION QUOTE. A construct to capture the correct intended meaning (with respect to the current lexical environment) of an anonymous function so it can be applied later in another environment; a lexical closure. It is considered good programming style to use function quotes as well when quoting just the name of a function.

GLOBAL VARIABLES. Objects that can be referred to (inspected, changed) from any part of the program.

HIGHER-ORDER FUNCTION. A function that has functions as arguments.

IMPERATIVE STYLE. A programming style in which assignment and iteration are the main constructs.

ITERATION. Repeating a certain segment of the program.

LAMBDA-LIST KEYWORD. A keyword that may occur in the list of parameter names in a function definition. It signals how this function expects its parameters to be passed, whether they may be omitted in the call, and so forth.

LEXICAL SCOPING. A rule that limits the "visibility" of a variable to a textual chunk of the program. Much confusion can result from the older—so-called dynamic scoping—rules.

OBJECT-ORIENTED PROGRAMMING. A style of programming whereby each data type is grouped with its own access function definitions, possibly inheriting them from other types.

PARAMETER-FREE PROGRAMMING. A style of programming whereby only combinators are used to build complex functions from simple ones.

PREFIX NOTATION. A way of notating function application by prefixing the arguments with the function.

QUOTE. A construct to prevent the LISP interpreter from evaluating an expression.

RECURSION. A method by which a function is allowed to use its own definition.

SELECTOR FUNCTION. A function that as part of the data abstraction layer provides access to a data structure by returning part of it.

SIDE EFFECT. Any actions of a program that may change the environment and so change the behavior of other programs.

STACK. A list of function calls that are initiated but have not yet returned a value.

STRUCTURE PRESERVING. A way of modifying data that keeps the internal construction intact but may change attributes attached to the structure.

TAIL RECURSION. A type of recursion in which the recursive call is the "last" thing the program does.

#### REFERENCES

- Abelson, Harold, and Gerald Jay Sussman. 1985. *Structure and Interpretation of Computer Programs*. Cambridge, MA: The MIT Press.
- Anderson, J.R., A.T. Corbett, and B.J. Reiser. 1987. *Essential Lisp*. Reading MA: Addison-Wesley.
- Backus, John. 1978. "Can Programming Be Liberated from the von Neumann Style? A Functional Style and its Algebra of Programs." *Communications of the ACM* 21: 613-41.
- Boynton, L., P.Lavoie, Y. Orlarey, C. Rueda, and D. Wessel. 1986. "MIDI- Lisp, A Lisp-Based Music Programming Environment for the Macintosh." In *Proceedings of the 1986 International Computer Music Conference*, edited by Paul Berg. San Francisco: Computer Music Association.
- Desain, Peter, and Henkjan Honing. 1988. "LOCO: A Composition Microworld in Logo." *Computer Music Journal* 12, no. 3 (Fall): 30-42.
- Friedman, D., and M. Felleisen. 1986. *The Little LISPer*. 2nd edition. Chicago: Science Research Associates Inc.
- Henderson, Peter. 1980. *Functional Programming: Application and Implementation*. Englewood Cliffs, NJ: Prentice-Hall.
- Kornfeld, William A. 1980. "Machine Tongues VII: LISP." *Computer Music Journal* 4, no. 2 (Summer): 6-12.



McCarthy, John. 1960. "Recursive Functions of Symbolic Expressions and Their Computation by Machine." *Communications of the ACM* 3, no. 4: 184-195.

Rahn, John. 1988. "Computer Music: A View from Seattle." *Computer Music Journal* 12, no. 3 (Fall): 15-29.

———. 1990. "The LISP Kernel: A Portable Environment for Composition." *Computer Music Journal*, forthcoming.

Steele, Guy Lewis Jr. 1984. *Common Lisp: the Language*. Burlington, MA: Digital Press.

*Symbolics 1986*. Cambridge MA: Lisp Machine Manuals.

Winston, P., and B. Horn. 1981. *Lisp*. Reading, MA: Addison Wesley.

## CONTENTS

6	From the Domaine Musical to IRCAM	PIERRE BOULEZ in conversation with PIERRE-MICHEL MENGER
20	Dating Charles Ives's Music: Facts and Fictions	CAROL K. BARON
COMPUTER MUSIC FORUM		
58	Sieves	IANNIS XENAKIS
80	Statistics and Compositional Balance	CHARLES AMES
112	Speech Extrapolated	DAVID EVAN JONES
144	Observations in the Art of Speech: Paul Lansky's <i>Six Fantasies</i>	DAVID LOBERG CODE
170	It's about Time: Some NeXT Perspectives (Part Two)	PAUL LANSKY
180	Processing Musical Abstraction: Remarks on LISP, the NeXT, and the Future of Musical Computing	JOHN RAHN
192	LISP as a Second Language: Functional Aspects	PETER DESAIN
224	A Major Webern Revision and Its Implications for Analysis	ALLEN FORTE
256	Row Derivation and Contour Association in Berg's <i>Der Wein</i>	DAVE HEADLAM
294	The Systematic Chromaticism of Robert Moevs	JAMES BOROS
324	A Conversation with Robert Moevs	JAMES BOROS
COLLOQUY AND REVIEW		
336	The 1989 International Computer Music Conference: An Overview of the Concerts	RICHARD KARPEN
344	<i>Learning to Compose</i> : A Review	CRAIG WESTON
	Poetry	
	1, 7, 79, 144	LOLA HASKINS
	Graphics	
	5, 223, 293, 343	RORY BUTLER
352	Editorial Notes	
355	Correspondence	
359	Personae	
361	Acknowledgments	



# *Computers*

## *In Music Research*

---

---

Volume II

Fall 1990

## **Parsing the Parser: a case study in programming style**

by Peter Desain<sup>1</sup>

### **INTRODUCTION**

Much of my effort during the last years was directed at explaining neat programming styles and functional use of programming languages (Desain 1990). My audience however tended to question the realism of the beautiful three line programs I mostly used as examples. They doubt whether they were not just of a didactical worth, real world problems being too hard to handle without compromise in this way. To contradict such insinuations I went looking for a really complex problem, in the form of a published program which I could convert into a good functional programming style. In research in expressive timing in music I came across the excellent papers of Longuet-Higgins (Longuet-Higgins 1976, 1979) describing a musical parser that, next to tonal analysis, parsed performed music rhythmically. It produced a metrical hierarchical structure, while tracking tempo changes and rounding performance inaccuracies. The actual code of the program written in POP-2 was attached to the article as an appendix, which made

---

<sup>1</sup>Christopher Longuet-Higgins is to be thanked here. His work is a continuous source of inspiration, and his encouragement was a great help in this research. With help from Henkjan Honing the first attack of the POP-2 code was made, and some of his good ideas were used in this article. This research was partly supported by an ESRC grant under number A413254004.

the proposed endeavor possible.

Researchers should really be encouraged to publish the actual code or parts thereof, like Longuet-Higgins did. Firstly it provides a means to verify or falsify the claimed results. Secondly it forces the author to account for every detail in the system. Especially if the algorithm is claimed to provide a cognitive model, it is important to study its internals, the data and control structures used, so as to be able to state the predictions it makes. Naturally, bulky programs are not useful as appendices to articles as usually nobody takes the trouble to look at them. Of more use are micro versions, from which unnecessary details are removed. Constructing such a micro version can be of benefit to the researcher too, being forced to decide what is essential and what are mere 'bells and whistles'. More than once I witnessed remarkable progress in research caused by the insight yielded while trimming a program to its bare minimum. In (Shank and Riesbeck, 1981) good examples can be found of micro versions of some famous computer understanding programs.

In the case of the Longuet-Higgins parser it seemed that the code could indeed already be called a micro version, apart from the fact that the tonal and the rhythmical analysis, which are being dealt with separately in the theory, were embodied in one program. However, speaking to several colleagues that had tried to understand the code, I discovered that the program did not at all function as a clarifying, and helpful addendum to the article itself. All readers (including myself) were put off by the difficulty of the program, even though the underlying theory was described extremely well. This was not because it was written in the (now obsolete) language POP-2, but because the

program itself used many awkward programming constructs: side effects, different binding regimes and scoping rules, non-local exits and even a GOTO. The term spaghetti program seems to be a good description of this piece of code (and indeed prof. Longuet-Higgins is fond of Italian food), and one has to have an appetite for reverse engineering to rediscover the workings of the program from the code. But at least there was well described code available, which cannot be said of all publications about AI programs. It is a symptom of the general state of AI research that rational reconstruction is becoming an significant AI methodology. Campbell (1990) defines this technique as reproducing the essence of a programs behavior with another program constructed from descriptions of the purportedly important aspects of the original, trying to verify claims made about this program. It seems a waste of effort as these programs should have been published and described well in the first place, but it makes again clear that the equation 'the program is the theory' that has had a long standing history in AI, does not hold.

When finally Edward Lisle, Longuet-Higgins present collaborator, remarked that I would never be able to port the code to LISP because one needs to be an expert LISP programmer to do that, I had gathered enough incentive to embark upon the task of rewriting the program in an understandable style. In this paper I will describe the route I took in porting the program, in the hope that similar methods will be useful for the reader on other occasions. I will also show how the standard repertoire of the LISP and AI programmer can be used to create elegant and modular programs for complex problems. The resulting code is rather easy to read for humans—which is the main, but often forgotten,

aim of programming—and can be much more easily experimented with, changed and tested.

But before I embark upon describing the program and the port, I first have to make clear that looking at the model so closely only boosted my admiration for the research itself. And although the flow of data and control needed for the theory is rather sophisticated, the questions involved are described very well in the two papers, and the performance of the algorithm is remarkable. Furthermore the code was written almost two decades ago when lots of the techniques I used, now common practice, had not yet found their way into the literature. Prof. Longuet-Higgins was so kind to encourage me on this task and clarify several issues. Any criticism in this article, in which I cannot hope to match his personal eloquent style and merciless polemics (see Longuet-Higgins, 1983), has to be seen in the light of these remarks.

#### UNDERSTANDING THE THEORY

The parsing and quantization process is known to be very hard. Different methods can be found in (Desain & Honing 1989, 1991) and a comparison of the performance of Longuet-Higgins' symbolic method and a connectionist model for the same task is given in (Desain, 1991). But first and foremost the reader must be asked to read the original papers, or the corresponding chapters of (Longuet-Higgins 1987), as I can only give a brief outline of the theory here.

The rhythmical part of the parser uses a hybrid method of tempo tracking plus the use of structural knowledge about meter. In this method the tempo tracking is done with respect to a time interval that

can span zero or more notes. At top level this time interval represents a beat. It is subdivided recursively in 2 or 3 parts looking for onset times near the start of each part, until the interval contains no more onsets. The 'best' subdivision is returned, but the program is 'reluctant' to change the number of subdivisions (the pulse) at each level. At each recursive level the interval length is adjusted on the basis of the onsets found, just as in simple tempo tracking methods. An articulation analysis is then performed, dividing notes into tied parts and deciding where a rest occurs. Next to the quantized results this program delivers a hierarchical metrical analysis, whose top level is the beat and whose bottom level are made up of notes and rests. From the article we can identify the input and output of the system, the data-types used, the parameters, the procedural modules and their communication. What follows is an outline of those issues, taking into account only the relevant ones from a rhythmical perspective.

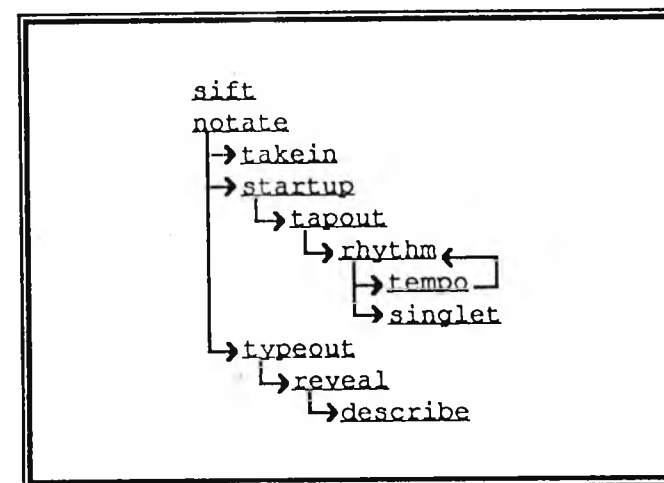
The input of the system consists of an ordered list of notes. Each note has an onset time, an offset time and a pitch. The output of the system consists of a list of trees, one for every analyzed beat. A beat is just a period of time, slicing the data in consecutive intervals. Each tree is of a combined binary-ternary nature, which means that each node has zero (in case it is a leaf of the tree) or two or three sub-trees. The arity of each internal node is called the pulse. During the construction of the tree there is a horizontal flow of pulse information through the layers of the tree, seeking to maintain the same pulse at a certain level as long as possible. The list of proposed pulses for the tree at each level is called meter. During the construction of the tree a strict left to right order is maintained, and new sub-trees are created on a generate and test basis.



This means that a proposed (and constructed) binary sub-tree may be rejected in favour of a tertiary one. The generate and test procedure is non-standard in that it may, after checking and rejecting the first alternative, still reject the second in which case as yet the first alternative is chosen. Notes, whose onsets happen in rapid succession (like in a trill) are collected in a group and treated as if they started at the onset of the first note in the group. Associated with each leaf of the resulting tree is a possibly empty, annotated list of sounding notes. There is one parameter identified in the program called *tolerance* which is used in different places as the allowed margin of deviation in deciding if notes start or stop at a certain times.

The main flow of control is dealt with in a mutual recursion of the procedures *tempo* and *rhythm*. *Tempo* decides if there is another subdivision needed, if not it calls *singlet* (bottom case of recursion). If there is, it calls *rhythm* which tries one or more subdivisions and calls *tempo* recursively on them. *Rhythm* then returns the best fitting sub-tree. There is some pre-processing (*sift*, *takein*), some top level initialization, (*startup*) and the post-processing is mainly done in the form of printing procedures (*typeout*, *reveal* and *describe*). The call-graph (it is not a hierarchy because of the recursion) depicts it all neatly (Figure 1).

Figure 1: Call graph of the parser routines.



### Planning the endeavor

Because the flow of data and control in the program is so complicated, and because a complete rewrite from scratch could only use the text of the articles as specification, which seemed not enough, I planned the whole endeavor as follows. I would try first to reconstruct the system as a LISP program directly translating the POP-2 constructs into LISP and staying as close as possible to the original code. During that stage some of the example input data incorporated in the articles could be used to test the program. It was planned to do only one change at a time and to keep all intermediate files for easy recovery and documentation. Every question about the working of the program was added as comment to those files. During that stage I would try to resist

improving the code making the translation as algorithmic as possible. It was decided to concentrate on the rhythmic part, leaving the simpler harmonic analysis for the future. When that program would work well a test suite had to be built to check the output using all of the available examples. Those two means, trying to translate the POP-2 code as directly (and mechanically) as possible and checking the input/output of this program, would hopefully insure a version (called LISP Program 1) that was semantically equivalent to the original. However, "testing can show the presence of bugs but not their absence" (Dijkstra in Bentley 1988, p. 60). I planned then to add some trace code to check the internal workings of the program and clarify the list of questions that was building up. The tracing code and the test suite all belong to the necessary scaffolding that has to be erected when building or modifying a program (Bentley 1988). Although these techniques are common practice for every experienced programmer it is a pity that they are so often neglected in student texts on programming, and that support facilities for these temporary constructs are lacking from almost all programming environments. In the next stage some non-essential add-ons could be removed and then enormous changes would be necessary to clean up the code. Only semantic invariant program transformations were to be used, insuring that, however different its appearance, the behavior of the program was not changed by the surgery: it would still exhibit the same input-output behavior. After each change that would be checked, using a test run of the program suite. After resulting in a clean functional program (LISP Program 2) I suspected that the internal flow of information and control would be so much clarified that the remaining questions about the internal workings

could be answered and the crucial theoretical concepts could be made apparent from the code itself. Then at last one might be able to point at possible improvements of the algorithm. With this plan in mind the next stage was started.

### LITERAL TRANSPORTATION

When starting this project I was not able to locate a POP-2 manual. Afraid that the whole project would turn out to be a piece of computer science archeology I was glad to find that POP-11 (the successor of POP-2) is still widely used and a manual (Barett e.a. 1985) only left a few constructs found in the program unexplained. When I eventually found the POP-2 manual it was quit instructive to see the sloppy semantics (Burstall e.a. 1986 reference manual p. 14-15) of this language, which was used for a lot of large programming projects and has had a great influence in the AI-community in Britain for a long time. Common LISP (Steele, 1984) was chosen as the LISP dialect because of the wide availability of implementations of this standard, although SCHEME (Abelson & Sussman, 1985) would yield even more elegant code.

We will now take a dive into the details of the POP-2 and LISP code, anyone interested in a more global view may skip this part and start reading again at the section 'Lispizing the code' or even move ahead to 'Theoretical issues'. The relevant parts of the POP-2 code are shown in appendix 1. I have inserted some comments (printed in italics) and added the line numbering. Any line numbers in the text refer to this appendix. Translations of POP-2 constructs in LISP are shown in Table 1.







environment: the output locals of `rhythm` declared in line 51. This ugly construct makes it impossible to study the behavior of a function in isolation—one has to search always for the part of the program that happens to call the function to decide upon the values of these variables on entry. In (Barett e.a 1985, p 37,38) this is called a convenient feature. Which only once again shows that one has to take care for the words like "handy" or "convenient". They inevitably signal danger when they occur in the description of the semantics of a programming language. A related problem is the fact that assignment to a local variable does not change the value of the variable with the same name in the calling context, but an assignment to a free variable (a variable that is not declared as argument or as output local or local), does change the value of the corresponding variable in the calling code. So only after scanning the whole program one can decide that the assignment to `nlist` in line 54 of `rhythm` is really an assignment to the the `nlist` argument of `tapout` in line 105 just because `rhythm` happens to be called in `tapout`. Such so called dynamic scoping makes the behavior of a function depend upon the actual coding of the functions it uses and by which it is used, complicating the semantics of the language, and of any program written in it.

Things become worse when programmers do not understand these constructs or use them in a sloppy way: why is there a `tol` local variable in line 106, while there is also a global variable `tol` in line 30 and it is used as a truly global constant? Why is there a local `stop` in line 51 when it is also declared as output local in the same line? Why is there a global `metre`, even declared twice in line 30 and 17, when it is clearly used in local backtracking in line 101 and 97? Indeed the whole

program is a sloppy mess regarding scope and binding of variables. The translation of these aspects was the most difficult part of the port and I will try to outline how I tried to make all communication between parts of the program explicit and lexical, which means that only references and assignments are made to variables that are local (and textually visible) to the piece of code under construction. All the semantics then become static, which means that the meaning of a part of the program can be described independently from the actual computational route taken by calling and called routines, only depending upon the program-text of those routines. Let us consider the example function definition in Table 2 in which every possible use of variables is listed.

Table 2: Translation of a POP-2 function into LISP.

construct	POP-2	LISP translation
function definition	function fun a b -> c d vars e f; <i>body with</i> <i>g referred to but not assigned to</i> <i>h referred to and assigned to</i> <i>c,e referred to before assigned to</i> <i>d,f assigned to before referred to</i> end;	(defun fun (a b c e g h) (let (d f) <i>translation of body</i> (values c d h)))
function call	fun(i,j)->k->l;	(multiple-value-setq (k l h) (fun i j c e g h))

In the function body there are formal arguments, output locals, locals, and free variables They are used (referred to and assigned to) in different order. In the translation into LISP we have to add formal arguments to the function for some output locals and free variables,

because they will be assigned an initial value by POP-2 upon entry of the function. Now we will do that initialization explicitly by adding them to the argument list in the function call. The output locals will have their values returned as the multiple results, which has to be done explicitly in LISP. Since the assignment to a free variable in the body will have effect in the calling context as well, we have to add this variable to the multiple results as well and do the assignment explicitly in the calling program. Note that not in each routine of the POP-2 code all the ways of treating variables are used, yielding a simpler translation. But as in general we now have added an assignment in the calling context, the caller may again change its translation, initiating more changes etc. Consequently the translation is not a very simple task. But after this translation all flow of information is clear and we can get rid of some of the remaining global variables because the routines will be explicitly passed their values as arguments when they need them. Only *metre* and *tol* remain global and are declared once at the begin of the program text to retain for the moment the spirit of their initialization (line 30).

For individual note a record datatype was used, which can be declared in POP-2 with the *recordclass* construct (line 1) which automatically defines accessor functions for each field of the record (in our case *onset*, *pitch* and *offset* in lines 21, 22 and 35) and a constructor function (in our case *consnote*, line 10). I took the liberty of defining lists of note structures directly (see the end of Appendix 3), without reading an input file, and passing one such list as argument to *startup* thereby removing the need for the *takein* procedure (line 150). Pitch names were inserted in the data, because it is then easy to

check the output against the output shown in the articles, even though I left out the tonal analysis. The examples given in the articles are a simple musical cliché and two fragments of the cor anglais solo in the Prelude to Act III of Wagner's *Tristan und Isolde*. All data is given in Appendix 2 retaining the original notation for pitches.

I could not resist the temptation to re-order the procedure definitions top down (using the call graph) and to separate them in several groups. Although this obscured the relation of the program with its POP-2 parent it is so much easier to navigate through code that is ordered well. The resulting program (LISP program 1) produced the same results as shown in the original article and I can assure you that I felt great relieve the moment I saw it parsing the input correctly. The only difference with the output listed in the articles is the output of the first 'count-down' beats, not shown in the original article. Later Christopher Longuet-Higgins affirmed me that his program does output these as spurious rests at the beginning of its parse. Finally I have to mention one typographical error in the original program: the comma in line 21 should be a period.

#### "LISPIZING" THE CODE

Now the program could be trimmed to remove all aspects that were not to be part of a real micro version. For example the function *sift* is a trivial function to remove spurious key bounces that stem from the recording equipment used. The article should mention such pre-processing but it surely is no part of the parse algorithm itself, and it has even less relevance for the cognitive model. Another feature that should not be in the micro version is the grouping of a number of beats

on one output line, faking an analysis above the beat level, while there is just a clever trick used: the musical data is played preceded by a measure of count down beats on the same low key, the number of which is used to collect the beats in measures on the output after analysis. This trick, explained in the article, could be well worth its value in a practical implementation, but it is again far from central to the theory and only distracts the reader of the program. The length of the last count down beat is used as a initial estimate for the beat length. This parameter of the program is thus concealed in the data, it will be cumbersome to experiment with different slices of data, or different initial estimates of the beat. But what is worse, a real issue that the theory does not tackle: how do human listeners pick up the initial beat of a piece of music, is hidden from view by inserting this information in the musical data. It must be said that it may be a wise decision to leave this difficult question aside in the theory, and the article is quite explicit about that, but then again it should be as easy to understand that fact from the program itself. Thus initial beat duration is changed into a parameter of the top level function (and for compatibility with the old data, it is optional and uses the old method if not specified).

As a general rule it is best not to do much processing in routines that produce text, because in textual output all internal structure is lost and other programs often cannot make use of the results in flat text format. In our case the scaffolding, like the test suite and programs that measure the sensitivity of the parser to parameter changes, is much easier to write if they can inspect the whole result structure from the parser. So any output side effects like printing results in a neat way should be postponed. And, if they are included at all, they should be

written as an almost trivial add-on. Because the built-in LISP pretty printer (`pprint`) can do a nice textual layout of the result of the parser, handling indentation etc, I decided that in this case the parser should behave as a real function without side effects (note list in, structure out and no printing going on), and I moved the post-processing inside the parser itself.

As decided, all further transformations were done as semantic invariant program transformations. Examples of such transformations that retain the behavior of a piece of code but change its form, are the substitution of a function call for its body (with appropriate substitutions of variables), the collection of statements into a (help)function, the 'unwinding' of loops, the movement of statements in or out conditionals and function bodies, the removal of uneffective assignments, the change of order of independent statements, and the systematic change of variable names. To begin with the latter: some abbreviations of variable names seem silly (`syncop` instead of `syncope`, `tolerance` instead of `tol`, etc.), I changed these all to there full names, but I did not consider changing them to names that described their role better, nor did I change the name of any routines, so as to maintain the relation with the original program.

Since `metre` was already an argument to the main parsing routines, stemming from the translation of function definitions, I could remove it completely as global variable, and turn it into a parameter of the top level `notate` function. And indeed, just like the initially expected beat period, the expected meter is conceptually an argument of the parser. The same was done for the `tolerance` parameter. This clean-up of global variables made the whole `startup` routine

superfluous.

Rewriting `tapout` as a recursive procedure, would made the output local `sequence` as a temporary hold of the growing structure unnecessary, and also would automatically built the structure in the correct order such that a final reverse (line 117) becomes obsolete. These simplifications come often for free when turning iteration into recursion. Using the loop macro, as was done here has the same benefits, and made the `tapout` function superfluous.

Some routines should really have been carved up into smaller units, either because they are just to large to comprehend as a whole, or because really a separate theoretical issue was being dealt with and modularization would show more clearly on which information the decisions taken were based. For example, in lines 34 to 45 an analysis of the type of articulation is taking place intertwined with some maintenance of data-structures (e.g., `last`: list of pending notes, and `group`). A simple separation of concern as implemented in the extra help function `articulation-mark`, makes it completely clear that the decision on the type of articulation is taken not on the basis of the gaps between notes, but on the basis of the gap between the end of a note and the beginning of the next metrical unit (Longuet-Higgins, 1987 p. 127).

The collection of notes from `note-list` in a group in lines 53 to 64 in `rhythm` is another complicated piece of code which deserves to be separated and cleaned up (the resulting function is called `collect-group`). Of course the `goto` construction (line 55) is obscuring and completely unnecessary. If you are not convinced of this: (Dijkstra, 1968) is the standard text to explain the horrors of `goto`'s.

Defining helping functions (`onset-before` and `offset-before`) for deciding if the onset or offset of a note occurs before a certain point in time with a margin of tollerance in a certain direction, a frequent operation in the code, again makes the program more readable.

We can keep the data representation of a group a bit closer to the problem at hand. For efficiency reasons a group of notes is represented backwards in the original code. This obscures the calculation of the articulation mark (the actual mark is calculated on the basis of the latest note in a group, not on the basis of the first one as line 35 seems to suggest. Furthermore, in a later stage and in an unrelated piece of program text, this reverse coding has to be undone (line 40). In micro programs one should not worry about tiny gains in processing speed but either keep the data structures as close as possible to a 'natural' representation of the problem at hand—or hide an encoding in a data abstraction layer.

One further problem shows when looking closely at the code. In lines 57 to 60 subsequent notes are removed from the `note-list` and put into the `group`. But in line 62 an actual undoing of the last of such actions might happen. This has severe consequences for high level descriptions of the parser as a process in which a stream of notes is fed in and a parsed structure per beat comes out. Unreading a stream is a rather awkward operation and might be psychologically unplausible. However, given the current algorithm I couldn't do better then change a 'read' plus subsequent 'unread' into one 'peek' operation. Having a more decent sense of a timed input stream, it might be possible to judge on the basis of the offset of the previous note and the current time if no note-onset had arrived yet and the group may be closed and processed.



We have now arrived at the most difficult part of the code: **tempo**. Firstly one can see that although **again** is given a numerical value that is incremented each time through the loop, the loop is done twice at most. So **again** can be changed to a boolean. It is a common error in programming to use under-restricted data types. But to get some grip on the code it is better to unwind the loop, writing down its body twice and get rid of the **again** variable and the non-local exit of line 95. Looking through the processing of **pulse** and **metre**, which is done in an incredibly ugly way in lines 76, 79, 94, 97, 99, and 101, and the clumsy storing and retrieving of local state in lines 74 and 98, the control structure slowly emerges. A trial is done with initial values, if it succeeds, its results are returned. If it does not, its results are stored in a variable called **new** and a second trial is initiated with initial values reset to their old values, but with an alternative **metre**. If this one succeeds, its results are returned, otherwise the results of the first trial is preferred because it used the expected meter. This generate-and-test process calls for a help function (named **trial**) which might be called twice with partly the same arguments, to generate the alternatives, elevating the need for resetting variables to their old values. Now **old** can be removed. Making the control structure stand out clearly in this way facilitates discussions about its nature and the cognitive plausibility of such constructs. It also enables the design of custom flow of control, which can be done in LISP by adding continuations to the **trial** function—which are functional arguments that specify what should be done with the result of it (Abelson & Sussman, 1985)—with a new control structure programmed as a function with functional arguments to specify the details, or with the general macro facilities.

The latter actually specifies an extra layer, a new programming language, in which the program is embedded. I choose here for the second construct because it completely isolates the control issue but is somewhat easier to write than macros. Note that the control structure cannot be called proper backtracking because failure or success is decided at the top level of each decision-subtree. It is implemented in the new function **generate-and-test**, which is given a way to generate a result (the function **trial**), a way to evaluate this result (made by **make-test**), a way to make alternative arguments for the generator (called **alternative**), and some initial arguments for the generator to use in the first trial. It will return either the result of the first or of the second trial according to the rules given above.

Some details about the body of **tempo** still have to be clarified. In line 99 there is an expression  $(5 - \text{pulse})$  which completely obscures the fact that **pulse** in the program can only be 2 or 3, and the effect of this expression is the switch from one to the other. Conceptually 2 and 3 as possible values of **pulse** are not related by their sum being 5 but by being the two smallest primes. Cleverness like this should be abolished from all micro programs (maybe even from all programs). By modularizing this operation into a function **alternative-metre** that calculates a changed meter (not just a changed pulse) a theoretical issue is again highlighted: in changing meter at a certain level, one disposes of all metrical structure below that level, which will again default to divisions into two, an assertion that clearly has cognitive relevance and can be tested as such. Modularizing the test for acceptability of a result into one function (made by **make-test**) again makes part of the theory stand out, showing that a different meter is tried if the metrical unit

fails to end with a note (a syncope), or ends rather early or late (Longuet-Higgins, 1987, p. 129). The method of tempo tracking used in *tempo* stands out more clearly now. An extra parameter *speed* will make one more hidden parameter explicit: the speed of tracking tempo at beat level, taken as a constant 2 in line 113). The speed at lower levels can be seen to be equal to 1/pulse.

Because in the program most assignments are now assured to have only local effects within function bodies and are done at most once, I could gradually change them into local binding constructs which made this even more clear. I changed *multiple-value-setq*'s into *multiple-value-bind*'s, moved initial assignments to local variables into the *let* headings where they were declared, and ended up with a program which was side-effect free except for two *multiple-value-setq*'s within a loop construct. This means that if spotting a variable referred to in the program, I could be sure that it was given an initial value only once and see immediately from the locally surrounding program text how that was done, and anywhere within the scope of that construct this variable would retain this value. Thus a computational variable would look much more like a mathematical one, and the actual dynamic aspects of computation-steps taken are now separated from- and irrelevant for- these issues.

The value of *stop*, returned from *rhythm* can be 0 (line 87), in which case it is used as a flag to indicate a detected syncope. It is in general unwise to store conceptually different types of information in one variable. So I made an extra result variable called *syncope*. Now one can return also a useful *stop* result in case of syncope: the initially estimated end of the unit. Moving even the tempo-track calculation to a

lower level made some more code simpler, to the expense of having to pass the estimated end of the unit (*aim*) and the tempo-track speed (*speed*) downward.

The next major surgery was the un-merging of structural analysis (based on the onset of notes) and articulation analysis (based on their offsets). Although they both use information about the tree being constructed (like *start*, *stop* and *period*) that is only available temporarily (during the local construction), I did feel that the maintenance of the list of still sounding notes in the *last* variable obscured the working of the structural analysis. And because it is well known that merging different algorithms into one is one of the main sources of bugs and confusion in programming, I decided to un-merge the algorithms. This would have the added advantage of making them available as separate modules. *Tapout* will now deliver tree structures in which the leaves contain only note groups whose onset start there but which are annotated with the extra information needed by the articulation analyzer. And because part of the articulation analysis was already done at a later stage (during the printing in *describe*), moving all of it to a separate module did not seem such an essential change. There are of course cognitive arguments to consider the two processes as intertwined, but then again one could consider the program as being implemented on a lazy evaluator which would only do a round of structural analysis only when the articulation analysis needed the result, thus eliminating any psychologically unplausible long-term intermediate storage. It is very difficult to describe the relation of a program to a cognitive model, especially to describe where and how the algorithm and the language semantics over-restrict the model (*describe*

the model in more detail than intended) and I strongly disagree with any indication of ducking these issues as in: "the program is, of course, no more than an embodiment of these ideas in computational form" (Longuet-Higgins 1987, page 183). Although it has to be said that at the time of the original paper the very idea that a program can be a medium for expressing ideas about cognition was a novel one.

The last changes delivered the final code: LISP program 2 (see Appendix 2 for the full code plus an example of its use). One of the most heard (and silliest) arguments used against a clean programming style is the supposed expense in calculation speed and memory usage. This argument was again proven false by this program that performs even faster than version 1 (using the three examples, without output printing, running on a Mac Ilci in Allegro Common Lisp).

The test suite I used during the transformations described above was quite small. It consisted of the three examples shown in the articles and given in Appendix 3. Because I suspected that subtle bugs (e.g., in the tempo tracking) might produce correct results on these examples but on the basis of wrongly calculated internal values, I added cases in which the tolerance was just so far off that the program came up with a wrong answer. The test suite then used that value plus the wrong answer as a reference to judge a correct working of the modified program. This way I could catch any subtle errors in my port which would otherwise go unnoticed because of the rounding mechanism. For one aspect of the program representative examples were missing: **collect-group** only once comes up with a group of two notes (in example *tris*: the notes at 1928 and 1932 centiseconds). The examples contain just not enough trills, grace notes etc. to test its working thoroughly.

Because the number of arguments to functions is large and the data structures passed around may also be large, build-in trace facilities bury one in pages of text. I designed a custom tracer that produced only the relevant information. Because this is still a lot of information, a graphical trace program would be desirable.

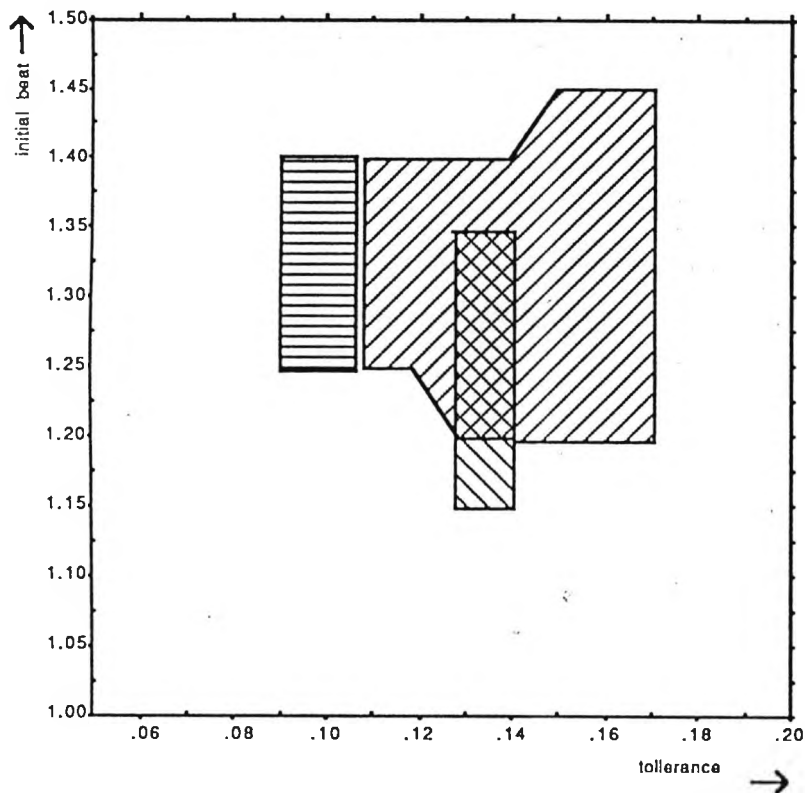
## THEORETICAL ISSUES

### Parameters

The different settings for the tolerance parameter used in parsing the examples in the article (0.10 sec. for the *cliche* example and 0.13 sec. for the others) raise the question how sensitive the model is for its parameters. It is easy to do an experiment to check the range of parameter values in which the parser works well for the examples. In Figure 2 these ranges are shown for the tolerance parameter, with the initial beat estimate used as a second independent variable because it may disturb correct parsing.

It is hard to base a conclusion on the basis of this limited set of three musical examples, but the small size and the non-overlapping nature of the regions identify a problem concerning robustness here. A delicate parameter setting, to be done anew for each piece of data, may be justifiable in the context of a technical tool, it is not so in the context of a cognitive model. These maps show how the initial beat estimate is more or less independent of these results. Thus the tempo tracking taking place at the highest (beat) level is not the source of the problems, but the processing at deeper levels of subdivision is.

Figure 2: Parameter ranges resulting in a correct parse.



The same conclusion can be drawn from the fact that allowed settings for the speed parameter in the successful regions are almost unrestricted within its full range between zero and one (not shown). This may indicate that for the data given, there is no heavy reliance on beat level tempo tracking. For definitive evaluations more data has to be used,

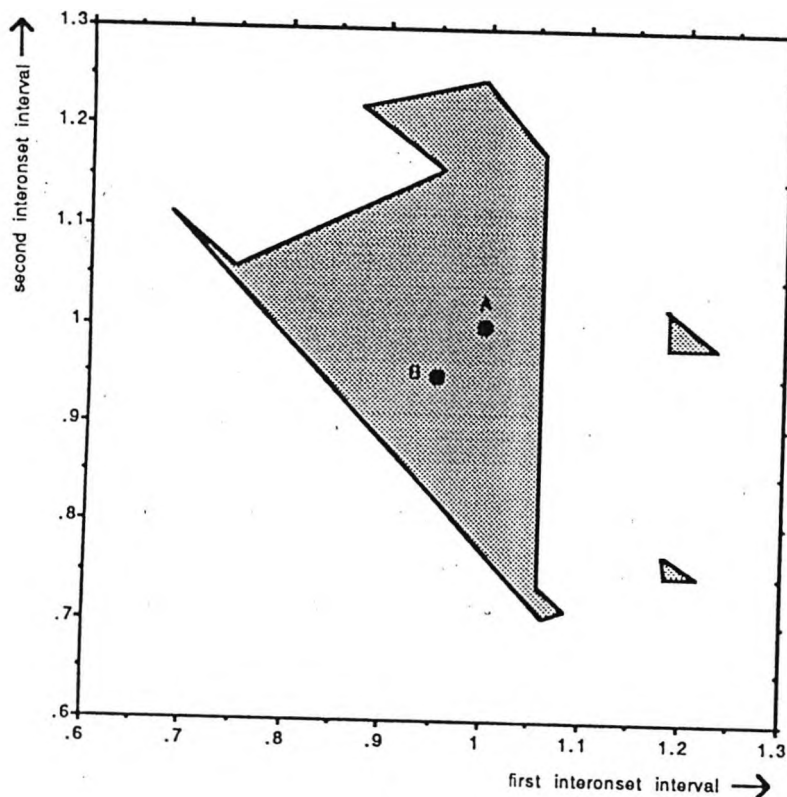
systematically mapping out the parameter space of the algorithm (parameter setting versus percentage parsed correctly). A line of further research that may make the parser more robust is the adaptation of the tolerance at deeper levels.

### Tempo Tracking

The tempo tracking at the highest (beat) level is implemented in lines 113. It simply averages the expected beat length and the measured one if the latter is available. Around line 88 a complex process controls the tempo tracking at deeper levels of subdivision. It incrementally adds each deviation found in the subdivision to the total period, and proportionally divides this period to estimate the position of the onset ending the next sub-period. Onsets are allowed within the tolerance around each estimate. For a three-division this effectively amounts to a positive feedback from the timing of second to the third onset. For example, if the onset ending the first subdivision is too early, the next onset is expected early as well (by  $2/3$  of error of the first onset). This process goes on, and assuming the second is early as well, the third is expected to be even earlier. This would yield nonsensical behavior were it not that after the completion of the parse of each subdivision the total length is passed one level upwards and compared with the beat estimate at that level, allowing for a deviation of tolerance. So here it turns out that after two short sub-divisions the third should be long to pass this test. Because in the parser the tempo tracking mechanism interacts with the change of meter decisions it is quite hard to derive at the mathematical characterization of the set of performed temporal patterns that will be recognized by the parser as a triplet, but testing this

empirically is quite simple.

Figure 3: *Temporal patterns interpreted as triplets.*



In Figure 3 a map of all possible subdivisions of a fixed beat length into three notes is given with the region that is identified by the parser as a triplet. The size of that region depends on the tolerance (which was taken as one tenth of the beat period). The initial pulse at that level was taken to be two. The idealized metronomical triplet is

located on the map at a point marked A. Given the performance of the parser it is not surprising that the actual form of the region is biased towards the pattern (short, short, long) found by Vos and Handel (1987) in their study of systematic deviations from the norm in their group of subjects who could play triplets well. This typical triplet pattern is located at point B. The same reasoning may be used to relate the behavior of the parser to empirical findings at higher metrical levels where elongation towards the end of the unit seems to be the rule (Todd, 1985 and Clarke, 1987). This knowledge about common performance practice implicitly incorporated in the model, which may explain its success, was not identified in the original article.

The unconnected small regions that also signify patterns parsed as triplets are a by-product of the interaction of the tempo tracking and the mechanism for meter change. While the former allows for a large area of triplet parsing extending to the right, the latter decides for a duple meter in most parts of that area, but fails to do so for the two small islands. A large tolerance will enlarge the islands, finally linking them up with the main region. But in general it can be said that the equivalence classes induced by the parser, the set of temporal patterns that will be interpreted as performances of the same rhythm, do not form one connected region. I could not find indications about the plausibility of this result in the literature about human rhythm perception, but it will greatly complicate empirical verification of the model.

#### Change of Meter

The decision when to change meter is taken, among others, on the basis of an syncope occurring in the last subdivision. This sometimes



seems too restricted, as syncopes in the other subdivisions might also contribute evidence for a change of meter. A more sophisticated model might adapt the reluctance to change the meter, to the metrical level in consideration, making the higher levels which resemble time signature less prone to changes than the lower levels. In the new program it is easy to experiment with this and other variants but in general the question seems very difficult to solve.

## CONCLUSION

One can ponder what the truth is in the following quotation about the procedures in the musical parser:

Such procedures are, unfortunately, much more difficult to specify precisely in English than in a suitably designed programming language: but this fact only underlines the value of casting perceptual theories in computational form. (Longuet-Higgins, 1987 p. 109)

But if a programming language allows the programmer to express such difficult constructs in a program, will it then be possible to see the ramifications of the theory? Or has theory degenerated into a black box mechanism that can only be used, instead of understood. I tend to attribute more value to the adagio:

If you can't write it down in English, you can't code it. (Peter Halpern in Bentley, 1988, p. 58)

But because moulding a theory into an implementation greatly helps in understanding and describing the theory in plain English, a computational approach in which the process of developing theory and implementing go hand in hand, is still most attractive to most AI researchers. That the resulting program often contains vestigial remains of earlier versions (Longuet-Higgins, personal communication) just calls for one more round of cleaning up and rewriting, as I hope to have shown in this article. The rewrite, at first sight a scholarly exercise, soon became a major undertaking because of the tangled flow of control and data in the program. But finally the program was made much more open for experimentation, verification or falsification and possibly extension. It is now easier to maintain and immerse in systematic testing, the more so since the algorithm was implemented in POCO (Honing 1990), an environment for research in expressive timing. In the process of rewriting, semantic invariant program transformations turned out to be very helpful as a methodology for reverse engineering as was the availability of a test suite to automate some test runs after each change.

I think that computational psychology can be a fruitful approach to the study of music, complementing musicology, experimental psychology and other disciplines. But to play this role well, researchers must force themselves to state their algorithmic contributions in the form of clean micro-programs and clarify which parts of the program are considered to model cognitive processes and which parts are implementation detail or technical tricks.

## REFERENCES

- Abelson, H., G.J. Sussman. (1985). *Structure and Interpretation of Computer Programs*. Cambridge, MA: The MIT Press.
- Barett R., A. Ramsay and A. Sloman. (1985). *POP-11, A Practical Language for Artificial Intelligence*. Chichester: Ellis Horwood.
- Burstall, R.M. and J.S. Colins and R.J. Poppelstone. (1968). *POP-2 papers*. Edinburgh: University Press.
- Bentley J. (1986). *Programming Pearls*. Reading, MA: Addison-Wesley.
- \_\_\_\_\_. (1988). *More Programming Pearls, Confessions of a Coder*. Reading, MA: Addison-Wesley.
- Campbell, J.A. (1990). "Three novelties of AI: theories, programs and rational reconstruction" In: *The foundations of artificial intelligence*. A source book, edited by D. Partridge and Y. Wilks. Cambridge: Cambridge University Press.
- Clarke, E.F. (1987). "Levels of Structure in the Organisation of Musical Time" in: *Music and Psychology, a Mutual Regard*. S. McAdams(Ed.) *Contemporary Music Review* 2(1).
- Desain, P. (1990). "Lisp as a Second Language, Functional Use" *Perspectives of New Music*. 27(1).
- \_\_\_\_\_. 1991 "A Connectionist and a Traditional AI Quantizer, Symbolic versus Sub-symbolic Models of Rhythm Perception" In I. Cross (Ed.), *Proceedings of the 1990 Music and the Cognitive Sciences Conference*. *Contemporary Music Review*. London: Harwood Press.

- Desain, P. and H. Honing, (1991). "The Quantization Problem: Traditional and Connectionist Approaches." in Balaban, M., K. Ebcioglu & O. Laske, eds *Musical Intelligence*. Menlo Park: The AAAI Press. (forthcoming)
- \_\_\_\_\_. (1989). "Quantization of Musical Time: A Connectionist Approach." *Computer Music Journal* 13(4) reprinted in.. Todd, P.M. & D.G. Loy (Eds.) (1991) *Music and Connectionism*, Cambridge, Mass.: MIT Press. (forthcoming).
- Dijkstra, E.W. (1968). "GOTO statement Considered Harmful" *Comm. ACM* 11(3)
- Honing, H. (1990). "POCO: An Environment for Analysing, Modifying, and Generating Expression in Music" In *Proceedings of the 1990 International Computer Music Conference*. San Francisco: Computer Music Association.
- Longuet-Higgins, H.C. (1976). "The Perception of Melodies" *Nature* 263: 646-653 and in Longuet-Higgins, 1987.
- \_\_\_\_\_. (1979). "The Perception of Music" *Proc. R. Soc. Lond. B* 205: 307-322 and in Longuet-Higgins, 1987.
- \_\_\_\_\_. (1983). "All in theory, the analysis of music" *Nature* 304(7), p 93.
- \_\_\_\_\_. (1987). *Mental Processes*. Cambridge, Mass.: MIT Press.
- Shank R.C. and C.K. Riesbeck. (1981). *Inside Computer Understanding*. Hillsdale, New Jersey: Lawrence Erlbaum.
- Steele, G.L. Jr. (1984). *Common Lisp: the Language*. Burlington, MA: Digital Press.
- Todd, N.P. (1985). "A Model of Expressive Timing in Tonal Music." *Music Perception* 3(1):33-58.

*COMPUTERS IN MUSIC RESEARCH*

Vos, P. and S. Handel. (1987). "Playing triplets: Facts and Preferences"  
in A. Gabrielson (Ed.) *Action and perception in Rhythm and  
Music*. Royal Swedish Academy of Music. 55.



## Appendix 1: relevant parts of original POP-2 code

```
1 recordclass note pitch onset offset ..extra fields declared here.
2 function sift notefile=>notefile;
3   maplist(notefile, lambda x;
4     if x.tl.tl.hd-x.tl.hd<5 then else x.close
5     end)->notefile;
6 end;
7
8 function takein notefile=>nlist;
9   maplist (notefile, lambda x;
10     consnote (applist(x, identfn), undef, undef, undef)
11     end)->nlist;
12 end;
13
14 functions res, int, modulate, hark, simplify, intervals, tuneup and vars
15 flag,k,l,m,n,place declared here
16
17 vars start beat position number group last metre nlist sequence;
18
19 function startup;
20   nil->sequence; nlist.hd.onset->start;
21   nlist.tl.hd.onset-start->beat;
22   nlist.hd.pitch->position;
23   nil->group; nil->last; 0->number;
24
25   loopif nlist.hd.pitch=position then
26     nlist.tl->nlist; number+1->number
27   close;
28 end;
29
30 vars tol metre; 13->tol; nil->metre;
31
32 function singlet->stop->fig;
33   vars period mark;
34   if group.null.not then
35     if group.hd.offset<stop-period/2 then "stc"
36     elseif group.hd.offset<stop-tol then "ten"
37     else "leg"
38     close->mark;
39
40     group.rev->last; nil->group; mark::last;
41   else
42     [%"tac",applist(last,lambda x;
43       if x.offset>start+tol then x
44       close end)%]
45     close->fig;
46     if nlist.null or nlist.hd.onset>stop+tol then 0
47     else nlist.hd.onset
48     close->stop;
49 end;
50
51 function rhythm start period->stop->fig; vars stop;
52   start+period->stop;
53   if nlist.null.not and nlist.hd.onset<start+tol
54   then nlist.hd::nil->group; nlist.tl->nlist;
55   else goto label
56   close;
57   loopif nlist.null.not and nlist.hd.onset<stop+tol
58     and nlist.hd.onset<group.hd.onset+tol
59   then nlist.hd::group->group; nlist.tl->nlist;
60   close;
61   if group.hd.onset>stop-tol
62   then group.hd::nlist->nlist; group.tl->group
63   close;
```

```

64 label;
65   if nlist.null or nlist.hd.onset>stop-tol
66     then .singlet
67     else .tempo
68     close->stop->fig;
69 end;
70
71 function tempo->stop->figure;
72   vars new old again pulse time count fig syncop;
73
74   [%nlist,last,group%]->old; 0->again;
75 loop:
76   if metre.null then 2::nil->metre
77     close;
78
79   metre.hd->pulse; metre.tl->metre;
80   nil->figure; period->time;
81   0->count; start->stop;
82   loopif count<pulse
83     then
84       count+1->count;
85       rhythm(stop, time/pulse)->stop->fig;
86       fig::figure->figure;
87       if stop=0 then start+count*time/pulse->stop; true
88       else stop-start+(pulse-count)*time/pulse->time; false
89       close->syncop;
90     close;
91     again+1->again;
92
93     if not (syncop or stop>start+period+tol or stop<start+period-tol)
94     then figure.rev->figure; pulse::metre->metre;
95     exit;
96     if again=1 then
97       [%nlist,last,group,figure.rev,stop,pulse::metre%]->new;
98       old.destlist->group->last->nlist;
99       (5-pulse)::nil->metre; goto loop;
100    else
101      new.destlist->metre->stop->figure->group->last->nlist;
102    close;
103  end
104
105 function tapout nlist->sequence;
106   vars start beat tol group last stop figure;
107   loopif nlist.null.not
108     then
109       rhythm (start, beat)->stop->figure;
110       figure::sequence->sequence;
111
112       if stop=0 then start+beat
113       else (stop-start+beat)/2->beat; stop
114       close->start;
115     close;
116     nil->metre;
117     sequence.rev->sequence;
118   end;
119
120   vars max,min,symbols,symbol declared and initialized here
121   function name declared here
122
123   function describe fig; vars word;
124     fig.hd->word; fig.tl->fig;
125     if fig.null then [rest]
126     elseif word="tac" then
127       "tied"::maplist(fig,index<>symbol)

```



```
128     elseif word="leg" then maplist(fig, name)
129     else [%applist(fig,name),word%]
130     close;
131 end;
132
133 function reveal figure;
134     if figure.hd.isword
135     then figure.describe
136     else maplist(figure,reveal)
137     close;
138 end;
139
140 function typeout seq; vars count;
141     0->count; 1.nl;
142     applist(seq,lambda x;
143         if count = number then 1->count; 2.nl
144         else count+1->count
145         close; x.reveal .pr
146     end); 2.nl;
147 end
148
149 function notate notefile;
150     notefile.takein->nlist;
151     .startup;
152     nlist.tapout->sequence;
153     nlist.tuneup;
154     sequence.typeout;
155 end;
```

## APPENDIX - 2: The Parser

```

;;; Longuet-Higgins Musical Parser,
;;; Micro-version 2, in Common Lisp (uses loop macro), Peter
;;; Desain, 1991.
;*****
; top level

(defvar *tolerance*)

(defun notate (note-list &key
              (metre '(2))
              (tolerance 10)
              (start (onset (first note-list)))
              (beat (- (onset (second note-list))
                       (onset (first note-list))))
              (speed 0.5))
  (setf *tolerance* tolerance)
  (loop while note-list
        with figure
        with group = nil
        do (multiple-value-setq
            (start figure group metre note-list beat)
            (rhythm start beat group metre note-list
                    (+ start beat) speed))
          collect figure into figures
          finally (return (articulation figures))))

;*****
; main parsing routines

(defun rhythm (start period group metre note-list aim speed)
  (let ((stop (+ start period)))
    (multiple-value-bind (group note-list)
      (collect-group group note-list start stop)
      (if (or (null note-list) (not (onset-before
                                     (first note-list) stop '-)))
          (singlet start period group metre note-list aim speed)
          (tempo start period group metre note-list aim speed))))))

```

```

(defun singlet (start period group metre note-list aim speed)
  (let* ((stop (+ start period))
         (syncope (or (null note-list)
                       (not (onset-before (first note-list) stop '+))))
         (end (if syncope aim (onset (first note-list)))))
    (values end (list start stop period group) nil metre note-list
            (+ period (* speed (- end aim))) syncope)))

(defun tempo (start period group metre note-list aim speed)
  (apply #'values
         (rest (generate-and-test
                #'trial
                (make-test (+ start period))
                #'alternative
                metre start period group note-list aim speed))))

(defun make-test (aim)
  #'(lambda (syncope stop &rest ignore)
      (and (not syncope)
           (< (abs (- stop aim)) *tolerance*))))

(defun alternative (metre &rest arguments)
  (cons (alternative-metre metre) arguments))

;*****
; control structure for change of metre

(defun generate-and-test (generate test alternative &rest states)
  (let ((result1 (apply generate states)))
    (if (apply test result1)
        result1
        (let ((result2 (apply generate
                               (apply alternative states))))
          (if (apply test result2)
              result2
              result1))))))

```

## COMPUTERS IN MUSIC RESEARCH

```
;*****
; subdivide a period

(defun trial (metre start period group note-list aim speed)
  (loop
    with pulse = (pop metre)
    with sub-start = start
    with sub-period = (/ period (float pulse))
    with syncope
    for count from 1 to pulse do
    (multiple-value-setq
      (sub-start fig group metre note-list sub-period syncope)
      (rhythm sub-start sub-period group
        (extent-metre metre) note-list
        (+ start (* count sub-period)) (/ (float pulse))))
    collect fig into figure
    finally (return (list syncope sub-start figure group
      (cons pulse metre) note-list
      (+ period (* speed (- sub-start aim)))))))

;*****
; metre calculus

(defun alternative-metre (metre)
  (case (first metre)
    (2 '(3))
    (3 '(2))))

(defun extent-metre (metre)
  (or metre '(2)))

;*****
; collect group of synchronous notes

(defun collect-group (group note-list start stop)
  (if (and note-list (onset-before (first note-list) start '+))
    (collect-new-group (list (first note-list))
      (rest note-list) stop)
    (values group note-list)))
```

## PARSING THE PARSER

```
(defun collect-new-group (group note-list stop)
  (if (and
      (collect-group-test (first note-list) (first group) stop)
      (or (collect-group-test (second note-list)
        (first note-list) stop)
        (onset-before (first note-list) stop '-)))
    (collect-new-group (cons (first note-list) group)
      (rest note-list) stop)
    (values (reverse group) note-list)))

(defun collect-group-test (note1 note2 stop)
  (and note1
    (onset-before note1 stop '+)
    (onset-before note1 (onset note2) '+)))

;*****
; articulation analysis

(defun articulation (l &optional last)
  (cond ((null l) (values nil last))
    ((listp (first l))
      (multiple-value-bind (result1 last1)
        (articulation (first l) last)
      (multiple-value-bind (result2 last2)
        (articulation (rest l) last1)
        (values (cons result1 result2) last2))))
    (t (apply #'articulate-figure last l))))

(defun articulate-figure (last start stop period group)
  (let* ((new-last (or group (remove-if #'(lambda(note)
    (offset-before note start '+))
    last)))
    (pitches (mapcar #'pitch new-last)))
    (values (figure-describe group stop period pitches)
      new-last)))

(defun figure-describe (group stop period pitches)
  (if (null group)
    (if pitches (cons 'tied pitches) '(rest))
    (append pitches (articulation-mark (first (last group))
      stop period))))
```

```
(defun articulation-mark (note stop period)
  (cond ((offset-before note (- stop (/ period 2.0)))
        '(stc))
        ((offset-before note stop '-)
         '(ten))
        (t nil)))

(defun snoc (l x) (nconc l (list x)))

;*****
; help functions

(defun onset-before (note time &optional (margin 0))
  (< (onset note) (+ time (case margin
                            (+ *tollerance*)
                            (- (- *tollerance*)
                                (otherwise 0))))))

(defun offset-before (note time &optional (margin 0))
  (< (offset note) (+ time (case margin
                             (+ *tollerance*)
                             (- (- *tollerance*)
                                 (otherwise 0))))))

;*****
; data abstraction for notes

(defstruct (note (:constructor note (pitch onset offset))
                (:conc-name nil))
  pitch onset offset)
```

```
;*****
; example of the use of the program
#|
defining a note list
(defvar *cliche* (list (note 'start 154 227)
                      (note 'c 285 294)
                      (note 'g 322 327)
                      (note 'g 336 341)
                      (note 'as 349 383)
                      (note 'g 384 407)
                      (note 'b 445 453)
                      (note 'c 484 527)))

calling the program:
(notate *cliche* :tollerance 10)

will produce the following results:
((START TEN)
 ((C STC)
  (G STC)
  (G STC)))
((AS) (G TEN)))
(((REST) (B STC))
 (C TEN)))
```

#|

APPENDIX - 3: Test Data

The TRIS example: a fragment of the cor anglais solo in the Prelude to Act III of Wagner's *Tristan und Isolde*.



Note	Onset	Offset
START	24	114
START	148	238
C	274	399
G	400	554
BB	551	587
AB	586	671
EB	669	711
AB	707	794
D	795	831
G	829	860
C	863	895
F	895	989
G	987	1021
F	1020	1145
EB	1140	1242
D	1268	1282
C	1289	1298

BB	1308	1320
F	1332	1452
D	1450	1495
BB	1508	1517
A	1528	1536
AB	1546	1556
EB	1570	1696
C	1692	1734
AB	1752	1762
G	1774	1782
FS	1792	1808
D	1815	1930
F	1928	1934
EB	1932	2062
D	2059	2188
DB	2183	2446
C	2491	2628



COMPUTERS IN MUSIC RESEARCH

The STAN example: a fragment of the cor anglais solo in the Prelude to Act III of Wagner's *Tristan und Isolde*.



Note	Onset	Offset
START	148	190
G	280	287
F	302	309
EB	322	329
BB	347	466
G	474	518
EB	538	548
D	559	566
CS	578	586
A	605	648
FS	646	657
D	669	678
CS	687	696
C	707	714
AB	729	760
F	769	777
DB	791	801
C	811	820
B	830	839
G	856	987

PARSING THE PARSER

EB	986	1027
C	1049	1054
B	1068	1075
BB	1087	1096
F	1111	1153
D	1152	1157
BB	1174	1183
A	1194	1202
AB	1211	1220
EB	1232	1270
C	1272	1279
AB	1295	1304
G	1316	1325
FS	1336	1348
D	1360	1619

The CLICHE Example.



Note	Onset	Offset
START	154	227
C	285	294
G	322	327
G	336	341
AS	349	383
G	384	407
B	445	453
C	484	527The

# Autocorrelation and the Study of Musical Expression

Peter Desain and Siebe de Vos

Music Department  
City University  
Northampton Square  
GB-London EC1V 0HB

Centre for Knowledge Technology  
Utrecht School of the Arts  
Lange Viestraat 2B  
NL-3511 BK Utrecht

**ABSTRACT:** In performances musical structure is conveyed as variations of timing and other parameters. A method was designed to analyse these variations using autocorrelation. Peaks in the autocorrelation function are interpreted as periods of repeated components in the musical structure. Care has to be taken in using the standard autocorrelation function in this domain. Partial autocorrelation was used to remove the multiples of a fundamental period.

## Introduction

In musical performances the performer uses variations of timing, dynamics and articulation. An often posed hypothesis is that these expressive variations are closely linked to—and intentionally convey properties of—the musical structure, i.e., the performers interpretation of it [Clarke 1987]. In a research project on expressive timing we designed a set of tools, called POCO, to analyse, modify and generate musical expression [Honing 1990]. One of the tools analyses expressive data using autocorrelation in order to find regularities which, according to the hypothesis above, correspond to musical structure. In this article we focus on expressive timing, but other expressive parameters can be dealt with using the same method.

Any deviation from a strict metronomical performance is regarded as expressive timing. We assume that it mainly stems from a multiplicative combination of tempo factors at several structural levels and the exact metrical note durations in a the score. By dividing the durations observed in the performance (the inter-onset intervals) by the durations in the score a measure of local tempo is obtained. This function from time (onset-time in the score) to relative duration, the expressive timing signal, the logarithm of which can depends linearly on the components.

As an example we will use Bach's C major prelude (WTC I), which was the subject of many other studies [Cook 1987, Lehrdahl and Jackendoff 1983, Povel 1977]. All notes in this piece are of equal duration. The main structural units are half-bar, bar (16 notes) and 2 bars, at higher levels the metrical grouping is not trivial. Performers generally exhibit an amazing consistency in expressive timing over performances. The expressive timing of one of the performances is shown in Figure 1. Todd's approach in analysing this kind of timing data is to look at the local maxima [Todd 1985]. They indicate a slowing down and Todd's analysis relates the relative height of these peaks directly to the structural boundary strength. Although it is not so difficult to spot obvious phrase-final lengthening, it is unlikely that a robust classification of peaks into structural levels can be made. Therefore we looked for more global methods to detect structure.

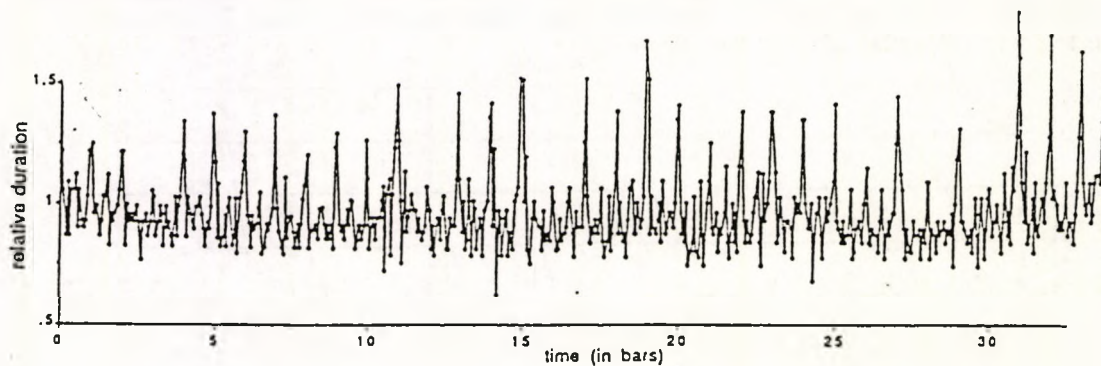


Figure 1. Expressive timing of the Bach Prelude

Regularity in musical structure will be reflected as periodicity in the expressive timing signal. We use autocorrelation as a statistical method to find periodicity, the periods found are interpreted as the lengths of structural components. We will assume here that musical structure is more or less homogeneous, at least for some time span and at some level.

### Autocorrelation

If a signal is periodic with a period  $P$ , it will resemble itself after an interval  $P$ . A well-known statistical measure of resemblance is correlation. By calculating correlations between a signal and the same signal delayed by different lags we obtain a series of values, the autocorrelation curve. When the signal contains a periodic component with period  $P$ , a peak in the autocorrelation curve occurs at this value. Considering our domain we have to be careful in the use of the standard autocorrelation [Bowermann 1979, Priestley 1981]. The function must depend on time to be able to show changes in periodic structure. We realise this by placing a window on the samples. Then the autocorrelation at  $t$  is the autocorrelation in the window that ends in  $t$ , or  $X(t-W+1) \dots X(t)$ , where  $W$  is the window size and  $X$  the signal. The window size should depend on the lag, otherwise a change in the level of a component with a small period will go unnoticed since there are still many 'old' periods contained in the window. We choose a window proportional to the lag, in the examples we used a factor  $p=4$ . A second reason to use relatively small windows is that the signal cannot be assumed to be stationary, which means that its statistical properties like mean and variance change over time. But using small time intervals the error introduced may be neglected. This leads to the following definitions of mean and autocovariance:

$$\bar{X}_{w,t} = \frac{1}{W} \sum_{i=0}^{W-1} X(t-i),$$

$$R_{w,t}(r) = \frac{1}{W} \sum_{i=0}^{W-r-1} (X(t-i) - \bar{X}_{w,t})(X(t-r-i) - \bar{X}_{w,t}).$$

The factor  $1/W$  instead of  $1/W-r$  corrects the values for greater lags which are calculated with only a fraction of the samples, as in the commonly used biased autocovariance estimator [Priestley 1981]. The autocorrelation is defined in terms of the autocovariance as:

$$\rho_{w,t}(r) = \frac{R_{w,t}(r)}{R_{w,t}(0)}$$

and the time dependent autocorrelation function with proportional window size is given by:

$$\rho(t,r) = \rho_{pr,t}(r)$$

Figure 2 shows the autocorrelation of the signal in Figure 1. Note the prominent peaks at the lags corresponding to the length of metrical units. We found these peaks only in data of expert performers, showing their ability to produce consistent timing patterns. Note also that the autocorrelation definition used is not very meaningful in the smaller lags, because it depends there on a very small number of measurements.

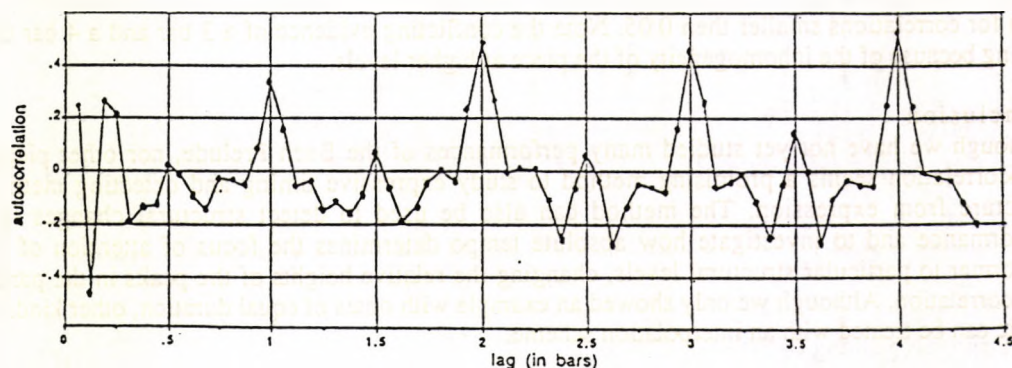


Figure 2. Autocorrelation of the data of Figure 1 (beginning of bar 31,  $p=4$ ).



### Partial autocorrelation

A problem occurs in interpreting the autocorrelation curve. When a signal repeats itself after period  $P$ , it will also be the same after period  $2P$ ,  $3P$ , etc. To detect if there is additional regularity at these levels over and above the regularity originating from their 'fundamental',  $P$ , we use partial autocorrelation. Partial correlation determines the correlation between two variables, cancelling out the influence of other variables on both of them. In the case of autocorrelation it removes the effect of smaller periodicities on the autocorrelation for a certain lag. The partial autocorrelation at lag  $r$ ,  $\rho(r,r)$ , is defined as [Bowerman 1979]:

$$\rho(1,1) = \rho(1)$$

$$\rho(k,k) = \frac{\rho(k) - \sum_{j=1}^{k-1} \rho(k-1,j)\rho(k-j)}{1 - \sum_{j=1}^{k-1} \rho(k-1,j)\rho(j)} \quad \text{if } k > 1,$$

$$\rho(k,j) = \rho(k-1,j) - \rho(k,k)\rho(k-1,k-j).$$

This formula depends on a statistically sound autocorrelation function. We cannot use the modified autocorrelation directly, but it is possible to retain the dependence on time and lag when we recalculate the autocorrelation function for each lag of the partial autocorrelation:

$$\rho(t,r,r) = \rho(r,r) \text{ where } \rho(k) = \rho_{p,r,t}(k).$$

The advantage of partial autocorrelation is seen in Figure 3: e.g. the peak at the 3.5 bar lag in figure 2, which arose only because it is a multiple of the half bar length vanishes in the partial autocorrelation.

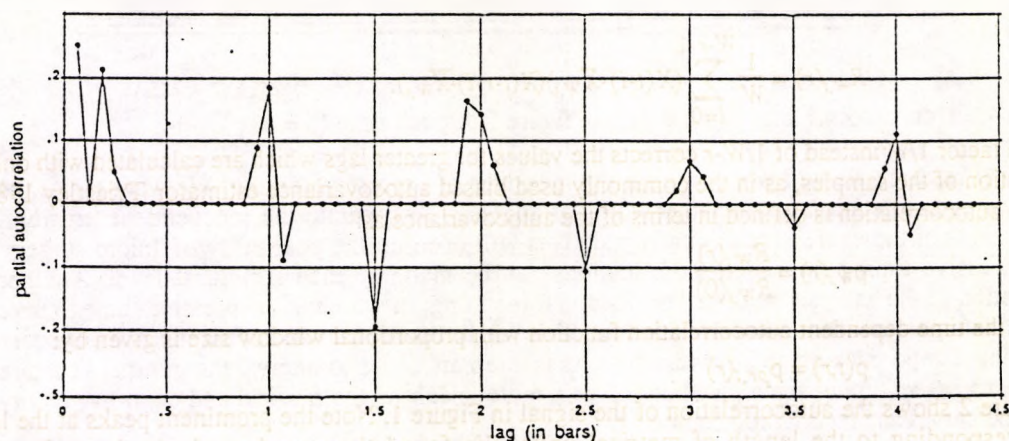


Figure 3. Partial autocorrelation of the data of figure 1 (beginning of bar 31,  $p = 4$ ).

Making use of the time dependency of the analysis we can show the relative stability of metrical units of 2 bars or smaller, throughout the piece (Figure 4). In this picture the data is truncated to zero for correlations smaller than 0.05. Note the conflicting evidence of a 3 bar and a 4 bar unit arising because of the inhomogeneity of the piece at higher levels.

### Conclusion

Although we have not yet studied many performances of the Bach Prelude, nor other pieces, autocorrelation seems a promising method to study expressive timing and detecting metrical structure from expression. The method can also be used to detect structural changes in a performance and to investigate how absolute tempo determines the focus of attention of the performer to particular structural levels, changing the relative heights of the peaks in the partial autocorrelation. Although we only showed an example with notes of equal duration, other kinds of music can be treated with an interpolation scheme.



However, the method has severe intrinsic limitations. It is based on the assumption that the expressive components at each structural level are more or less stable and periodic and it assumes independent combination of the components, an assumption that clearly limits the applicability of this method. Furthermore, no phase information is retained, statistical reliability is questionable for small windows and we can use the method only for generating hypothesis about the structure of the piece, not for testing them statistically.

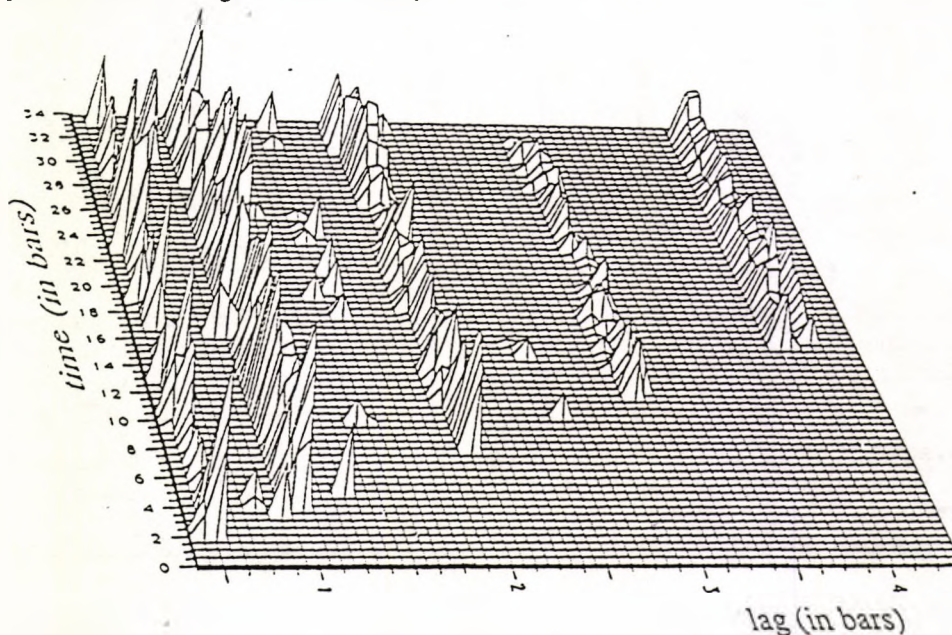


Figure 4. Partial autocorrelation of the data of figure 1 through time ( $p = 4$ ).

In further research we will work on different measures of expressive timing. We want to use the result of autocorrelation (and another kinds of analysis) to separate the independent structural components of the expressive timing signal. This will open up the possibility of 'micro surgery' on expressive timing, in which modifications can be made at each structural level. Another application of the analysis method described is forecasting, in which the expressive timing curve can be extrapolated from a known segment. This might result in more robust methods of score following, tempo tracking and quantization. As this research tries to unravel the internal structure of the expressive timing signal, we hope to gain more insight in the musical and cognitive reality of tempo curves and their representation.

#### Acknowledgements

We would like to thank Eric Clarke, Henkjan Honing, Steve McAdams, Klaus de Rijk, and Luc van Vugt for their help and we are especially grateful to Chris Mould for his performances.

#### References

- Bowerman B.L. and R.T. O'Connell. 1979 "Time Series and Forecasting, an Applied Approach" Boston, MA: PWS .
- Clarke, E.F. 1987 "Levels of Structure in the Organisation of Musical Time" in: Music and Psychology, a Mutual Regard. S. McAdams(Ed.) Contemporary Music Review 2(1).
- Cook, N. 1987 "Structure and Performance Timing in Bachs C major Prelude (WTC 1): an empirical Study" Music Analysis 6(3).
- Honing, H. 1990 "POCO, An Environment for Analysing, Modifying and Generating Expression in Music." ICMC '90, San Fransisco: CMA.
- Lehrdahl, F. and R. Jackendoff. 1983 "A Generative Theory of Tonal Music" Cambridge, Mass: MIT Press.
- Povel, D. 1977 "Temporal Structure of Performed Music: Some Preliminary Observations" Acta Psychologica, Vol 41.
- Priestley, M.B. 1981 "Spectral Analysis and Time Series" London: Academic Press.
- Todd, N.P. 1985 "A Model of Expressive Timing in Tonal Music" Music Perception 3(1).

# TEMPO CURVES CONSIDERED HARMFUL

*Peter Desain & Henkjan Honing*

MARCH 1991

EDITED MAY 1991

Will be published as: Desain, P. & H. Honing. (1991). Tempo curves considered harmful. In "Music and Time", edited by J.D. Kramer. *Contemporary Music Review*. London: Harwood Press.

© copyright 1991, Peter Desain & Henkjan Honing

Center for Knowledge Technology  
Lange Viestraat 2b  
3511 BK Utrecht  
The Netherlands

## CONTENTS

Abstract.....	3
Keywords .....	3
In which we decided to have a good time, invited an expert, and had our first disappointment.....	4
<b>Tempo, Metre and Beat</b> .....	6
<b>Tempo, Timing and Structure</b> .....	9
Wherein we looked at multiple performances, learned from a conductor and tried different hierarchies but had no success. ....	9
<b>Timing and Tempo, Patterns and Curves</b> .....	11
<b>Generative models</b> .....	15
In which we investigated discrete patterns and continuous curves, tried interpolation and failed again .....	15
<b>Subjective Time, Duration and Tempo Magnitudes</b> .....	16
<b>Objective Time, Duration and Tempo Measurements</b> .....	20
Epilogue .....	20
Acknowledgements.....	22
References.....	22

## ABSTRACT

In the literature of musicology, computer music research and the psychology of music, timing or tempo measurements are mostly presented in the form of continuous curves. The notion of these tempo curves is dangerous, despite its widespread use, because it lulls its users into the false impression that a continuous concept of temporal flow has an independent existence, a musical or psychological reality, and that time can be perceived independent of events carrying it. But if one bases a transformation or manipulation of timing on the implied characteristics of such a notion, one is doomed to fail.

## KEYWORDS

representation of time, tempo curves, expressive timing



# TEMPO CURVES CONSIDERED HARMFUL

*Peter Desain & Henkjan Honing*

**In which we decided to have a good time, invited an expert, and had our first disappointment.**

Not so long ago we decided to spend a Christmas holiday studying music and its performance. One of us is an amateur mathematician (M) and the other one likes to delve into old psychology textbooks (P), and because we enjoy impressing each other with new facts and insights, we often find ourselves in vehement discussions. Therefore we thought we might have a pleasant and peaceful time by putting our beloved hobby horses aside and embark upon a subject about which neither of us knew much: the timing aspects of music. We became interested in this field because we had noticed, while playing with the computer, our favourite toy, that adding just a bit of random timing noise to a program that played a score in an otherwise metronomically perfect way, made the music much more pleasant to listen to. It seemed as if we could make more sense of it. But we suspected that there was more to timing and expressive performance than adding bits of noise, so we invited a mutual friend who is a retired professional pianist to spend Christmas in our small but well equipped laboratory. Our friend has a great love for the piano and its music, but is completely ignorant of the advances of modern technology. To demonstrate to him our latest sequencer program we asked him to play the theme from the six variations composed by Ludwig van Beethoven on the duet *Nel cor più non mi sento*, the score of which we had lying around (see Figure 1).



Figure 1. Score of the theme of *Nel cor più non mi sento*.



Even though he was somewhat disturbed by the touch and harpsichord-like sound of the electronic piano, he was quite fascinated with the possibility of recording and playing back on the same instrument. Enthusiastically we told him that this system was more than just a modern version of the pianola: 'You can examine and change every detail you want; for instance, inspect the timing, accurately to the millisecond, add and remove notes, make notes longer or shorter, or louder or softer, and so on and so forth.' Our friend became quite excited and asked: 'Could your machine play my performance in a minor key?' We were a bit put off by the simplicity of his demand, but patiently demonstrated the key-change feature. After hearing his performance with the key changed to G minor our friend was not impressed. 'O dear, I'm afraid this sounds much too hasty. For example, the "dramatic" e-flat in bar 3 needs more time. Let me play it in minor for you.' When we looked at the timing data of his new performance it indeed showed a different pattern. Upon noticing our disappointed faces our friend remarked 'this was not a minor change; it really turns it into another piece. We did not expect your device to know about that, did we?' We kept silent. 'But your machine can undoubtedly play the same piece at a faster tempo.' That set us in motion again. We changed the setting of the tempo knob to a tempo one-and-a-half times as high and pushed the play button. The face of our friend again did not show the expression we had hoped for. 'I'm awfully sorry, but this is not right! It sounds like a gramophone record played at the wrong speed, but without changing the pitches.' Suspiciously, we wanted some proof for his crude statement and asked him to play it the way he thought it ought to be performed. His version at the higher tempo was indeed different. We had to admit that it sounded more natural than our artificially speeded-up version. What made it sound so much better? We tried to unravel this mystery by examining the timing of the onsets and the offsets of the notes, since these were the variables that could be altered with our electronic keyboard, just like a real harpsichord.

## Tempo, Metre and Beat

*Temporal pattern* is a series of time intervals, without any interpretation or structure.

*Rhythm* is a temporal pattern with durational and accentual relationships and possibly structural interpretations (Dowling & Harwood, 1986).

*Beat* refers to a perceived pulse marking off equal durational units (Dowling & Harwood, 1986, p. 185). They set the most basic level of metrical organisation. The interval between beats is sometimes called a "time-span" (Lerdahl & Jackendoff, 1983), or, less abstract, beat duration, beat period or metrical unit (Longuet-Higgins & Lisle, 1989).

*Metre* involves a ratio relationship between at least two time levels (Yeston, 1976). One is a referent time level, the beat period, and the other is a higher order period based on a fixed number of beat periods, the measure. It imposes an accent structure on beats, because beats initiating higher level boundaries are considered more important.

*Tempo* refers to the rate at which beats occur (often expressed as beats per minute), and is therefore closely linked to the metrical structure.

*Density* is used to refer to the average presentation rate taken across events of different duration (i.e. events per second) when a piece has events of different durations and the beat is hard to determine unambiguously, if at all (Dowling & Harwood, 1986).

It is important to note that rhythm, tempo, metre and density can be conceived independently: it is possible to maintain the same tempo while changing density; for example, a musical fragment can have a lot of embellishments (i.e. have a high density) and still be perceived as having a slow tempo. Furthermore, rhythm can exist without a regular metre and any type of rhythmical grouping can occur in any type of metrical structure (Cooper & Meyer, 1960).

*Tactus* is the tempo expressed at the level at which the units (beats) pass at a moderate rate (Lerdahl & Jackendoff, 1983). This rate is around the "preferred" or "spontaneous" tempo of about 100 beats per minute (Fraisse, 1982).

Our sequencer, a very recent version, had a separate tempo track. In this track, the tempo can be changed from fragment to fragment, even from note to note. With this feature we could put the original score on one track and the timing of the performance, expressed as tempo changes per note, on the tempo track, although it took quite a bit of calculating and editing by hand. After a while we had completely recreated the original performance, but now as a score plus a separate track of expressive timing

information. This tempo track looked like the graph in Figure 2a (for clarity we show only the timing of the melody). We could now compare the timing of this performance with the one played at tempo 90 (see Figure 2b). Their form was quite different even by visual inspection, although our ears were, of course, the only valid judges.

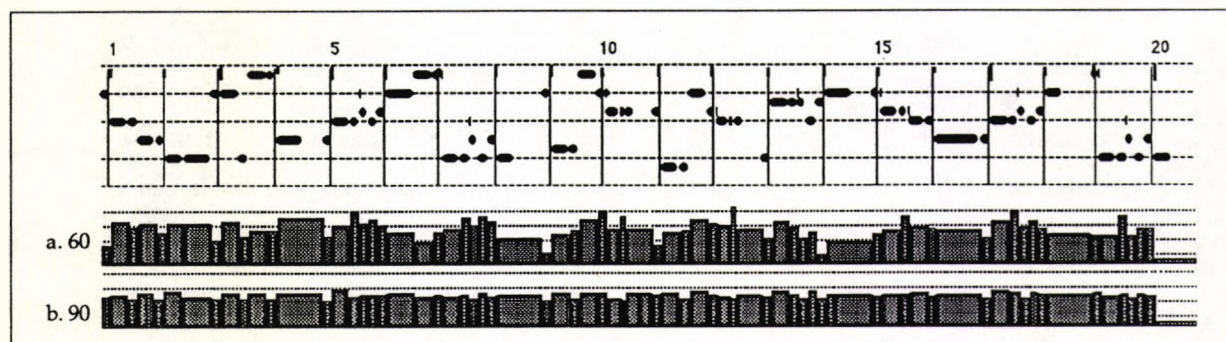


Figure 2. Tempo deviations in the performance of the theme at tempo 60 (a) and at tempo 90 (b).

What had happened? The sequencer had speeded everything up by the same amount (which we all agreed sounded awkward), while in the performance the expressive timing appears not to scale up everywhere by the same factor. Our friend adapted his rubato according to the tempo, which he explained to us as: 'My timing is very much linked to the musical structure and what I want to communicate of it in an artistic manner to the listener. If I play the piece at another tempo, other structural levels become more important; for instance, at a lower tempo the tactus will shift to a lower level, the subdivisions of the beat will get more "in focus", so to say, and my phrasing will have much more detail.' After some scratching with pen on paper, M found a quite elegant way of representing these changes using simple mathematics. We took the time interval between the onsets of every two succeeding notes and calculated the ratios of these time intervals in the two tempi. If the expressive timing pattern would scale-up linearly, we would find the ratios for all the notes to be around the ratio between the two tempi, and most ratios were indeed around 1.5. There was some variance around that factor, though, and we thought that could be explained by the more elaborate short-span phrasing at the lower tempo. But, even more noticeable was the fact that for some notes the ratio was close to 1. We found that these notes were notated as grace notes in the score. They did not change at all when performed at an another tempo. We also found that not all grace notes behaved like this. For example, the two grace notes that cover an interval of a sixth, in bar 7 and 19, were timed like any other note: they were actually played in a metrical way. Our pianist got really excited about our observations. He pointed at grace notes in the score that were notated in the same way,



but that needed a different interpretation, and he started to lecture about the different kinds of ornaments, so popular in the eighteenth century, the difference between *acciaccatura* and *appoggiatura*, 'ornaments that "crush in" or "lean on" notes', about their possible harmonic or melodic function changing their performance, and so on and so forth. When he noticed that we were getting bored with his lengthy historical observations, he woke us up again with a new, sharp attack on our beautiful sequencer program: 'It might be forgivable that your program cannot play the onsets of ornaments correctly, but it also murders the articulation of most notes, especially the staccato ones. And have you heard what the program did to my detailed colouring of the timbre of chords?' Well, in fact, we had not, but we could well understand that the timbral aspect brought about by the chord spread (playing some notes in a chord a tiny bit earlier or later than others) was not kept intact when all timing information is just scaled by a certain factor. And we did not even dare to play the performance again at a lower tempo, afraid that each chord would turn into an arpeggio.

So our sequencer was not so wonderful after all. It could not be used to *change* something, not even such a minor thing as the key in which the piece was played. Again our pianist explained that a change of key was not a minor thing. The minimal variation that he could think of was the repetition of bars 5-8 at the end of the theme. 'The only difference between them is the fact that the second segment is a repetition of the first, and I even expressed that minimal aspect by timing. This problem is exacerbated if the difference between two sections is the overall tempo. Then detailed knowledge about structural levels, articulation, timing of ornamentations and chords, is indispensable.' We had to agree. How dumb of us, after all, to assume that a tempo knob on a commercial sequencer package could be used to adjust the tempo.

## Tempo, Timing and Structure

In principle, timing can be linked to any musical structural concept. The most concrete of those are the following.

Although the most obvious *metrical units* are bar and beat, this strictly hierarchical structure may extend above and below these levels. Special expressive marking of the first beat in the bar, either by timing, dynamics or articulation, is a common phenomenon (Sloboda, 1983).

*Phrases* may not be ordered in a strict hierarchy, and may cut across metrical structure. Phrase final lengthening is the most well-known way in which they are treated (Todd, 1989)

A large proportion of the timing variance can be attributed to *rhythmical groups* (Drake & Palmer, 1990). Some standard rhythmical patterns, like triplets, seem to have a preferred timing profile (Vos & Handel, 1987).

Small timing asynchronies within a *chord* (called chord spread) are perceived as an overall timbral effect - the actual timing pattern is hard to perceive.

*Ornaments*, like *grace notes* and *trills*, can be divided in *acciaccatura*, so called timeless ornaments, and *appoggiatura*, ornaments that take time and can have a relatively important harmonic or melodic function. The former normally falls outside the metrical framework, the latter tends to get performed in a metrical way.

The independent timing of individual *voices* is sometimes hard to perceive because their components are immediately organised by the perceptual system in different streams (Bregman, 1990). This is not the case with (almost) simultaneous onsets that result in clear timbral differences. This can be heard in ensemble playing where often the leading voice takes a small lead of around 10 ms. (Rasch, 1979).

Any *associative relation*, e.g. between a musical fragment and its repetition, can be given intentional expression by using the same or different timing patterns.

## Wherein we looked at multiple performances, learned from a conductor and tried different hierarchies but had no success.

But we were convinced we could make our friend happy, and proposed to program some additions to the sequencer ourselves. We showed him a video tape about research done at MIT by Barry Vercoe and his collaborators on computer accompaniment of a real musician. In this project the computer is given a score and several performances of the piece. With that information it can be "trained" to follow and accompany the musician.



Not that we were trying to do that, but we could use the idea to annotate each note in the score with its deviation in the performance, in our case in different tempi. Our friend friendly agreed to perform the Beethoven theme at four different tempi that were musically acceptable to him. We saw again that some notes exhibited a large change when tempo is changed, while others were less influenced by the tempo. But we could now use statistical methods to derive the right timing information for each tempo from this data. Our friend, who started to develop a little bit of suspicion, asked: 'Will that solve playing at different tempi then?' We were not quite sure. We definitely had more information now, but the representation of the music was still flat; no structural information was provided. It seemed we could not avoid incorporating some organisation above the note level into our program. Our friend agreed with a smile that was almost saying: 'are you stupid or am I?' We got a bit nervous. But after some discussion he agreed to concentrate on the timing of simple structural units like beats and bars only, leaving the note by note details aside for the moment.

Then we remembered Max Mathews working at CCRMA, Stanford University, who does important work in conductor systems (sort of the opposite of what Vercoe is doing). He made a system where one can conduct a sequencer on the beat level, which was just what we needed. The idea of a conductor shook our friend up; that sounded a much better approach than all those statistics we tried to explain to him before. We gave our friend an electronic baton, connected to our sequencer, and asked him to conduct the piece. In the score in the sequencer the beats were marked. The program followed the conductor by aligning each conducted beat with the corresponding mark in the score, and it tracked the tempo indicated by the conductor in doing so. At the high tempo, beating the baton very quickly, it seemed all right, but at the moderate tempo it was impossible to steer the timing deviations within the beat. 'It sounds too jumpy,' our friend complained. Since the beat level of the system of Mathews is arbitrary (he calls it 'generalised'), we annotated the score with marks at a lower metrical level, which alleviated the problem a bit. But, as our friend was still complaining about the controlability, we eventually ended up by marking each note in the score. This gave complete control at last, though our poor pianist, out of breath by the acrobatics needed to draw each note out of the sequencer by means of a single baton, made a cynical remark about the wonderful invention, which we may have heard of, called a keyboard. We became a bit vapid and proposed to help our conductor by connecting three MIDI batons to the computer, the first two used by us to time the bars and the beats, and the third to be used by our friend to fill in the details, using batons inter-connected with a complex mechanism of wires, to keep the timing at all levels consistent. We fantasized for some

time about a whole orchestra of conductors, leading one pianist before them. It was clearly time for a tea break.

### Timing and Tempo, Patterns and Curves

In studying *timing deviations* a first distinction should be made between non-intended *motor noise* and intended *expressive timing* or *rubato*. The first category deviates in the range of 10 to 100 ms; the latter can deviate up to 50% of the notated metrical duration in the score.

Expressive timing is continuously variable and reproducible (Shaffer, Clarke & Todd, 1985) and clearly related to structure (Clarke, 1988; Palmer, 1989).

It is important to note that there is interaction between timing and the other *expressive parameters* (like articulation, dynamics, intonation and timbre). For example, a note might be accented by playing it louder, a fraction earlier than expected or by lengthening its sounding duration. Which method of accentuation is used is difficult to perceive, even when the accentuation itself is obvious.

To refer to expressive timing, in computer music the term *micro tempo* is often used, comparable to the term *local tempo* used in the psychology of music (the tempo changes from event to event, expressed as a ratio of a performance time interval and a score time interval). For clarity, the term *timing* would be more appropriate here. It specifies the timing deviation on a note-to-note basis and is often referred to as the *expressive timing profile* (Clarke, 1985; Shaffer, 1981; Sloboda, 1983), *timing pattern* or *rubato pattern* (Palmer, 1989).

In these patterns, points are often connected, either stepwise with straight line segments or with a smooth interpolation, yielding a *timing curve*. Only the first representation maintains a proper relation with the time map in which points are connected with line segments. These continuous time maps are used by Jaffe (1985) and most people of the computer music community. Time maps can be superimposed, using one for each voice.

Time maps can also be constructed for uniformly spaced units in the score like bars or beats. The corresponding duration patterns form a true *tempo pattern*. The points in these patterns can be connected by line segments, yielding so called *tempo curves*. Some authors insist on stepwise tempo changes, like Mathews (Boulanger, 1990), in which they are linked to one level of the metrical structure.



Over tea our friend told us about a series of programs on BBC radio, presented by the English conductor Denis Vaughan, on the composer's pulse he used in conducting. The pulse is a hierarchical, composer specific way of timing the beats. This pulse was an idea proposed and actually programmed by someone working in Australia. We went to our library and looked for some references that might tell us more on this composer's pulse. We ran into a collection of articles by Manfred Clynes who had invented the notion. This pulse, coincidentally, had precisely the characteristics we were looking for: hierarchical tempo patterns linked to the metrical structure. It basically entailed a system of automated hierarchical batons, and reduced the complexity further by postulating a fixed pattern for each baton. We took a final sip of our tea and hurried back to the lab and added Clynes' Beethoven 6/8 pulse as tempo changes in the tempo track to our sequencer. It divided the time for each bar into two unequal time intervals for the first and second half-bar and divided each half-bar into 3 unequal parts, one for each beat. With some adjustments here and there, we had our program running in no time. We called in our musical friend from the library to provide some professional judgements. He was definitely not unhappy with the result. 'This sounds much better than the things I've heard before,' he said.



Figure 3. Score of the first variation of *Nel cor più non mi sento*.

'Let's do the first variation, and see how our system performs it,' our friend said, far more optimistic now. He was talking about "our" system. This was a good sign. 'This variation is written in an ornamental style,' our friend explained, while we loaded the score of the first variation (Figure 3) into our system and created the tempo track containing the Beethoven pulse for this material. 'The metrical and harmonic structure is the same for both theme and the first variation. The only difference is that there are more "ornamental" notes added,' he said in a patronizing tone. When everything was set we played him the result. 'Well, this is disappointing,' was his short and decisive answer. After seconds of uncomfortable silence he added, 'it lacks the general phrasing and detailed subtlety I think is essential to make it an acceptable performance. The rhythmical materials of the theme and the first variation are different. The sixteenth notes of the variation ask for a different kind of timing than the mainly short-long, short-long, short-long rhythm of the theme. This pulse plays only with the metrical structure, but musical structure has far more to offer than that.' So the composer's pulse could not just be mapped onto any rhythmic material. Furthermore, it only linked timing to the meter, and, as our friend made clear, phrasing and other musical structure was ignored.

That rang a bell. We remembered one of the articles by Neil Todd on a model of rubato, linked to phrase structure. His proposal is very similar to Clynes; it explains timing in terms of a hierarchical structure, but now phrase structure is the basic ingredient. The beat is again the lowest level; below that no timing is modelled. The abundance of mathematical notation in Todd's articles did not put off our amateur mathematician. Quite the contrary. 'This, on first sight, will give us a solid basis to work with. What he states here is that, if you remove all the constants from the formula, it is actually quite simple,' M said. 'Todd proposes to attach a parabola to each level of the hierarchical phrase structure, and sum their values to calculate the beat length.' He simplified a formula, found an error on the way and finally the model became easy to implement. We were quite conscious of the fact that we were the first really to hear Todd's model (he himself had never listened to it). It did not sound very pleasing because this model was expressed in terms of the phrase structure only (based on the idea of systematically lengthening the end of a phrase in a hierarchical way), and because it lacked all expressive timing below the level of beats.

Longing to show our collaborator that the computer could, in principle, also calculate detailed note-by-note timing, we looked for a model that would provide these. Happily we found masses of rules for those subtle nuances in the articles of Johan Sundberg and his colleagues. These rules formulated simple actions, like inserting a



small pause in between two notes or shortening a note. The actions had to be performed if the notes matched a certain pattern, such as constituting a pitch leap or forming part of a run of notes of equal duration. In fact there were so many rule sets proposed in his articles that we got a bit lost in the details, but it has to be said that some rule-cocktails really seemed to work for our piece. Especially if their influence was adjusted to effect a subtle change only, the music gained some liveliness. But because these rules are based on the surface structure of the music only we could predict the judgement of our musical expert by now. And indeed he did not even bother to comment on the artificially produced performances. Instead he kindly reminded us that we might give up looking for a system that enabled us to generate a "musically acceptable" performance, given a score (that is what Clynes, Todd and Sundberg are aiming at), for the simple reason that we already had an "acceptable" performance, namely his own. It was true, the initial aim of our endeavour was to find ways of manipulating the timing in a musically and perceptually plausible way, given a score *and* a performance. Because the simple representations we had used proved unsuccessful, we had been sidetracked by studying even simpler representations that could at most model a small aspect of our friend's performances. We decided to close the session, look for more details in the literature, and give it another try the next day.

## **Generative models**

Clynes (1983; 1987) proposes composer specific and metre specific, discrete tempo patterns. This so called composer's pulse is assumed to communicate the individual composer's personality. E.g. in the Beethoven 6/8 pulse the subsequent half-bars span 49 and 51% of the bar duration and each half bar is divided again in 35, 29 and 36%. Clynes is opposed to analysis of performance data: the pulses stem from his intuition. Repp (1990) has undertaken a careful evaluation of this model.

Todd (1985; 1989) proposes an additive model in which beat duration is calculated as a summation of parabola shaped curves, one for each level of hierarchical phrase structure. He complemented the model with an analysis method that calculates phrase structure from beat durations.

Sundberg et al. (1983; 1989) proposes a rule system to generate expression from a score based on surface structure. His research was done in an analysis-by-synthesis paradigm and captures expert intuition in the form of a large set of these rules. An example of a rule is "faster uphill": A duration of a note is shortened if it is preceded by a lower pitched note and followed by a higher pitched one. Van Oosten (1990) has undertaken a critical evaluation of this system.

## **In which we investigated discrete patterns and continuous curves, tried interpolation and failed again.**

We found all kinds of references in the literature and read a lot that evening. It was amazing to find how much work actually was done on a problem that we had thought was not a problem at all. We became a little bit more conscious of the whole thing. It looked as if P's hobby horse, psychology, had to be given a chance. He explained that the perception of time had been modelled postulating a certain (often exponential) relation between objective time and experienced time. But this research had all been done with impoverished stimulus material, often consisting of just one time interval marked-off with two clicks. 'Other research,' P added, 'found that duration judgment depends on the way the interval is filled with more or fewer events, so unfortunately these simple laws cannot be directly applied to more complex material like real music.' Even P was disappointed with the results of his beautiful science. 'But psychology has something to offer to us here', he spoke in a defensive tone. 'Take a look at all the articles that present timing or tempo measurements in the form of continuous curves instead of just a scattergram of measurements. These curves more or less imply an independent existence, apart from the rhythmic material where they were measured

from. But psychological research has shown that one cannot perceive timing without events carrying it.' He found this convincingly argued in an article by the psychologist James J. Gibson called "Events are perceivable but time is not". 'Can you imagine perceiving a rubato without any notes carrying it?' P asked. 'And vice versa: "filling up" time by adding an event between two measured points is problematic, isn't it?' There seemed to be no possible argument.

### Subjective Time, Duration and Tempo Magnitudes

Most psychophysical scales for time intervals are described by Stevens' Law, that relates the physical magnitude of a stimulus to its perceived magnitude as perceptual-time = a-constant.physical-time<sup>b-constant</sup>. The *b* value differs from one dimension to the other. For time duration *b* is commonly found to be 1.1, slightly over estimation of the interval. However, for intervals shorter than 500 ms it is found that *b* is around 0.5, the square root of its physical duration (Michon, 1975).

Humans seem to have a relatively poor ability for time discrimination of intervals presented without context. The just notable differences (JND) are in the range of 5-10% (Woodrow, 1951) with an optimum near 600 ms intervals. However, in the context of a steady beat, the JND's are around 3% with the same optimum interval (Povel, 1981).

Much research was done on the existence of a spontaneous tempo, preferred rate or natural pace (Fraisse, 1982). This tempo should occur as a preferred rate of spontaneous tapping, and material presented at that rate should be easy to perceive and remember. There is weak, but converging evidence for the existence of such a rate, again with intervals around 600 ms. There is no consistent evidence for physiological correlates like heart rate.

There has been quite some research done on the influence of different dimensions on time perception, mainly in the fifties. Evidence was found that, in general, the higher pitched the sound the longer the percept (Cohen et al., 1954), and the same holds for louder sounds (Hirsch et al, 1956). Evenly divided intervals seem longer than irregular divided ones (Ornstein 1969).

Time intervals shorter than 120 ms, preceded by a physically shorter neighbour time interval, are underestimated to such a remarkable degree that one can speak of an auditory illusion (Nakajima et al., 1989).

We decided to do the acid test using a feature of the sequencer program. In this program it was possible to copy tempo tracks from one piece to the other. We applied

the tempo track of the original performance of the theme (see Figure 2) to the score of the first variation. The result was poor; even we could hear that. The timing made sudden jumps, like a beginner sight-reading and hesitating at unexpected points because of a difficult note. The expressive timing pattern found in the theme did not "fit" the variation. Our friend's performance of the variation was much smoother and had gestures on a larger scale, as far as we were able to judge (Figure 4). Also, the other way around, taking the timing data from the variation and applying it to the score of the theme had the same awkward effect. It seemed impossible to just add or remove notes using these stepwise tempo curves. We felt stupid again for having assumed that the independence of tempo tracks in the sequencer made musical sense. But it made us look in the literature for alternatives.

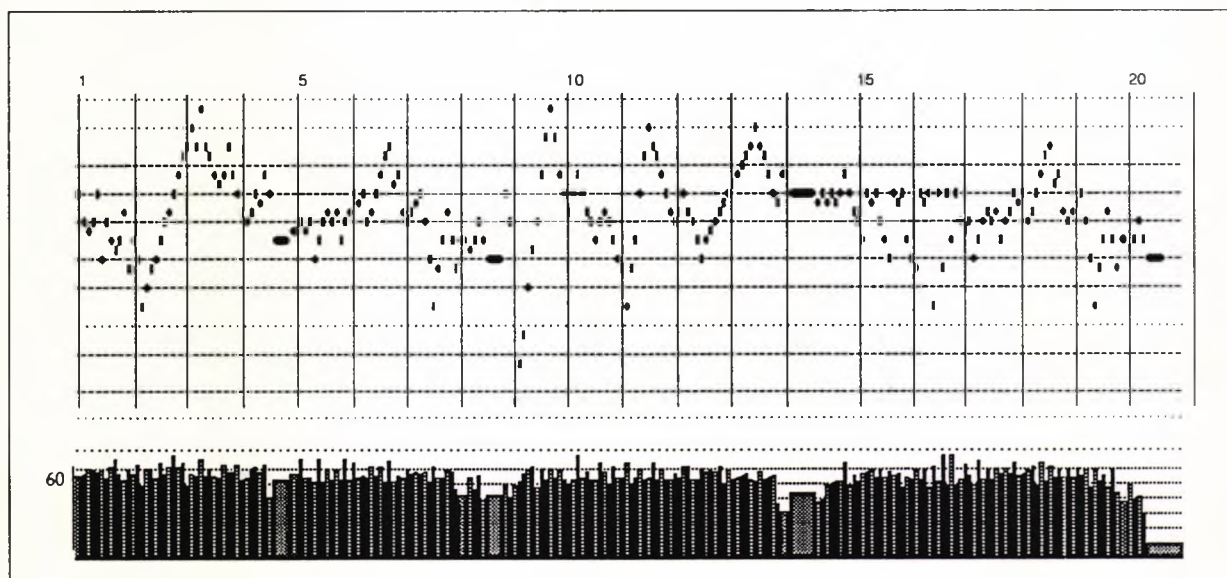


Figure 4. Tempo deviations in the performance of the variation at tempo 60.

The answer was not far away. In the field of computer music research continuous rubato curves were used almost by default. We decided to take the path of the continuous timing functions, hoping it would get rid of this awkward "jumpiness". Thus M's hobby horse was brought out again. 'Functions are far easier to handle. One can calculate, given the right kind of function, a good timing curve for every piece,' M argued convincingly. This combined approach of formality (in the mathematical sense) and pragmatics reminded us of a method developed by David Jaffe of CCRMA to model the timing of different parts of a computer orchestra. Jaffe wanted the different instruments to have their own timing, but they had to synchronise at specific points as well. By using a time map, instead of tempo changes, coordination and synchronization



became possible. 'What he actually does is to specify the timing for each event by means of a function from score time to performance time,' M explained, 'a blatantly simple idea indeed: to integrate velocity or one-over-tempo, as Jaffe calls it, to get time. This of course restrains the possible functions one can use to make up such a time map; they have to increase monotonously and one must be able to calculate a first derivative.' This was again a method, among many others, in which different authors presented their ideas of tempo curves (see Figure 5). We tried to bring some order to the ways the different representations were used.

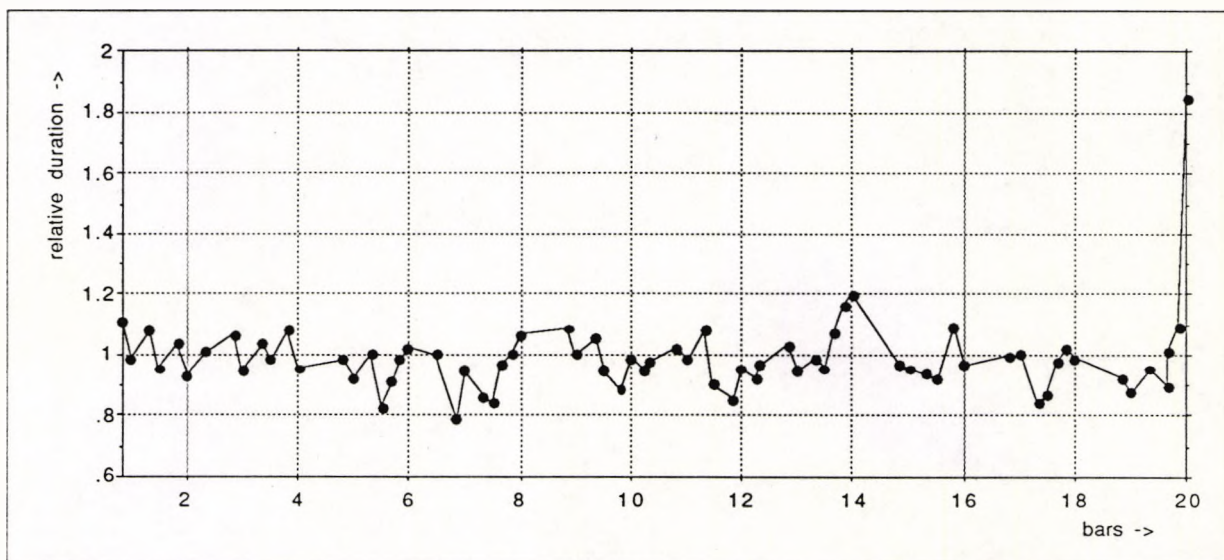


Figure 5. A typical so called "Tempo Curve" with duration factors for each note as a function of metrical time.

Soon M gave up, stating that it was a hopeless mess; no two authors used the same dependent and independent variables and measurement scales. And while in the end all the information needed could be extracted from most presentations, it was a difficult job, the more so because of the confusion in terminology. We decided to return to the practical application of the time map. We adapted the sequencer's tempo track to contain a time map (composed of line segments) instead of the discrete tempo changes we had used before. We then applied this continuous curve to the variation and had our pianist judge it. He thought it was much better than the direct application of the discrete curve of the theme to the variation. The interpolation (with line segments) did improve the smoothness of the timing, but he still complained about the sudden tempo jumps at the junctions of the curve. M remarked that one could restrict the allowed tempo map functions further or smooth the existing function, for instance, with splines. This brought us to an article describing work done at IRCAM by David Wessel and others, which indeed proposes the use of splines. We took an algorithm we had lying

around that did splines and added it to our tempo track algorithm. And there it was: with some twiddling of the parameters we could interpolate the timing pattern of the theme for its use on the variation. We almost thought that with this interpolation we had proven Gibson wrong. There was a smooth sense of timing in between events, and if one is smart enough one can tap it and hook new events onto it in a reasonable way. But our musical friend did not agree 'Reasonable?' he reacted angry, 'it sounds reasonable, yes, but your numerical calculations have nothing to do with the way I played it, whatsoever. The musical structure, my dear friends, remember the musical structure. How often do I have to repeat this. Timing *is* related to structure!' We suggested to him a cup of tea, in the hope that this would calm him down.

## Objective Time, Duration and Tempo Measurements

When an event happens (an onset of a note) one can measure the *real time* elapsed since the beginning of the piece (called *performance time*) and also the point in the score where this onset was notated (called *score time*). The latter can be measured either in seconds (taking the tempo marking in the score serious, or normalising the total score length to the performance), in metrical units like beats or quarter notes (called *metrical time*), or as an event count (called *event time*). The last loses so much information that the timing pattern cannot be reconstructed without reference to the score.

Performance time can be shown as a function of score time (called a *time map*), or vice versa. In these representations it is easy to spot (a)synchronies between voices because they depict points in absolute time.

Calculating differences between subsequent performance times in a time map makes the step from time to duration. Because in such a representation it is difficult to compare notes of different nominal duration, a proportional measure is better. It makes the step from duration to relative duration by dividing two corresponding durations. In case a performance duration is divided by a score duration, this forms a series of duration factors (often misleadingly called tempo). This measure is mostly notated in a graph with the independent axis labelled with metrical or event time. In the case of the inverse calculation, the ratios form the velocity, the local speed of reading the score.

In both cases the measured points are often filled in with line segments - implying the existence of a tempo measurement in between events. This is misleading - the more so because integration does not yield the original time map again.

Gabrielsson (1974) uses note duration expressed in proportion to the length of the bar. This allows for comparison with exact note values in different meters. The method might be generalizable to study timing at different levels of structure.

Tempo is sometimes presented on a logarithmic scale; this is a first step towards the use of subjective magnitudes.

An interesting hypothesis was given by Brown (1979). He argues that a musician makes use of a collection of discrete tempi: a collection of discrete physically possible tempi, where the choice is defined by musical and performing factors.

## EPILOGUE

What this partly fictitious story (the characters are fictitious, but the examples and arguments are real!) shows is that we have to be aware of the Tempo Curve. Of course



one should be encouraged to measure tempo curves and use them for the study of expressive timing. But it is a dangerous notion, despite its widespread use and comfortable description, because it lulls its users into the false impression that it has a musical and psychological reality. There is no abstract tempo curve in the music nor is there a mental tempo curve in the head of a performer or listener. And any transformation or manipulation based on the implied characteristics of such a notion is doomed to fail.

That does not mean that generic models that represent timing in terms of some sort of structure, even when they describe just a fraction of the many aspects of expressive timing, do not constitute a valuable contribution to the field. They only have to be seen in a proper perspective in which their limitations are understood as well. It also does not mean that certain features in computer music software and commercial sequencers should be forbidden. Their mere existence at least makes the realisation of their limited worth evident.

It should be noted here that the views expressed in this article comply more or less with the British school of expressive timing research (E.F. Clarke, H.C. Longuet-Higgins, L. Shaffer, J. Sloboda and N. Todd), in which the link between structure and timing is paramount. There are alternative views developing at the moment, denying such a strong link (Kendall & Carterette, 1991). We hope this controversy will eventually lead to more understanding of this wonderfully complex aspect of music performance.

In reality the experiments were done using POCO, an environment for analysing, manipulating and generating musical expression (Honing, 1990), which took a bit longer to build than one Christmas.

The holiday was almost over now and we felt that we had not found out many useful things. Our musical friend announced that he would go back to his own piano. He thanked us for the interesting sessions, from which he had learned a lot. But underneath these friendly remarks we could hear the cynicism. He advised us in a fatherly way to get rid of our research papers and start reading biographies of famous composers, in which the true facts about music and its performance could be found. This made the feeling of disappointment even more pronounced. But in a last irrational attack of bravery, we decided not to give in yet and we invited him to come back next Christmas, and to bring his biographies if he wished.

To be continued ...



## ACKNOWLEDGEMENTS

Thanks to Boy Honing and Mariken Zandvliet for their performances of Beethoven. Thanks to Bruno Repp for information on Clynes' model and to Shaun Stevens for his help with the English language. We would like to thank also all the researchers mentioned, for their contribution to the field of timing in music. We are very grateful to Eric Clarke who made it possible for us to work for two years on research in expressive timing which allowed us to gain an insight into timing through our numerous discussions, and the British ESRC for their financial support throughout these two years (grant A413254004).

## REFERENCES

- Boulanger, R. (1990) Conducting the MIDI Orchestra, Part 1: Interviews with Max Mathews, Barry Vercoe and Roger Dannenberg. Computer Music Journal 14(2).
- Bregman, A. (1990) Auditory Scene Analysis: the Perceptual Organisation of Sound. Cambridge, Mass: Bradford Books.
- Brown, P. (1979) An enquiry into the origins and nature of tempo behaviour. Psychology of Music 7(1).
- Clarke, E.F. (1985) Some Aspects of Rhythm and Expression in Performances of Erik Satie's "Gnossienne No.5". Music Perception, 2(3).
- Clarke, E.F. (1987) Levels of structure in the organisation of musical time. In "Music and psychology: a mutual regard", edited by S. McAdams. Contemporary Music Review, 2(1).
- Clarke, E.F. (1988) Generative principles in music performance. In Generative processes in music. The psychology of performance, improvisation and composition, edited by J. A. Sloboda. Oxford: Science Publications.
- Clynes, M. (1983) Expressive Microstructure in Music, linked to Living Qualities. In: Studies of Music Performance, edited by J.Sundberg . Stockholm: Royal Swedish Academy of Music, No. 39.
- Clynes, M. (1987) What can a musician learn about music performance from newly discovered microstructure principles (PM and PAS)? In Action and Perception in Rhythm and Music, edited by A. Gabrielsson. Royal Swedish Academy of Music, No. 55.
- Cohen, J., C.E.M. Hansell, & J.D. Sylvester. (1954) Interdependence of temporal and auditory judgements. Nature, 174 .

- Cooper, G. & Meyer, L. B. (1960) The rhythmic structure of music. Chicago: University of Chicago Press.
- Dowling, W. J. & D. L. Harwood (1986) Music Cognition. London: Academic Press.
- Drake, C. and C. Palmer (1990) Accent Structures in Music Performance. manuscript.
- Fraisse, P. (1982) Rhythm and Tempo. In The Psychology of Music, edited by D. Deutsch. New York: Academic Press.
- Gabrielsson, A. (1974) Performance of rhythm patterns. Scandinavian Journal of Psychology, 15.
- Gibson, J. J. (1975) Events are perceivable but time is not. In The Study of Time, 2, edited by J.T. Fraser & N. Lawrence. Berlin: Springer Verlag.
- Hirsch, I.J., R.C. Bilger & B.H. Deathrage (1956) The effect of auditory and visual background on apparent duration. American Journal of Psychology, 69.
- Honing, H. (1990) POCO, An Environment for Analysing, Modifying and Generating Expression in Music. In Proceedings of the 1990 International Computer Music Association. San Francisco: CMA.
- Jaffe, D. (1985) Ensemble timing in Computer Music. Computer Music Journal, 9(4).
- Kendall, R.A. and E.C. Carterette (1990) The Communication of Musical Expression. Music Perception, 8(2).
- Lerdahl, F. & R. Jackendoff (1983) A Generative Theory of Tonal Music. Cambridge, Mass.: MIT Press.
- Longuet-Higgins, H.C. & E. Lisle. (1989) Modelling musical cognition. In "Music, Mind and Structure", edited by E. Clarke and S. Emmerson. Contemporary Music Review 3(1).
- Michon, J.A. (1975) Time experience and memory processes. The Study of Time, 2, edited by J.T. Fraser & N. Lawrence. Berlin: Springer Verlag.
- Nakajima, Y., T. Sasaki, R.G.H. van der Wilk, & G. ten Hoopen. (1989) A new illusion in time perception. Proceedings of the First International Conference on Music Perception and Cognition. Kyoto, Japan: The Japanese Society of Music Perception and Cognition.
- Oosten, P. van (1990) A Critical Study of Sundbergs' Rules for Expression in the Performance of Melodies. Proceedings of the Music and the Cognitive Sciences Conference. Cambridge.
- Ornstein, R.E. (1969) On the Experience of time. London: Pinguin.
- Palmer, C. (1989) Mapping Musical thought to musical performance. Journal of Experimental Psychology, 15(12).

- Povel D.J. (1981) Internal Representation of Simple Temporal Patterns. Journal of Experimental Psychology: Human Perception and Production. 7(1).
- Rasch, R. A. (1979) Synchronization in Performed Ensemble Music. Acustica 43(2).
- Repp, B. (1990) Further Perceptual Evaluations of Pulse Microstructure in Computer Performances of Classical Piano Music. Music Perception. 8(1).
- Shaffer, L.H. (1981) Performances of Chopin, Bach and Bartok: Studies in motor programming. Cognitive Psychology. 35A.
- Shaffer, L.H., E.F. Clarke, & N.P. Todd (1985) Metre and rhythm in piano playing. Cognition. 20.
- Sloboda. J. (1983) The communication of musical metre in piano performance. Quarterly Journal of Experimental Psychology. 35.
- Sundberg, J., A. Askenfelt & L. Frydén (1983) Musical Performance: A synthesis-by-rule Approach. Computer Music Journal. 7(1)
- Sundberg, J., A. Friberg & L. Frydén (1989) Rules for Automated Performance of Ensemble Music. Contemporary Music Review. 3.
- Todd, N. (1989) A Computational Model of Rubato. In "Music, Mind and Structure", edited by E. Clarke and S. Emmerson. Contemporary Music Review 3(1).
- Todd, N.P. (1985) A model of expressive timing in tonal music. Music Perception. 3.
- Vos. P. & Handel, S., (1987) Playing Triplets: Facts and Preferences. In: Action and Perception in Rhythm and Music. Edited by A. Gabrielsson Royal Swedish Academy of Music. No. 55.
- Woodrow, H. (1951) Time Perception. In Handbook of Experimental Psychology. edited by S.S. Stevens. New York: Wiley.
- Yeston, M (1976). The stratification of musical rhythm. New Haven CT: Yale University Press.

TOWARDS A CALCULUS  
FOR EXPRESSIVE TIMING IN MUSIC

*Peter Desain & Henkjan Honing*

JULY 1991

*Submitted to Psychology of Music*

© copyright 1991, Peter Desain & Henkjan Honing

Center for Knowledge Technology

Lange Viestraat 2b

3511 BK Utrecht

The Netherlands



## CONTENTS

Introduction.....	1
Overview of the calculus.....	3
Characteristics.....	3
Representation.....	5
Implementation.....	6
Musical objects.....	8
Basic musical objects.....	8
Structured musical objects.....	9
Multilateral structures.....	9
Collateral structures (ornamented objects).....	10
<b>S, a multilateral successive structure.....</b>	<b>11</b>
<b>P, a multilateral simultaneous structure.....</b>	<b>12</b>
<b>APPOG, a collateral successive structure.....</b>	<b>13</b>
<b>ACCIA, a collateral simultaneous structure.....</b>	<b>14</b>
Example of the representation of a musical object.....	15
Representing expression.....	16
Expressive tempo.....	16
Expressive asynchrony.....	16
Expressive articulation.....	17
<b>Definition of articulation.....</b>	<b>17</b>
Estimate onsets.....	18
Articulation invariance.....	18
Expression maps.....	18
Onset timing.....	18
Articulation expression.....	20
Operations on expression maps.....	21
Scale maps.....	21
Scaling expressive tempo.....	21
Scaling the expressive tempo of an S section.....	22
Scaling the expressive tempo of an APPOG section.....	24
Scaling expressive asynchrony.....	25
Scaling the expressive asynchrony of a P section.....	26
Scaling the expressive asynchrony of an ACCIA section.....	28
Scaling expressive articulation.....	29
Scaling the expressive articulation of a multilateral section.....	30
Scaling the expressive articulation of a collateral section.....	32
Keeping articulation consistent in the scaling of expressive timing.....	33
Stretch maps.....	35
Interpolate maps.....	35
Transfer maps.....	35
Transformations.....	35
Scale timing.....	37
Keeping articulation consistent.....	41
Scale intervoice expression.....	41
Conclusion.....	43
Acknowledgements.....	43
References.....	43
Microworld expression calculus.....	45

# TOWARDS A CALCULUS FOR EXPRESSIVE TIMING IN MUSIC

*Peter Desain & Henkjan Honing*

Center for Knowledge Technology  
Utrecht School of the Arts  
Lange Viestraat 2B  
NL-3511 BK Utrecht

This paper presents a calculus that enables expressive timing to be transformed on the basis of the structural aspects of the music. Expression within a unit is defined as the deviations of its parts with respect to the norm set by the unit itself. The behaviour of musical material under expressive transformations is determined uniquely by its structural description and the type of expression. Although the calculus separates different kinds of behaviour, it entails no musical knowledge of the transformations themselves and it also does not model music cognition. The algorithmic simplicity of the calculus combined with its elaborate knowledge representation mirrors the common hypothesis that the complex expressive timing profiles found in musical performances can be explained as the product of a small collection of simple rules linked to a relatively complex structure. The calculus (and the program implementing it) will hopefully prove to be a solid basis for formalised theories of music cognition.

## INTRODUCTION

In Desain and Honing (in press, a) we argued that a simplistic notion of a tempo curve of a musical performance is a dangerous and harmful theoretical construct. Although the use of a tempo curve to describe time measurements is perfectly sound, the notion itself is often presented as a cognitive or musical concept. And tempo curves do not have any right to exist in those domains. In the above article, this was concluded from the fact that when it is used as a basis of transformations, inevitably the results make no musical sense. The cause of this failure can often be attributed to the lack of structural information in the tempo curve. For example, in changing the overall tempo of a performance, by manipulating the tempo curve alone, all time intervals of equal length between two notes are scaled in the same way. But some notes may constitute a particular kind of ornamentation, whose duration should be more or less unaffected by tempo. As a result the timing of the piece becomes unmusical. And there are many more examples of transformations that cannot be done on isolated tempo curves. Because the article had an essentially negative tone - identifying the problems and their causes - we felt compelled to follow it up with a study of possible solutions.

This paper is an attempt to identify ways in which structural knowledge can be used to enable expression transformations on musical performances that do make musical sense.

In past research we considered expression merely as deviations of attributes of performed notes from their value notated in a score. This definition, however useful in the initial study of expressive timing, soon lost its attractiveness. In general, listeners can appreciate expression in music performance without knowing the score. And a full reconstruction of the score in the form of a mental representation is often impossible. Take for instance the notion of loudness of notes. Should a listener be required to fully reconstruct the dynamic markings in the score before it is possible to appreciate the deviations from this norm as expressive information added by the performer? Such a nonsensical conjecture indeed follows from a rigid definition of expression as deviations from the score. But it is possible to find ways of defining expression on the basis of performance information only. The more so since it became possible to model the quantization of performed note durations into discrete categories (Desain & Honing, 1991), and therefore even the extraction of performed tempo is possible directly from the performance itself.

In this paper we will base expression on the notion of structural units in a working definition: expression within a unit is defined as the deviations of its parts with respect to the norm set by the unit itself. An example might make this more clear. Lets take, for instance, a metrical hierarchy of bars and beats; the expressive tempo within a bar can be defined as the pattern of deviations from the global bar tempo generated by the tempo of each beat. Or, take the loudness of the individual notes of a chord; the dynamic expression within a chord can be defined as the set of deviations from the mean chord loudness by the individual notes. Using this intrinsic definition, expression can be extracted from the performance data itself, taking more global measurements as reference for local ones, assuming that the structural units themselves are known. Thus the structural description of the piece becomes central, both to establish the units which will act as a reference, and to determine its subunits that will act as atomic parts whose internal detail will be ignored. A generalization of this concept can also deal with expression arising from the interplay of two or more voices.

It will be clear by now that any other connotations of the concept of musical expression, its link to human affect and extra-musical indexicality, however interesting, will be ignored here completely.

Before the details of the calculus are presented it might be fitting to give some explanation for undertaking for this work. First of all, we think that the research of expression in music is in need of measurement instruments that can cope with the enormous complexity of performance data and that are much more sophisticated than tempo curves. Some of the proposed transformations can be used as an "auditory microscope" by



exaggerating expression at certain structural levels, like amplifying the timing lead, the melody often has over the accompaniment. Some of the tools presented can be used as "expression scalpels" for trimming away certain kinds of expression that might obscure other phenomena, like removing the tempo deviations within each beat, but holding the timing patterns of the beats themselves invariant. Other tools can "transplant" musical expression from one piece of music to the other, say from a theme to its variation. The availability of this 'machinery' will deepen our understanding of the intricacies of music performance expression.

A further motivation is the practical applicability of this work in systems for computer music. Especially the music editors and sequencer programs that are commercially available nowadays which are in need of better ways to treat musical information in musical ways. Expressive timing should not be considered a nasty feature of performed music, as it is in nowadays multi-track recording techniques where tempo, timing and synchronization are treated as technical problems. Instead expressive timing has to be regarded as an integral quality of performed music whereby the performer communicates structural aspects of the music to the listener (Clarke, 1988). We hope that our work can inspire new music software based on this view.

## OVERVIEW OF THE CALCULUS

### Characteristics

The calculus has the following important characteristics:

The calculus is described here only for different brands of expressive timing. Dynamics could be formalised along the same lines, but for clarity we restrict ourselves to the domain of expressive timing. Other attributes that carry expression, like intonation, vibrato and timbre may require a different treatment.

The types of expression have to be computable to be within reach of this calculus. One must be able to calculate the expression at every level of the structural hierarchy, given the expression of their components (e.g. the timing of a chord must be computable when the timing of the embedded notes is given). One also must be able to state ways to effectively set the expression of the components once the expression of the whole is given (i.e. propagate a shift in timing down the hierarchy, to the basic objects carrying the expression). Types of expression that do not have this characteristic - or are not yet formalised as such - cannot be described.

Both performance and "score" timing of individual notes are clearly defined. Notes require attributes that can be measured more or less directly from the performance data like the note onset time and the offset time. At least the onset time must be clearly



specified, which makes the calculus less appropriate for expressive performances by instruments for which onset times are not so clear cut. Secondly, the metrical note duration (the timing of the note as notated in the score) must also be available as a note attribute - either via quantization or by matching a performance to a known score. These processes are considered preprocessing here. Although the reference to score duration, score onset and score offset times is less appropriate in the context of our definition of expression - we will use this terminology, for lack of better terms.

The "score" timing of rests is clearly defined. Perhaps surprisingly, the rest plays a key role in some transformations. So we assume that it either can be inferred from the performance timing (Longuet-Higgins, 1976 shows a way of doing so), or it is recovered via the matching of a performance and a known score.

All proposed transformations are structure preserving. This means that the calculus is restricted to true expressive transformations: the score timing of the notes is known and fixed, and transformations will leave this and the structural description invariant.

The behaviour of musical material under expressive transformations is determined uniquely by its structural description and the type of expression.

The transformations are defined on a hierarchical structural description uniquely linking all material. Ambiguous structural descriptions (e.g. two or more possible structural descriptions) or incomplete descriptions cannot be dealt with. The obvious need for knowledge representations containing multiple structural descriptions (metrical, phrase, and rhythmical grouping structures, different analysis etc.) is not denied. We just require that such representations be preprocessed to select only one complete structural description. This is not a real restriction since transformations based on different kinds of structural knowledge of the same piece can always be done in sequence. Re-inserting the trimmed structural descriptions into a transformed piece is trivial because the transformations preserve the structure.

Naturally, the higher-level structural description of the piece must be consistent with the performance timing. For example, a structural description of the piece in which two notes are given a certain sequential time order (one after the other) - can only fit a performance in which at least the onset of the corresponding notes obey the same order. The precise rules will be given when the structural descriptions are introduced.

The transformations are defined to apply to a certain level of the hierarchical structural description, ignoring details from lower levels and keeping higher levels invariant. Means to select such a level are assumed. In sophisticated realisations of the calculus

this may entail a match language ("the first bar of the piano solo that begins with a C") or a graphical representation. In this paper we will simply assume that each musical object has a name as attribute and defines a structural level as the set of objects with a certain name.

Although the calculus separates different kinds of behaviour, it entails no musical knowledge of the transformations themselves. Accordingly, the proposed knowledge representation does support for example, arbitrary descriptions of the metrical structure of a piece, but has no knowledge of "the best structural analysis". To give a second example: the proposed knowledge representation does support ways to modify timing (a)synchrony between voices, but it has no knowledge about correct or effective ways of using this in musical performance.

The calculus also does not model cognition. It does not state how, for example, voice-leading helps auditory streaming, how phrase final lengthening beyond a limited range disables rhythm perception, or how structure is communicated by the expressive timing profiles. However, this work constitutes a solid basis for formalised theories about these issues, providing a powerful representation in which they can be expressed.

## Representation

Several concepts are used in the calculus:

**Musical objects** are either of a basic nature or form a structural description of a collection of musical objects. Basic musical objects consist of notes and rests. Notes are the only musical objects that carry the expressive information. Structural descriptions form collections of musical objects. They may describe hierarchical time intervals like metrical-, phrase- or rhythmical grouping, they can group the notes of chords and ornaments together, or form large horizontal slices through the piece, describing the separate voices etc. Mere collections (sets) of objects are too meager a basis for most transformations, therefore, structural descriptions specify the intended relations in time between these objects as well (Honing, 1991). Most transformations can be defined if two orthogonal characteristics of the structural description are given: the temporal nature and the ornamenting quality. The first describes whether a sequence or a parallelism (a so called successive or simultaneous construct) is represented. The second describes whether the musical object is considered an ornament attached to another object or not. Ornaments are shielded from certain modifications and refer to another object for certain attributes. These two binary characteristics result in four concrete types of structural description that will be described in detail later.

**Expressive magnitudes** are values of expressive measurement on a certain scale. The scales themselves are of course crucial in modeling effective transformations, in cognitive and musical senses. For example, a tempo scale on which a transformation to make something twice as fast actually yields a double perceived tempo is quite useful. But for the sake of simplicity we abstract from the perceptual processes and the instruments that generate the sound, and will just assume simple physical measurements of time and other expressive attributes.

**Expression maps** describe the expressive patterns of structured musical objects at a certain structural level. They consist of a section for each musical object at that level. A section lists the expressive values for all components of that object. They come in different brands - consistent with the type of musical structure where they were extracted from. Expression maps can be extracted from and applied to musical objects, with possibly a modification of the map in between.

**Expression types** are sets of procedures to extract a particular type of expression map from a musical object, to impose it on a musical object, and to modify the map. They capture the difference between expressive tempo, asynchrony, and articulation. They may become fairly sophisticated, like a brand of expressive tempo that knows how to keep the articulation of an individual note invariant when the timing of the note onsets is changed.

**Modifications** are defined as operations on expression maps. They may scale, interpolate, or do any other operation on the map. They are often designed such that certain characteristics are kept invariant, e.g. the total duration of a section while changing the timing of the parts.

**Transformations** are defined as operations on musical objects. They are often direct generalizations of the expression map modifications - first extracting the map, applying the modification and imposing the modified map. They also handle the selection of the level of structural description on which to apply the transformation. Furthermore, they may have means to maintain consistency among the affected level and other musical material, e.g. making an accompaniment obediently follow the transformation in expressive tempo applied to the melody.

## Implementation

Part of the work described in this paper was done in the design of the POCO system (Honing, 1990) for which a scaling operation of expressive timing linked to structural descriptions was implemented. But, in evaluating this rather complex piece of software, better abstractions arose. Especially the design of a set of data structures for music that



capture the differences in behaviour under transformation proved beneficial. Which again illustrated the adage:

*"Get your data structures correct first, and the rest of the program will write itself."* (David Jones, quoted in Bentley, 1988)

Because the constructs interact heavily, and because it should be easy to add unforeseen new constructs (like a new type of expression), musical objects, expression maps and expression types are implemented as classes in an object-oriented language. In that way it is easy to express modifications and transformations as polymorphic operations that will behave according to the type (the class) of their arguments. The slicing-up of knowledge in classes means answering questions like: which part of the extraction procedure of an expressive tempo map of a sequential musical object is specific for expressive tempo only and should be stated within the expression type; which part only depends on the sequential nature of the musical structure, and should be part of the class for sequential musical objects; and which part describes the creation of an expression map and belongs to that class?

Although a good Object-Oriented Language (we used CLOS, a Lisp-based system) provides one with the programming-constructs needed to express these concepts, the actual process of factoring knowledge into these polymorphic procedures is still a difficult one, especially because during the design of the best structure of the classes - allowing for the most elegant factoring of the procedures - cannot be completely foreseen. This forced us to go through several re-design rounds before the concepts stabilized in their present form.

The following CLOS (Keene, 1989; Steele, 1990) constructs were used heavily in the implementation: multiple inheritance (forming class dependencies that are more complex than simple hierarchies), multi-methods (functions that are polymorphic in more arguments), mix-in type of inheritance (grouping of partial behaviour in an abstract class that must be mixed in with other classes to supply that behaviour to their instances), method combination (providing ways of combining partial descriptions of behaviour of one method for more classes). Together they make it possible to extend the system by adding program code only, instead of rewriting it.

The calculus will be incorporated in POCO. The other tools available in POCO, like score-performance matchers, multiple structural descriptions, storage and retrieval from standard MIDI-files, playback and editors for music text formats etc., will support a comfortable use of the calculus with real performance data. An implementation in the form of the microworld is given in the appendix and aims at conciseness and elegance. Luckily, this goal only occasionally conflicts with computational efficiency.

The following five paragraphs will describe the calculus in more detail. The reader interested in the more general aspects of the calculus is advised to continue reading below Transformations.



## MUSICAL OBJECTS

Musical objects come in different brands. Some types are specific enough to describe an object completely (the instantiatable or concrete classes). Other types are used as a descriptive grouping of likewise behaviour (the abstract classes). The types of musical objects and their interrelations are shown in figure 1.

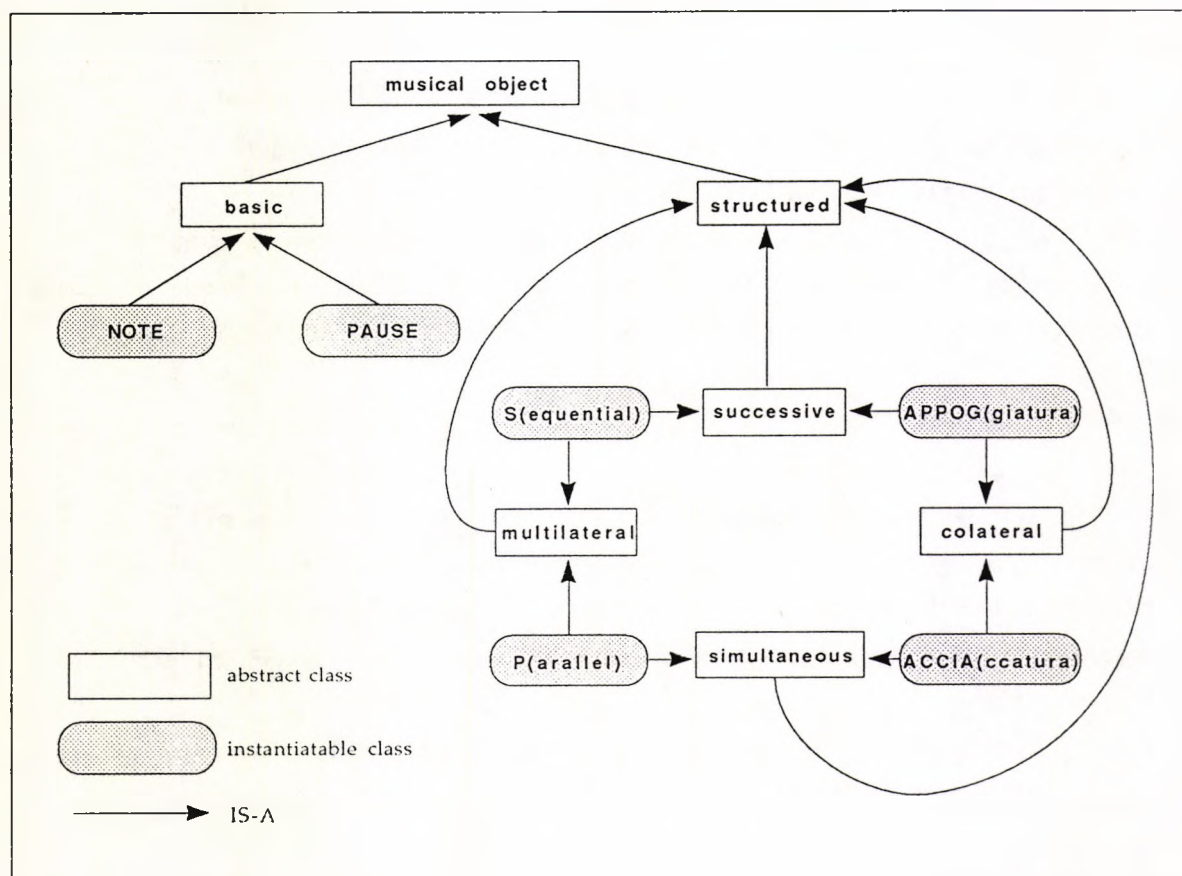


Figure 1. Classes of musical objects and their interrelations.

### Basic musical objects

Basic musical objects are notes and rests (In the program we use the word PAUSE to avoid the name clash with the Common Lisp primitive REST). In examples we will use notes with clearly observable onset and offset times (called PERF-ONSET and PERF-OFFSET) measured in ms. from the beginning of the performance. Both notes and rests have as a property a time position in the score (called SCORE-ONSET and SCORE-OFFSET) measured in any kind of (beat)-count (a rational number). These score times are calculated automatically from the supplied score durations of notes and rests via the structural descriptions. This facilitates easy creation of large scores.

Rests are essential and cannot just be ignored, as is done in some low-level representations (e.g. the Midi-file standard). They are central e.g. in dealing with articulation - a short

note followed by a rest behaves differently under transformation than a longer note played in a staccato way.

## Structured musical objects

### Multilateral structures

In research on music perception and cognition a distinction is often made between successive temporal processes that deal with events occurring one after another, and simultaneous temporal processes that handle events occurring around the same time (e.g. Bregman, 1990; Serafine, 1988). For the first type of events of the expressive means can be rubato - the change of tempo over the sequence. In the second one the expressive means can be chord-spread and asynchrony between voices, both more timbral aspects. These processes work differently in perception. Since we want to imply differences in behaviour mainly by differences in structural description a way should be found in which both these constructs can be represented.

We propose to use for this purpose the simple time structures S and P that functioned well in (Desain & Honing, 1988; Desain, 1990; Desain & Honing, in press, b). If a collection of musical objects is formed such that they occur one after another they are described as a successive structured object named S (for Sequential). If a collection of musical objects occur at the same time they can be collected in a simultaneous structured object called P (for Parallel). These structures serve as a general way to represent a collection together with the temporal relation between the components, as stated in the score. We call the objects *multilateral* because their components are considered to be of equal importance, and are to be treated as such in expressive transformations.

The score times of a structured object and its parts are constrained by consistency rules. They are described separately in frames 1 and 2. These constraints are enforced by specifying only notes and rests with a score duration. The constraints propagate these automatically when a structural description is created and set all score onset and offset times.

In calculating expression, the previous and subsequent context of musical objects is sometimes needed. For instance, consider articulation: possibly defined as the overlap between the sounding parts of a note and the next one, i.e. the time difference between the offset of the note itself and the onset of the "next" note. Besides "next material" a link to "previous material" is foreseen to be needed as well, e.g. in the calculation of local accent patterns. To formalize and generalize this notion of "previous" and "next" material a definition of the left and right context of a musical object is given. This notion also reflects the fact that some expressive values cannot be calculated because some contexts are not available or carry no expression e.g. the tempo of the last note in a piece, or the performance onset of a voice that starts with a rest. Expressive transformations must thus expect to come across missing values in an expression map. The notion of context is

explicitly represented in the program as attributes of the objects themselves. This is possible because the structural description is invariant and so are the contexts. Another possibility would be to represent them implicitly, recovering them by search via a bi-directional part-of link between musical objects. Alternatively, they could be represented tacitly, i.e. supplying them by a general control structure that walks through structured musical objects.

#### **Collateral structures (ornamented objects)**

Some musical objects contain components that should maintain a dependency relation to one another. If such a *collateral* pair is transformed, the transformation should be carried out on the main component only, the submissive one obediently following the main component's transformation, but not being transformed itself. An ornamented musical object like a graced note (a note preceded by a grace note), is a good example of a collateral object. For example, in the scaling of the expressive tempo of a melody which contains a graced note, the data on which the expressive transformation is carried out (in this case the performance onset) stems from the main object. The grace note is ignored. When in the actual transformation the graced note pair is stretched or compressed and moved to an other point in time, only the main note will undergo that operation. The ornament will just follow its shift in time.

A second use of this concept is made when the relation of an ornament to its main object, within such a collateral couple, is considered to be expressive, and a potential source of expressive transformations. In this case, the main object stays invariant, and only the ornament undergoes transformation. Take for example the asynchrony between the performance onset of a grace note and the note it is attached to. This time interval can be modified by appropriate means, resulting in local changes of the timing of the grace note - but keeping the timing of the main note invariant.

Collateral (ornamented) objects can again have two kinds of temporal nature: successive or simultaneous. The first one is called APPOG (for *appoggiatura*). It describes a "time-taking" ornament where the ornament is considered to finish when its main object starts (all in terms of score times). The second is called ACCIA (for *acciaccatura*). It can represent a so called "time-less" ornament that is supposed to start at the same time as the object it is attached to. Note that both parts of a collateral pair are musical objects themselves and can have internal structure. The concepts of APPOG and ACCIA ornamented objects are an elaboration of the PRE and POST objects that were introduced in (Desain & Honing, 1988). Consistency rules for score times and context are described in frames 3 and 4.



## S, a multilateral successive structure

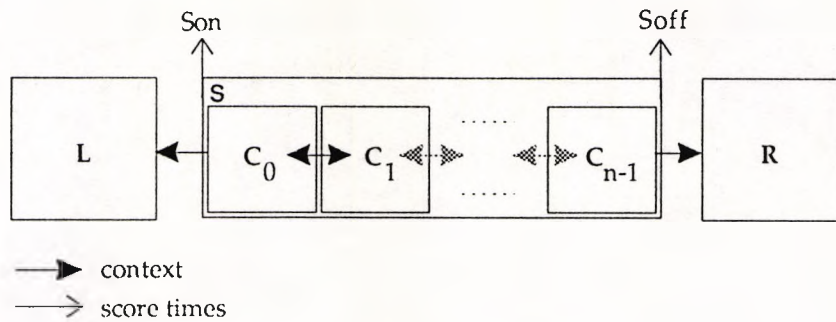


Figure of S object

Consider an S structure of  $n$  components  $C_i$  with  $0 \leq i \leq n-1$ .

Assume that component  $C_i$  has score onset time  $Son_i$ , score offset time  $Soff_i$  and that the whole structure has score onset time  $Son$  and score offset time  $Soff$ . Then the following must hold:

$$Son = Son_0$$

$$Soff = Soff_{n-1}$$

$$Soff_i = Son_{i+1}, \text{ for } 0 \leq i \leq n-2$$

Assume that component  $C_i$  has performance onset time  $Pon_i$  and that the whole structure has performance onset time  $Pon$ . Then the following must hold:

$$Pon = Pon_0$$

$$Pon_i \leq Pon_{i+1}, \text{ for } 0 \leq i \leq n-2$$

Assume that component  $C_i$  has left context  $L_i$  and right context  $R_i$  and that the whole structure has left context  $L$  and right context  $R$ . Then the following holds:

$$L = L_0$$

$$R = R_{n-1}$$

$$R_i = C_{i+1}, \text{ for } 0 \leq i \leq n-2$$

$$L_i = C_{i-1}, \text{ for } 1 \leq i \leq n-1$$

Frame 1. Description of a S structure.



### P, a multilateral simultaneous structure

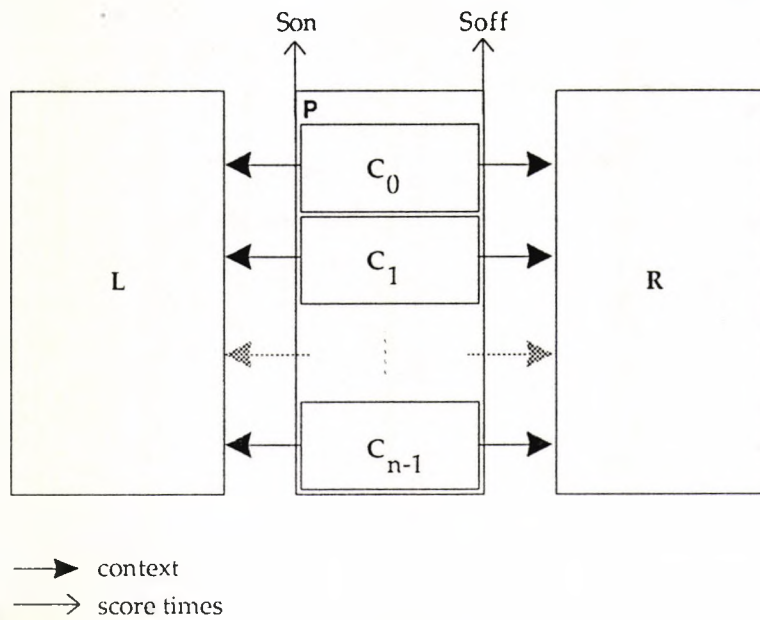


Figure of P object

Consider a P structure of  $n$  components  $C_i$  with  $0 \leq i \leq n-1$ .

Assume that component  $C_i$  has score onset time  $Son_i$  and score offset time  $Soff_i$  and that the whole structure has score onset time  $Son$  and score offset time  $Soff$ . Then the following must hold:

$$Son_i = Son, \text{ for } 0 \leq i \leq n-1$$

$$Soff_i = Soff, \text{ for } 0 \leq i \leq n-1$$

Assume that component  $C_i$  has performance onset time  $Pon_i$  and that the whole structure has performance onset time  $Pon$ . Then the following holds:

$$Pon = \text{MIN}_{0 \leq i \leq n-1} Pon_i$$

Assume that component  $C_i$  has left context  $L_i$  and right context  $R_i$  and that the whole structure has left context  $L$  and right context  $R$ . Then the following holds:

$$L = L_i, \text{ for } 0 \leq i \leq n-1$$

$$R = R_i, \text{ for } 0 \leq i \leq n-1$$

Frame 2. Description of a P structure.

### APPOG, a collateral successive structure

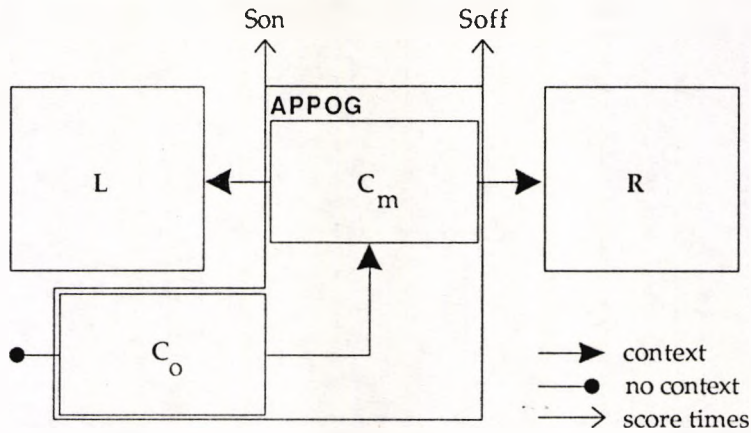


Figure of APPOG object

Consider a APPOG structure of a ornament component  $C_o$  and a main component  $C_m$ . Assume that component  $C_o$  has score onset time  $Son_o$ , score offset time  $Soff_o$ , that component  $C_m$  has score onset time  $Son_m$  and score offset time  $Soff_m$  and that the whole structure has score onset time  $Son$  and score offset time  $Soff$ . Then the following must hold:

$$\begin{aligned}
 Son_m &= Son \\
 Soff_m &= Soff \\
 Soff_o &= Son_m
 \end{aligned}$$

Assume that component  $C_o$  has performance onset time  $Pon_o$ , component  $C_m$  has performance onset time  $Pon_m$  and that the whole structure has performance onset time  $Pon$ . Then the following holds:

$$\begin{aligned}
 Pon &= Pon_m \\
 Pon_o &\leq Pon_m
 \end{aligned}$$

Assume that component  $C_o$  has left context  $L_o$  and right context  $R_o$ , component  $C_m$  has left context  $L_m$  and right context  $R_m$  and that the whole structure has left context  $L$  and right context  $R$ . Then the following holds:

$$\begin{aligned}
 L &= L_m \\
 R &= R_m \\
 R_o &= C_m \\
 L_o &= \text{undefined}
 \end{aligned}$$

Frame 3. Description of an APPOG structure.

### ACCIA, a collateral simultaneous structure

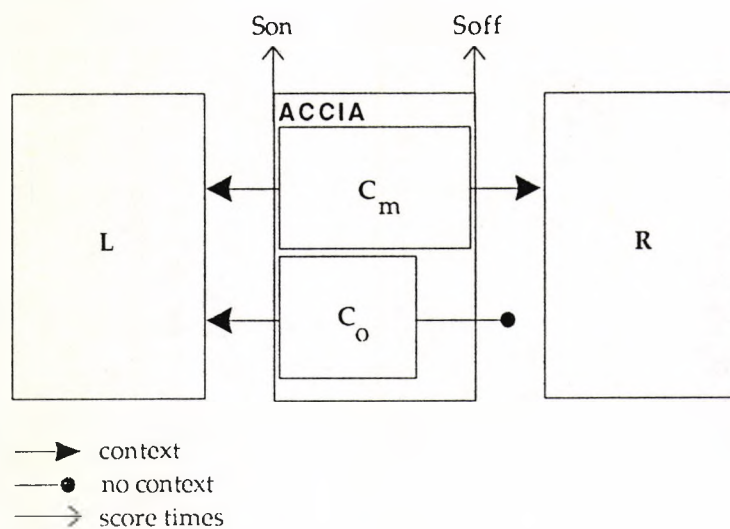


Figure of ACCIA object

Consider a ACCIA structure of a ornament component  $C_o$  and a main component  $C_m$ . Assume that component  $C_o$  has score onset time  $Son_o$ , score offset time  $Soff_o$ , that component  $C_m$  has score onset time  $Son_m$  and score offset time  $Soff_m$  and that the whole structure has score onset time  $Son$  and score offset time  $Soff$ . Then the following must hold:

$$Son_o = Son_m = Son$$

$$Soff_m = Soff$$

Assume that component  $C_o$  has performance onset time  $Pon_o$ , component  $C_m$  has performance onset time  $Pon_m$  and that the whole structure has performance onset time  $Pon$ . Then the following holds:

$$Pon = Pon_m$$

Assume that component  $C_o$  has left context  $L_o$  and right context  $R_o$ , component  $C_m$  has left context  $L_m$  and right context  $R_m$  and that the whole structure has left context  $L$  and right context  $R$ . Then the following holds:

$$L = L_m = L_o$$

$$R = R_m$$

$$R_o = \text{undefined}$$

Frame 4. Description of an ACCIA structure.



## EXAMPLE OF THE REPRESENTATION OF A MUSICAL OBJECT

In figure 2 a fragment of a score is shown that will serve as a basis for the examples at the end of this article. It is the score of the last bars of the theme of six variations over the duet *Nel cor più non mi sento*, by Ludwig van Beethoven (with some adaptations), which is the same material used to study tempo curves in (Desain & Honing, in press, a). It contains examples of several kinds of musical structure: chords, voices, sequences, bars and beats, phrases and two types of ornaments. Figure 3 shows a graphical notation indicating two structural descriptions: a metrical hierarchy and a separation into voices. The way these structures are specified in Lisp is given in the appendix.



Figure 2. Score of the last bars of the theme of six variations over the duet *Nel cor più non mi sento*, by Ludwig van Beethoven.

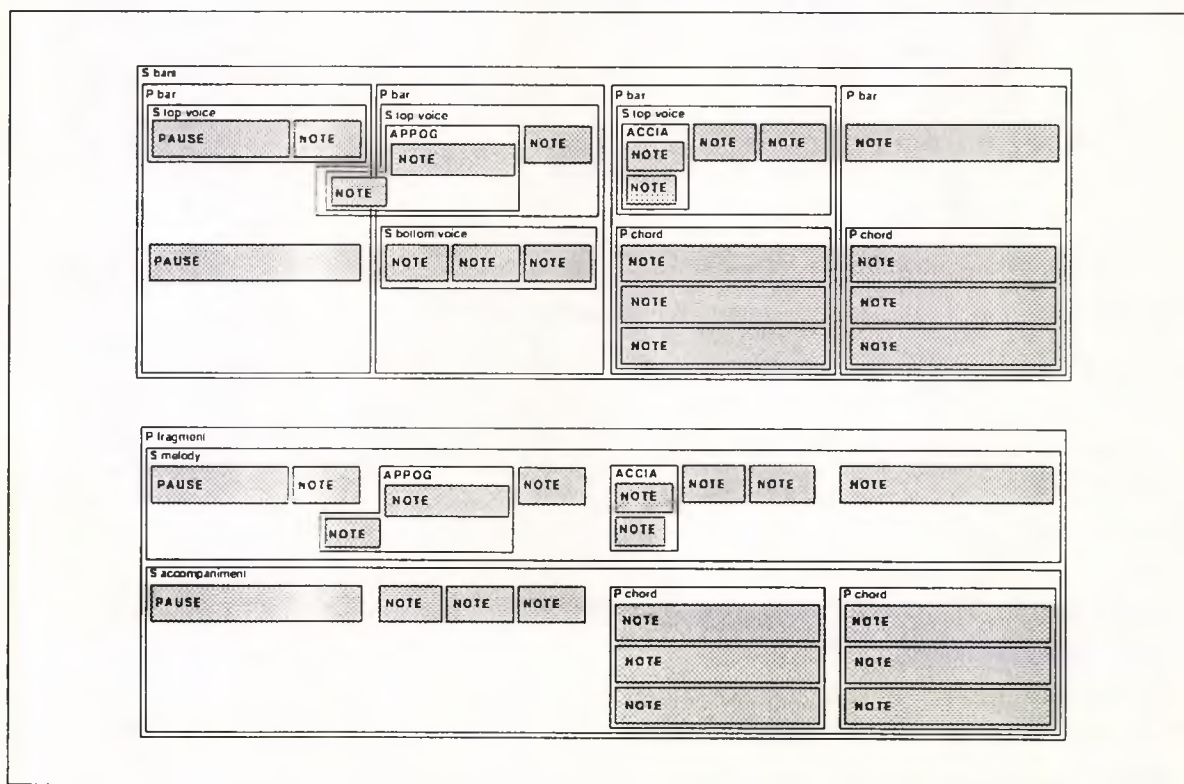


Figure 3. a) Structural description of the metrical hierarchy of the score in figure 2, and b) Structural description of the voices in that piece.



## REPRESENTING EXPRESSION

There are three kinds of expressive timing: expressive tempo, expressive asynchrony and expressive articulation. The first two are based on performance onset times only, the third is based on performance onset and offset times (see figure 4).

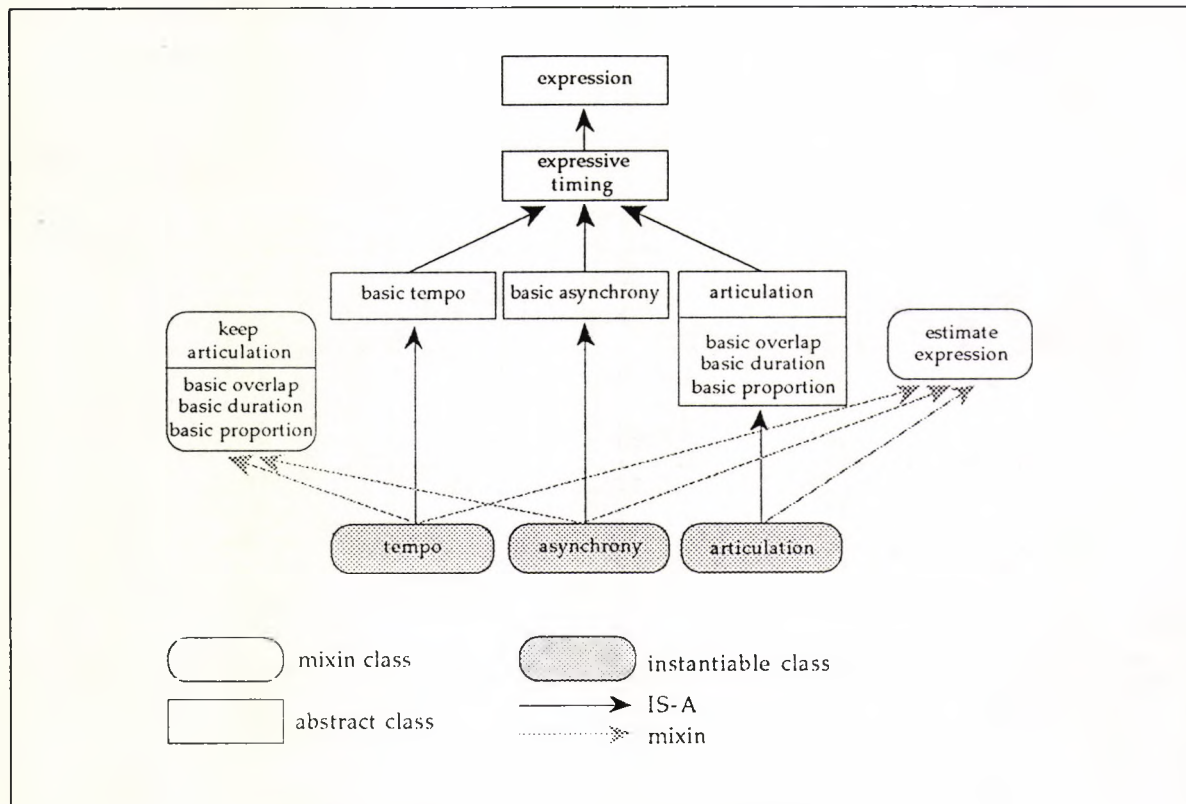


Figure 4. Expression type hierarchy.

One could imagine sophisticated algorithms that calculate the onset of a note and of parallel structures on the basis of their perceptual onset (P-center; see Vos & Rasch, 1981). But for clarity we use a very simple definition of onset times, which was already given in the frames 1 to 4. In that way, all musical objects have performance onset times and so can be used as units on which tempo and asynchrony measures are built.

### Expressive tempo

The notion of tempo is relevant only for successive structures. It is defined as the ratio of score duration and performance duration. This velocity-like notion the inverse of the notion of a tempo factor, as is used in the psychology of music literature.

### Expressive asynchrony

The notion of asynchrony is relevant only for simultaneous structures. It is defined as the difference of performance onsets. It is thus independent of score times.

## Expressive articulation

Expressive articulation uses the performance offsets of individual notes. It simply assumes that they are given. A definition of performance offset of structured musical objects is not needed. Articulation is also independent of score times.

Articulation can be defined in several ways - but it is hard to find a way that will suffice in all circumstances. In the legato range the absolute overlap time of the sounding part of a note and the next one seems a good candidate for an articulation scale. In the staccato range the absolute sounding duration of the note seems the most prominent perceived aspect. In the intermediate range the relative sounding proportion is a good measure. For the moment we cannot do better than to supply these three concepts of articulation expression (overlap-, duration- and proportion-articulation) - leaving it for the user to choose the most appropriate one (see frame 5). For a multilateral structure the expressive articulation value is taken to be the average articulation of its parts. For a collateral structure the expressive articulation value is defined to be the articulation of its main part.

### Definition of articulation

Consider a note with performance onset  $P_{on}$ , performance onset  $P_{off}$  and performance onset of its right context  $P_{on_r}$ . There are three alternative definitions of articulation  $A$  given:

overlap articulation  $A = P_{off} - P_{on_r}$

duration articulation  $A = P_{off} - P_{on}$

proportion articulation  $A = \frac{P_{off} - P_{on}}{P_{on_r} - P_{on}}$

If a multilateral structure with articulation  $A$  has components  $C_i$  for  $0 \leq i \leq n-1$ , and  $C_i$  has articulation  $A_i$  then:

$$A = \text{MEAN}_{0 \leq i \leq n-1} A_i$$

If a collateral structure has articulation  $A$ , and its main component  $C_m$  has articulation  $A_m$  then:

$$A = A_m$$

Frame 5. Definition of articulation expression.

## Estimate onsets

Because sometimes the performance onset of missing objects (like the virtual note after the end of the piece) or the performance onset of a rest are needed, we devised a set of procedures that estimates these missing values on the basis of performance onsets that can be found in the context, using a linear interpolation or extrapolation method. The set of procedures forms a mix-in class that can be combined with any expressive timing type enabling that kind of expressive timing to deal - in all operations - with missing values. Estimation is derived from the same structural level as the transformation itself. For example, a transformation on a beat structure in need of a missing expressive value at the end of the piece (cf. the onset the final barline in a score) will be estimated on basis of the two previous beats -not on the basis of any internal detail. In the case of extreme tempo variations, as occur in a final retard, the estimation feature cannot work well. In this case it is better not to use it.

## Articulation invariance

When moving the onsets of notes around (e.g. in modifying the performance onsets) it is quite annoying that the articulation of the individual notes also changes - an effect that is very easy to perceive and which may well overshadow subtle modifications of onset timing. Therefore a set of procedures can be mixed-in with expressive tempo and asynchrony. They are given a chance to calculate the articulation of individual notes before onsets are changed and to reinstall it afterwards. This will insure that articulation is kept invariant under transformations of onset timing (see figure 12).

## EXPRESSION MAPS

An abstraction of the expression of an object is useful for many operations because it can hide the irrelevant details of the structure and provides a means to transfer expression from one object to another. Therefore expression maps were introduced. They describe expression of musical objects at one level of a structural description. All objects at the level described must have the same structural type. Maps contain a list of sections, one for each of those musical objects. A section lists the expressive values of the components of that musical object. Of course maps may be partial - consisting of several sections with gaps in between, or even have missing values within a section.

## Onset timing

The application of a (modified) map of performance onsets on an object works as follows. First, all objects at the indicated level are found, paired with their corresponding sections. Then each section is applied to its object. This means that the components of that object are provided one by one with a new onset from that section.

This setting of onsets is handled differently according on the structural type of the component. If this component is a note, the onset is set directly. For S components the whole structure is stretched between that onset and and the next onset (the onset of the succeeding component). A P component is set to the provided onset, but keeps its internal asynchrony invariant and truncates at the next onset. In the case of a ACCIA component, the main structure is set to the onset, with the ornament following the displacement of the main structure. Finally, for a APPOG component, the main structure is stretched between that onset and the next onset, with the ornament also simply following the displacement of the main object.

Now we have indicated how an expressive timing is applied to components of structured objects - it remains to be shown how such a change propagates when these components again are embedded structured objects themselves. This fairly complex process depends on the type of the embedded structured object and mirrors the decisions given above: S components are stretched, P components are shifted and truncated, and ornaments follow the shift of their main components.



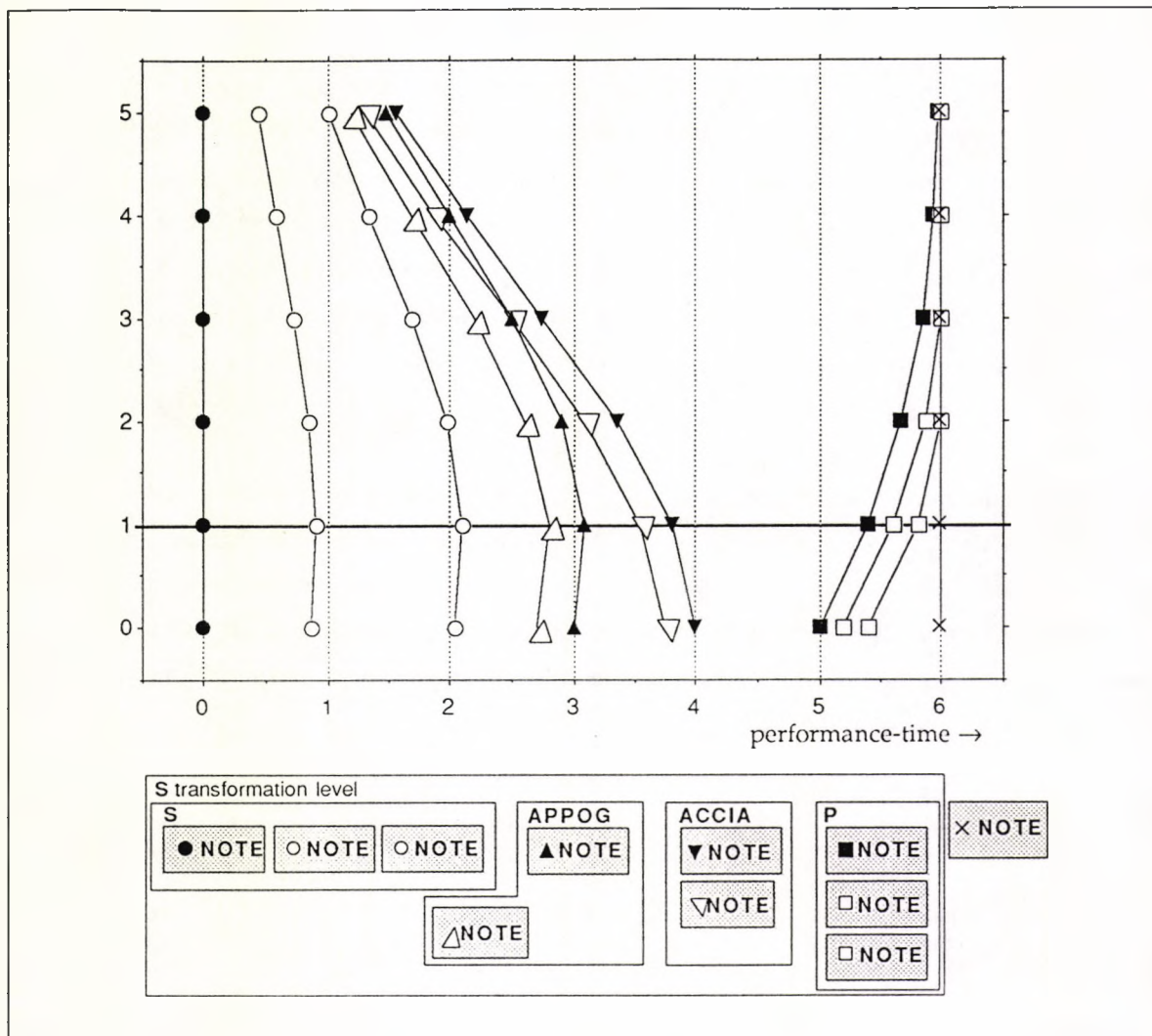


Figure 5. Propagation of change of onset within an S structure for different component types. This figure shows the propagation process for an S structure containing different types of structural components. We assume the components are moved around by an arbitrary transformation, parametrized by a factor. In this figure it is shown how this change is propagated to the internal structure of different kinds of components. The first component is an S structure and the onsets of its internal parts (lines marked with white circles) are stretched along proportionally. The second sub-structure is an APPOG structure and one can see that the onset of its ornament (line marked with upward pointing white triangles) shifts along with the main object. The third sub-structure is an ACCIA structure and the onset of its ornament behaves likewise. Note that the onset of the ornament is allowed to shift freely (line marked with downward pointing white triangles), even the order of notes is allowed to change here. The fourth sub-structure is a P structure and the onset of its components (lines marked with squares) are shifted and truncated at the end (the right context note; line marked with x's).

## Articulation expression

In comparison, to set the articulation expression to a structured object is much simpler.

When a section of an articulation map is applied to a multilateral or collateral structure the articulation of its components are set to their respective values from the section.

The propagation of a (modified) articulation value to a component works as follows. If that component is a note, a new offset is calculated from the articulation value and set directly, taking care to maintain reasonable offset times (e.g. not shifting before its onset). If that component is a multilateral structure, its articulation is calculated (the

mean articulation of its components) and the difference with the required articulation is propagated as an increment to all components. If it is a collateral structure, its articulation is calculated (the articulation of its main component) and the difference with the required articulation is propagated as an increment to both main and ornament components.

## OPERATIONS ON EXPRESSION MAPS

Operations on expression maps work section by section. In each section the expression of a structured musical object is represented. The operations delivers a new section to be applied to that object. Care was taken to maintain structural consistency in all operations even in case of extreme parameter values. Of course expression transformations are intended as subtle changes and truncation or extreme normalization should in practice never occur.

### Scale maps

#### Scaling expressive tempo

Scaling tempo is done in an exponential way. Inverse tempi are considered to be related by a scale factor -1; twice as slow is considered to be the mirror image of half as slow. This exponential scaling of expressive tempo mirrors the exponential nature of notated note durations.

## Scaling the expressive tempo of an S section

The scaling of the expressive tempo of a multilateral successive structure works as follows. Assume the structure has  $n$  components named  $C_i$  with  $0 \leq i \leq n-1$ . Assume component  $C_i$  has score onset time  $Son_i$  and performance onset time  $Pon_i$ . Assume the right context of the structure (and thus the right context of component  $C_{n-1}$ ) is object  $C_n$ . It has score onset  $Son_n$  and performance onset time  $Pon_n$ . A section of the expressive tempo map of the structure contains all  $Son_i$  and  $Pon_i$  including  $Son_n$  and  $Pon_n$ . The scale operation on such a section delivers a new section with performance onsets  $Pon_i'$  according to the following rules:

Define the score inter-onset interval  $\Delta Son_i$  and the performance inter-onset interval  $\Delta Pon_i$  and the local tempo  $T_i$  for  $0 \leq i \leq n-1$  (a better term would be velocity) as:

$$\Delta Son_i = Son_{i+1} - Son_i$$

$$\Delta Pon_i = Pon_{i+1} - Pon_i$$

$$T_i = \frac{\Delta Son_i}{\Delta Pon_i}$$

This ratio is scaled by an exponential factor  $f$ .

$$T_i' = T_i^f$$

Then new raw performance durations  $\Delta Pon_i''$  are calculated:

$$\Delta Pon_i'' = \frac{\Delta Son_i}{T_i'}$$

These are re-normalised such that the total performance duration is kept invariant.

$$\Delta Pon_i' = \Delta Pon_i'' * \frac{Pon_n - Pon_0}{\sum_{i=0}^{n-1} \Delta Pon_i''}$$

Starting at the same point, the new performance times are given as:

$$Pon_i' = Pon_0 + \sum_{j=0}^{i-1} \Delta Pon_j'$$

Frame 6. Scaling the expressive tempo an S section.



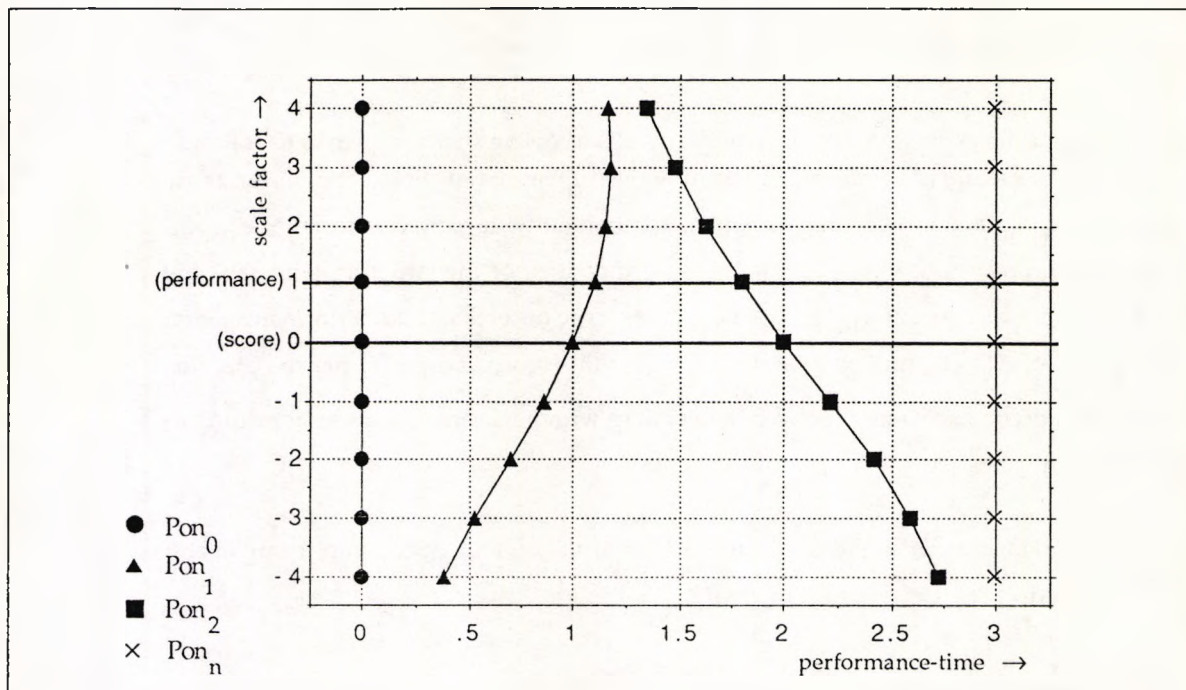


Figure 6. Scaling the expressive tempo of an S section. This process is shown for a specific set of performance onsets  $Pon_i$ . In this figure the horizontal axis is the performance time  $P$ . On the vertical axis the scale factor  $f$  is given. Thus at the horizontal line at scale factor 1 the performance times  $Pon_i'$  are shown as markers on the line; they are identical to the original performance times  $Pon_i$ . This operation (with scale factor 1) is the identity transformation with respect to the performance timing. At the horizontal line at scale factor 0 the performance times  $Pon_i'$  are identical to the score times  $Son_i$  (modulo normalization to the total performance duration). This operation (with scale factor 0) effectively removes the expressive timing of the performance. At factor .5 a diminished expressive timing profile will result, and at factor 2 an exaggerated rubato can be obtained. At negative values of the scale factor the expressive profile is inverted: a slower tempo becomes faster and vice versa. At extreme values of the scale factor the note that is played at the slowest tempo in the performance will gain almost the whole performance time interval spanned by the structure, pushing other notes to zero duration. When the performance onset  $Pon_n$  is not available, the scale transformation uses  $Pon_{n-1}$  instead, and scales the tempo of the section with regard to the onset of the last component in the section - instead of the onset of the right context. This tempo scaling method works well for S constructs with many components and small tempo deviations.



### Scaling the expressive tempo of an APPOG section

The scaling of the expressive tempo of a collateral successive structure works as follows. Assume this structure has a main component with score onset time  $Son_m$  and performance onset time  $Pon_m$  and a preceding ornament component with score onset time  $Son_o$  and performance onset time  $Pon_o$ . Assume the right context of the structure (and thus the right context of component  $C_m$ ) is object  $C_r$ . It has score onset  $Son_r$  and performance onset time  $Pon_r$ . An APPOG time map section contains this score and performance data. The scale operation on such a map delivers a new map with performance onsets according to the following rules:

Define the main and ornament score inter-onset interval  $\Delta Son_m$ ,  $\Delta Son_o$  and the main and ornament performance inter-onset interval  $\Delta Pon_m$ ,  $\Delta Pon_o$  as:

$$\Delta Son_m = Son_r - Son_m$$

$$\Delta Son_o = Son_m - Son_o$$

$$\Delta Pon_m = Pon_r - Pon_m$$

$$\Delta Pon_o = Pon_m - Pon_o$$

The ornament tempo  $T_o$  and the main tempo  $T_m$  are calculated as:

$$T_o = \frac{\Delta Son_o}{\Delta Pon_o}$$

$$T_m = \frac{\Delta Son_m}{\Delta Pon_m}$$

$T_o/m$  is tempo of the ornament relative to the main tempo. This factor is scaled by an exponential parameter  $f$ , and a new ornament tempo  $T_o'$  is calculated:

$$T_o/m = \frac{T_o}{T_m}$$

$$T_o' = T_m * T_o/m^f$$

This gives a new performance duration  $\Delta P_o'$ , which yields the new performance times  $Pon_m'$  and  $Pon_o'$ :

$$\Delta P_o' = \frac{\Delta Son_o}{T_o'}$$

$$Pon_m' = Pon_m$$

$$Pon_o' = Pon_m - \Delta P_o'$$

Frame 7. Scaling the expressive tempo of an APPOG section.

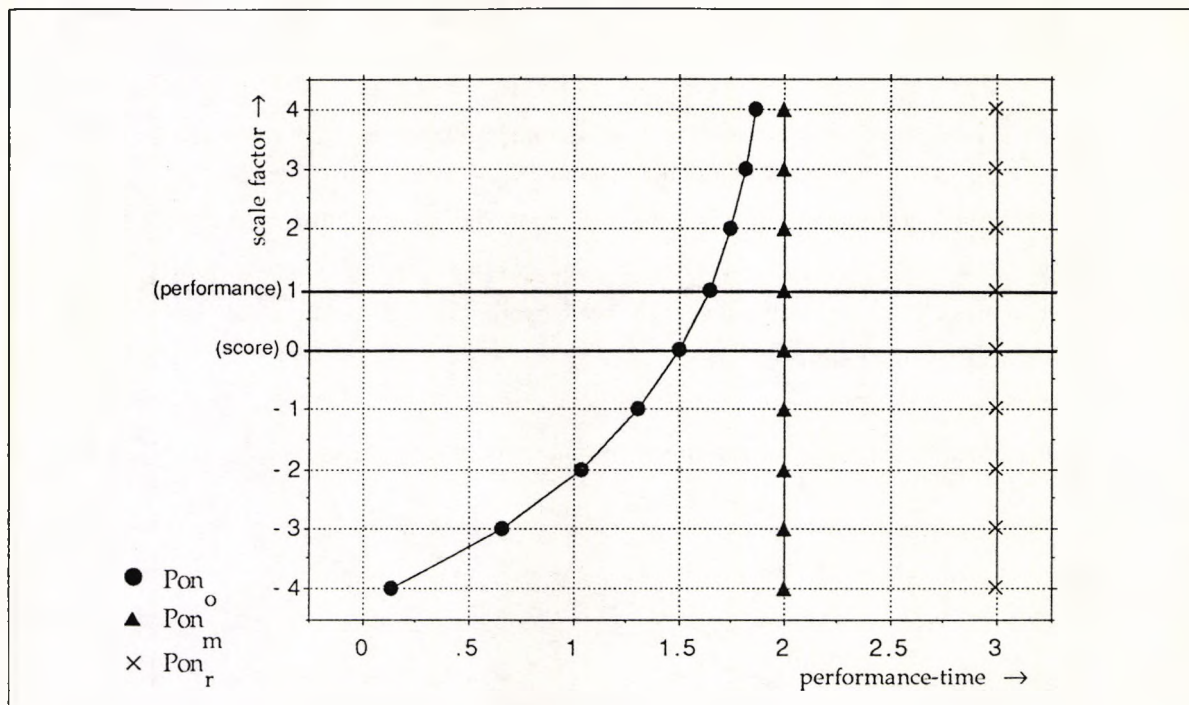


Figure 7. Scaling expressive timing of an APPOG section. This process is shown for a specific set of performance onsets. Note that only the performance timing of the ornament is affected. At scale factor 1 the timing of the ornament is identical to the original timing. At scale factor 0 the ornament is performed at the same tempo as the main object (in this particular example the score duration of the ornament is half that of the main component). This operation (with scale factor 0) effectively removes the expressive way in which the ornament is performed, relative to the main component. At factor .5 a diminished expressive timing effect will result, and at factor 2 an exaggerated effect will be obtained. At negative values of the scale factor the expressive timing is inverted: a performance of the ornament at a lower tempo than the main component becomes one at a faster tempo and vice versa.

### Scaling expressive asynchrony

Asynchrony occurs when two or more simultaneous musical objects - prescribed to happen at the same score time - have unequal performance onsets. The differences can be scaled linearly but care has to be taken not to disrupt the timing of higher levels.

### Scaling the expressive asynchrony of a P section

The scaling of the expressive asynchrony of a multilateral simultaneous structure works as follows. Assume the structure has  $n$  components named  $C_i$  with  $0 \leq i \leq n-1$ . Component  $C_i$  has performance onset time  $Pon_i$ . Assume the right context of the structure (and thus the right context of all components) has performance onset  $Pon_n$ . A parallel time map of the structure contains all  $Pon_i$  including  $Pon_n$ . The scale operation on such a map delivers a new  $Pon_i'$  according to the following rules:

Let the global performance onset  $Pon$  and the performance onset asynchronies  $\Delta Pon_i$  be defined as:

$$Pon = \text{MIN}_{0 \leq i \leq n-1} Pon_i$$

$$\Delta Pon_i = Pon_i - Pon \text{ for } 0 \leq i \leq n-1$$

The asynchronies are scaled by an multiplication factor  $f$ :

$$\Delta Pon_i' = \Delta Pon_i * f$$

New performance onsets  $Pon_i'$  are calculated, shifting such that the global performance onset is kept invariant ( $\min(Pon_i') = \min(Pon_i) = Pon$ ). The result is truncated such that the onsets never move beyond  $Pon_n$ . Of these two safeguards the first applying in case  $f$  is negative, the second applying in case  $f$  is large compared to the ratio of the asynchronies and performance duration of the whole structure. Together they ensure consistency with higher-level structural descriptions by keeping the components within the bounds of the structure.

$$Pon_i' = \text{MIN}(Pon_n, Pon + \Delta Pon_i' + \text{MIN}(\Delta Pon_i'))$$

*Frame 8. Scaling the expressive asynchrony of a P section*

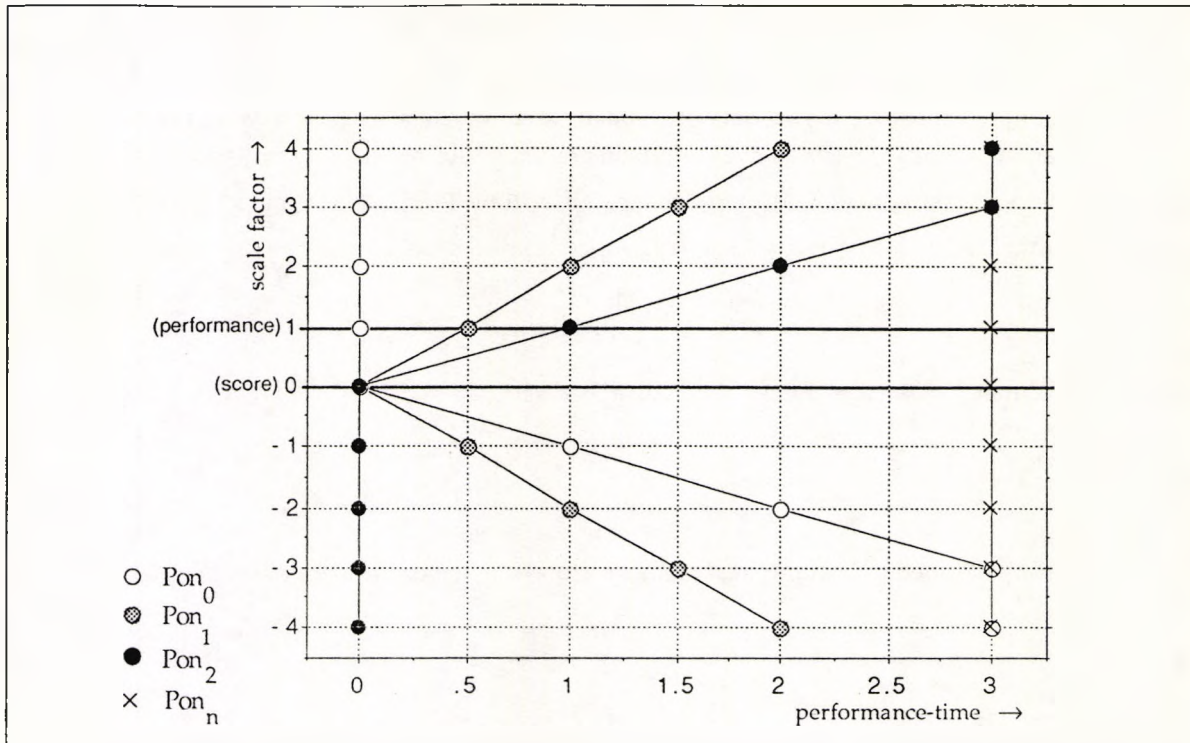


Figure 8. Scaling expressive timing of a P section. This figure shows this process for a specific set of performance times  $P_i$  (say a chord performed with some spread). At scale factor 1 the performance onsets  $Pon_i'$  are identical to their original  $Pon_i$ . At scale factor 0 all  $Pon_i'$  occur synchronously at the minimum of their originals (i.e. removed chord spread). At factor .5 a diminished chord spread will result, and at factor 2 an exaggerated chord spread can be obtained. At negative values of this factor the spread is inverted: first notes becoming last and vice versa. At extreme values of the scale factor the notes are restrained from moving out of the chord structure into the next musical object by truncation. Note that the whole operation is independent of score times.



### Scaling the expressive asynchrony of an ACCIA section

The scaling of the expressive asynchrony of a collateral simultaneous structure works as follows. Assume the structure has a main component with performance onset time  $Pon_m$  and an ornament component with performance onset time  $Pon_o$ . A time-map of the structure contains  $Pon_o$  and  $Pon_m$ . The scale operation on such a map delivers new performance onsets according to the following rules:

Let the performance onset asynchrony  $\Delta Pon$  be defined as:

$$\Delta Pon = Pon_o - Pon_m$$

The asynchrony is scaled by a multiplication factor  $f$ , and a new performance onset  $Pon_o'$  is calculated:

$$\Delta Pon' = \Delta Pon * f$$

$$Pon_o' = Pon_m + \Delta Pon'$$

$$Pon_m' = Pon_m$$

*Frame 9. Scaling the expressive asynchrony of an ACCIA section.*

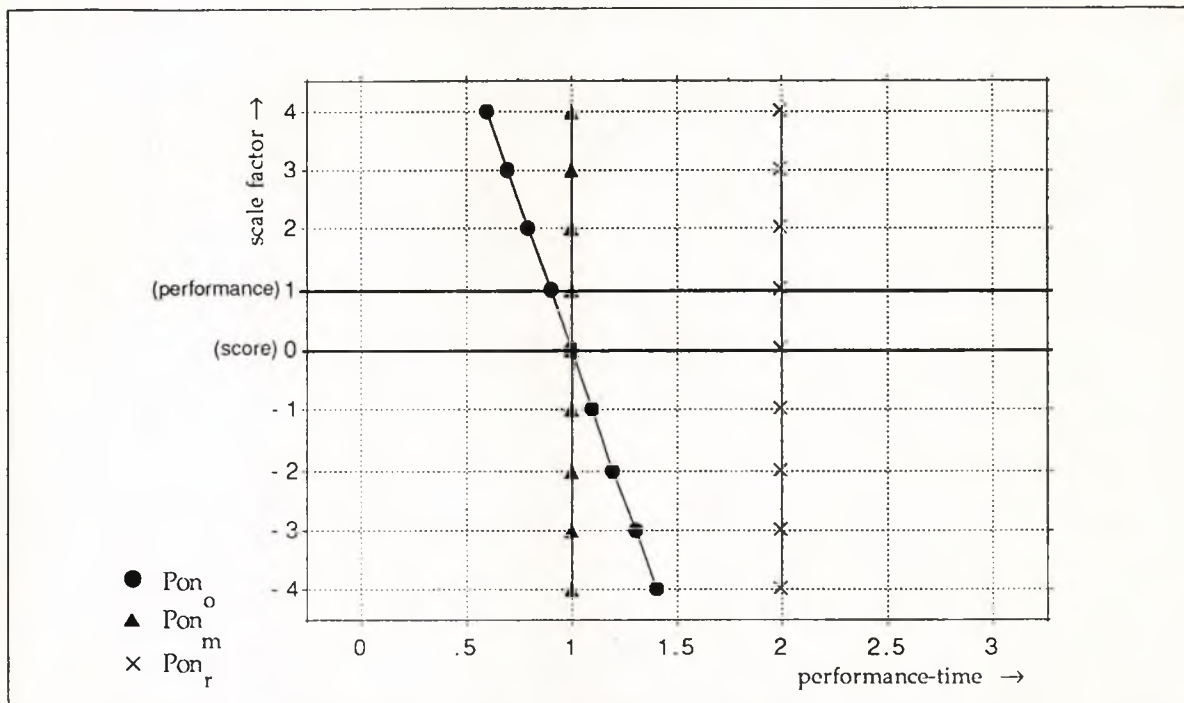


Figure 9. Scaling expressive timing of a ACCIA section. It shows this process for a specific set of performance times (a note preceded by an acciaccatura). At scale factor 1 all performance onsets are identical to their original. At scale factor 0 the ornament occurs synchronously with the main note (removed asynchronously). At negative values of this factor the order of onset of ornament and main note is inverted. Note that the ornament is allowed to shift freely - even outside the bounds of the whole ACCIA structure.

### Scaling expressive articulation

The articulation of a note is interpreted (scaled) relative to the articulation of the structure that it forms part of. For multilateral structures this is the average articulation. If thus the first note in a bar is played with more overlap than the other notes, a removal of the overlap articulation expression (a zero scale factor) will set the overlap of all notes to the mean overlap of the notes in the structure. And exaggerating the articulation expression (a scale factor larger than 1) will move the individual overlaps away from the mean - but maintaining the average overlap of all the notes in the bar. Of course all articulation types maintain reasonable performance offsets in the case of extreme values (i.e. note offsets will not shift before their onsets).

### Scaling the expressive articulation of a multilateral section

Assume a multilateral structure has  $n$  components  $C_i$  with  $0 \leq i \leq n-1$ . Component  $C_i$  has articulation  $A_i$  (see frame 5 for the calculation of  $A_i$ ). A section of the expression map of the structure contains all  $A_i$ . The articulation  $A$  of the structure itself is defined as:

$$A = \text{MEAN}_{0 \leq i \leq n-1} A_i$$

Let the expression deviations be

$$\Delta A_i = A_i - A \text{ for } 0 \leq i \leq n-1$$

The deviations are simply scaled by a multiplication factor  $f$

$$\Delta A_i' = f * \Delta A_i$$

The scale transformation delivers new articulations  $A_i'$  by adding the new deviations to the reference articulation  $A$  such that the articulation of the whole structure is kept invariant (mean  $A_i' = A$ ).

$$A_i' = A + \Delta A_i'$$

Keeping the expressive values in a reasonable range can only be done while applying them to the individual notes.

*Frame 10. Scaling the expressive articulation of a multilateral section*



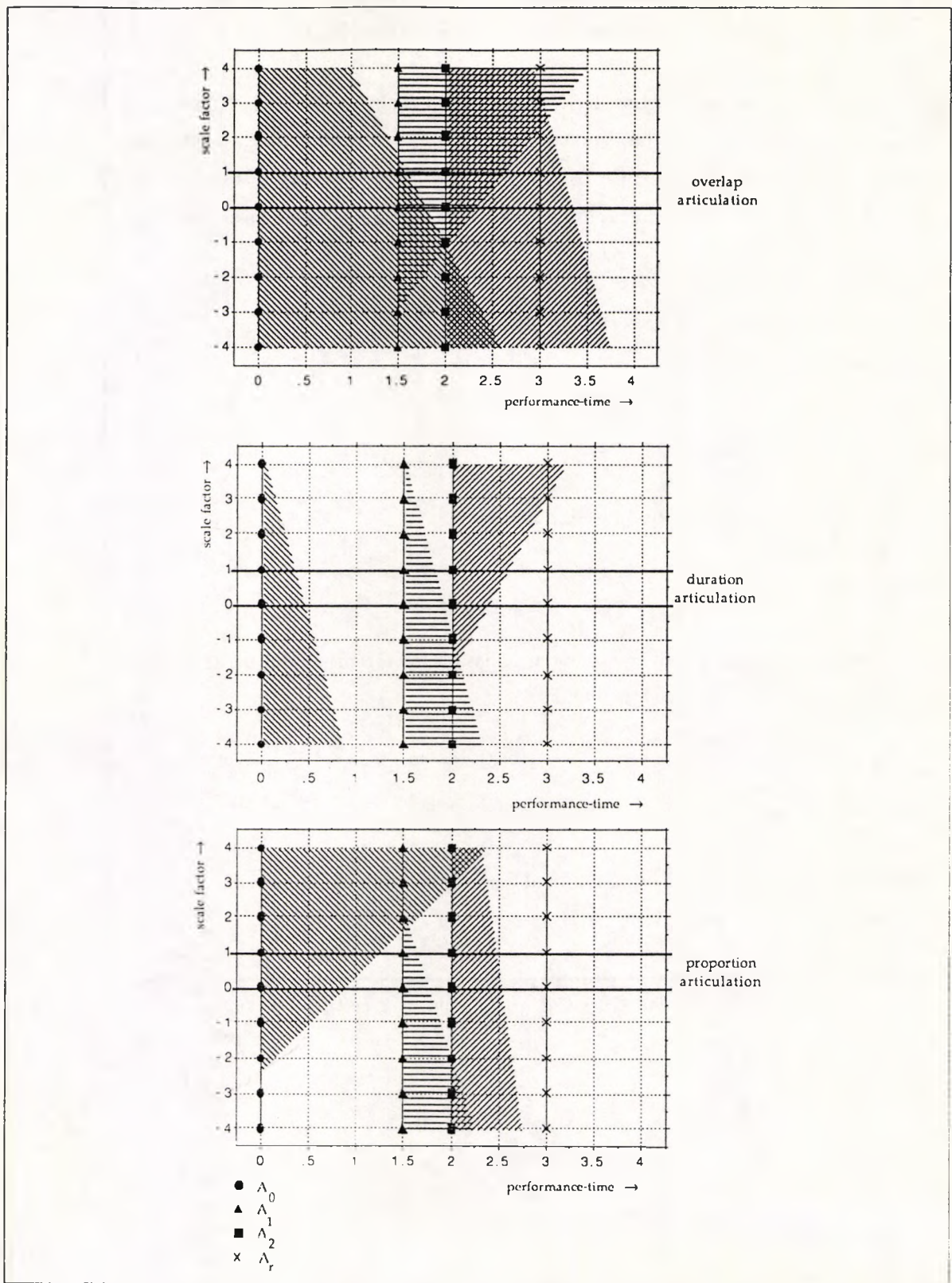


Figure 10. Scaling of an S section with three different kinds of articulation. It shows the scaling of three types of articulation for a multilateral structure, in this instance an S structure with a specific set of performance onset and offset times. Here, at scale factor 1 articulations  $A_i$  are identical to the original performance. At scale factor 0 all  $A_i$  are scaled to the mean articulation A. At a scale factor above 1 the deviation of each  $A_i$  with respect to A is exaggerated, with negative values constituting an inverse deviation: legato notes become more staccato and vice versa. Note that the mean articulation A is always kept invariant.



### Scaling the expressive articulation of a collateral section

Assume that a collateral structure has ornament and main components  $C_o$  and  $C_m$ . Component  $C_o$  has articulation  $A_o$  and component  $C_m$  has articulation  $A_m$  (see frame 5 for the calculation of  $A_o$  and  $A_m$ ). A section of the expression map of the structure contains these values. The articulation  $A$  of the structure itself is defined as:

$$A = A_m$$

Let the expression deviation be

$$\Delta A = A_o - A$$

The deviation is scaled by a multiplication factor  $f$

$$\Delta A' = f * \Delta A$$

The scale transformation delivers a new articulation for the ornament by adding the new deviation to the reference articulation  $A$ .

$$A_o' = A + \Delta A'$$

$$A_m' = A_m$$

Keeping the expressive values in a reasonable range can only be done while applying them to the individual notes.

*Frame 11. Scaling the expressive articulation of a collateral section.*

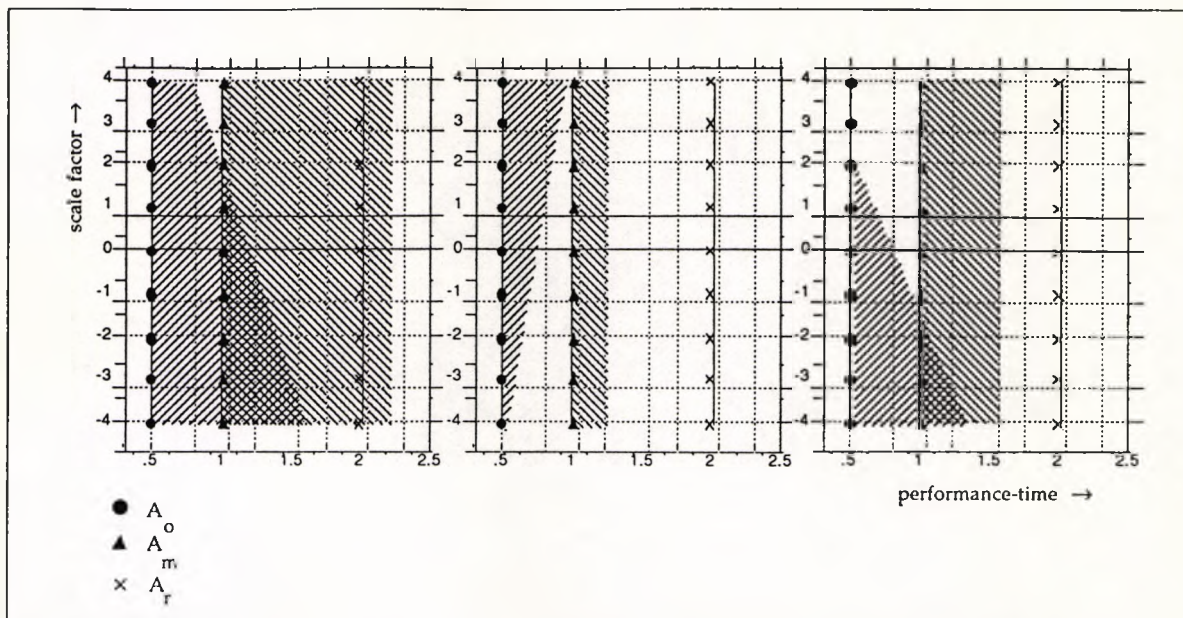


Figure 11. Scaling of an APPOG section with three different kinds of articulation. It shows three types of articulation scaling for an ornament (here an APPOG structure). At scale factor 1 the articulation  $A_o$  is identical to the original articulation of the ornament. At scale factor 0  $A_o$  is identical to the articulation of the main component  $A_m$ . At a scale factor above 1 the deviation of  $A_o$  with respect to the main component  $A_m$  is exaggerated, negative values constituting an inverse articulation: legato ornament articulation become more staccato and vice versa.

#### Keeping articulation consistent in the scaling of expressive timing

In the scaling of timing of onsets we ignored the influence it should have on its offsets. To obtain some sort of articulation consistency we can use the three types of articulation (as described above) when scaling expressive tempo and expressive asynchrony. In figure 12, we use expressive tempo scaling for an S section as an example in illustrating the different types of articulation consistency.



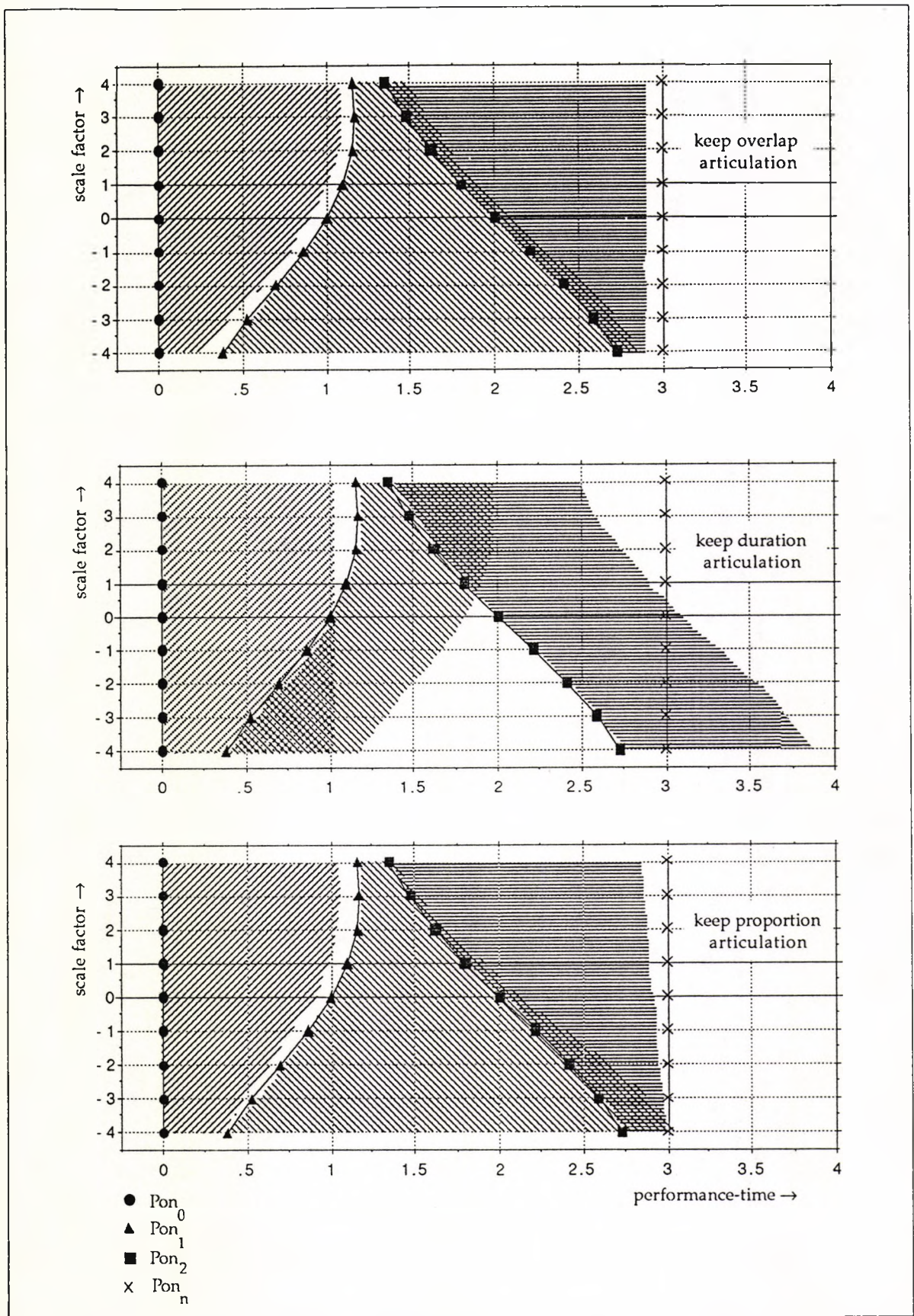


Figure 12. Scaling of an S section that keeps a particular type of articulation consistent. Shown for the same set of performance onsets as used in figure 6.



## Stretch maps

Sometimes it is useful to be able to keep a consistency in performance timing between voices when modifying one of them. Naming the modified material as the foreground and describing the rest as the background, the consistency requires that a series of performance onsets, at a selected background level, that happen between two performance onsets in the foreground are "stretched along" with the changes in the foreground. This feature is implemented by first extracting a timing map from the background, and "stretching" this map between the old and modified foreground map before it is reapplied to the background. The fore- and background must be parallel (must happen during the same score time interval) and have to be S structures. Maintaining the consistency between other kinds of structure remains a problem.

## Interpolate maps

A more sophisticated notion of expression entails the difference in expression between two structured objects. The best known example is voice leading in ensemble playing (Rasch, 1979) whereby the leading instrument often takes a small but consistent timing lead (around 10 ms). Inter- or extrapolation between two extracted timing maps yields the possibility to scale this kind of expression.

## Transfer maps

Sometimes it is useful to apply an expression map extracted from one object, to another object, possibly with a different structure, e.g. boldly applying the expressive timing of the melody to the accompaniment. This is supported via an operation on timing maps that uses the structure of one map but imposes expressive values of the other.

## TRANSFORMATIONS

Transformations of musical structures are generalizations of the operations on expression maps. They handle the selection of a level of structural description, extract a map, do the operation and re-impose the map. However, they often become quite sophisticated because they also take care of maintaining consistency with a background (material that is not affected directly). The application of the modified map has its own complexity, whereby changes are propagated to lower levels depending on the types of musical structure encountered. Finally, in the setting of new performance onsets of the notes, also the offsets may change in order to keep the articulation invariant. Out of the wealth of possibilities we choose some examples to be illustrated further by means of figures. In the figures the performance onsets and/or offsets of the individual notes at different parameter settings are given. The structure of the musical objects transformed are shown underneath.



In the following examples the same performance of a Beethoven theme is used (the fragment as shown in figure 2), allowing for comparison of the different transformations and to see the effect of applying the same transformation to different levels or types of structure. Note that for all the transformations the identity transformation is shown at scale factor 1.

## Scale timing

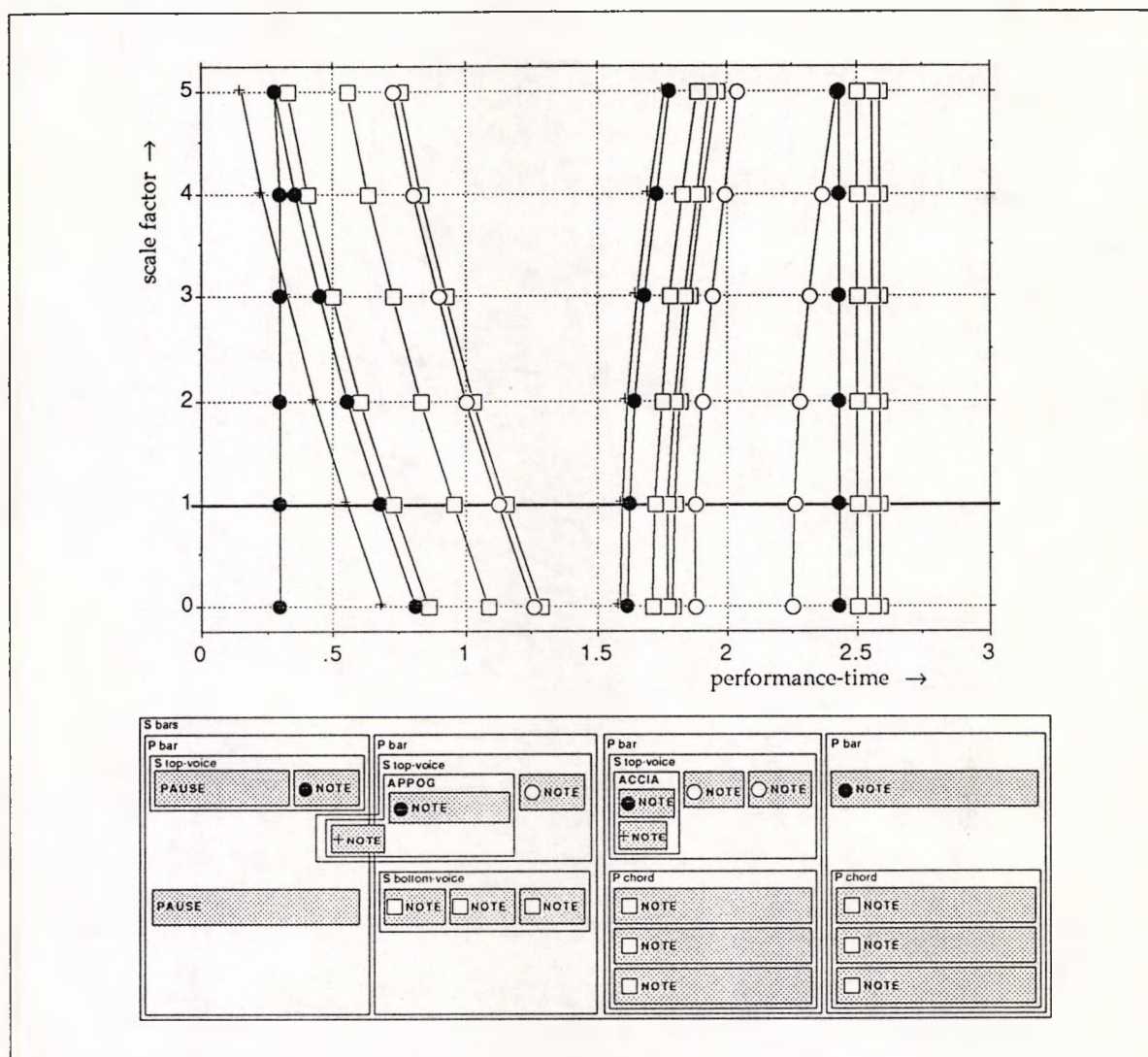


Figure 13. Scaling the expressive tempo of bars in the Beethoven fragment. Underneath the figure a structural description of the fragment is shown in bars. Imagine what would happen if we asked a performer to emphasize his/her timing of the bars? One possibility would be to play the onsets of the bars, that were played slightly early, even earlier, and ones that were played late, later still. This particular transformation can be read from figure 12 as the lines with the black markers, indicating the component in the bar that carries the expressive timing. Both the performance onset of the first and the last bar of the enclosing bars' structure are not changed; the transformation is done at the level named "bars", with its timing kept invariant. The lines with white markers show the embedded material that follows the change of the performance onset of each bar. Note that the timing of the ornamented notes does not change (they keep the same distance with respect to the note they cling to), as does the spread of the chords.

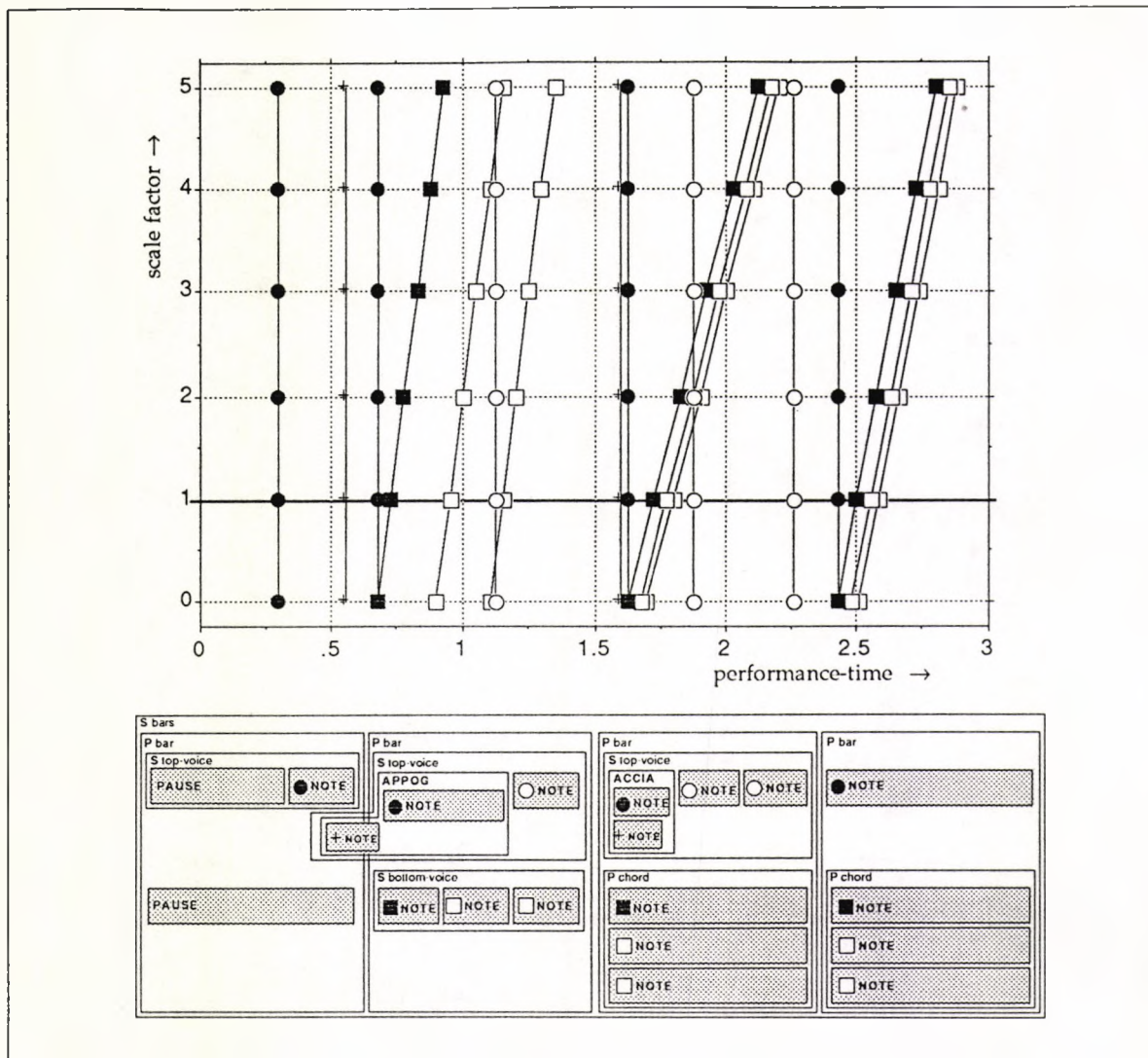


Figure 14. Scaling the expressive asynchrony of each bar in the Beethoven fragment. It shows the expressive transformation we might expect to happen when a performer is asked to exaggerate the asynchrony between the top-voice and bottom-voice at the onset of each bar. The figure shows the scaling of the asynchrony of the bottom voice onsets (the black squares), without changing the timing of the bars (lines marked with black circles and triangles). The embedded notes of the bottom voice (lines with white squares) just shift along with the expressive timing of their embedding structure. Here again, the ornament timing and the chord spread stay invariant.



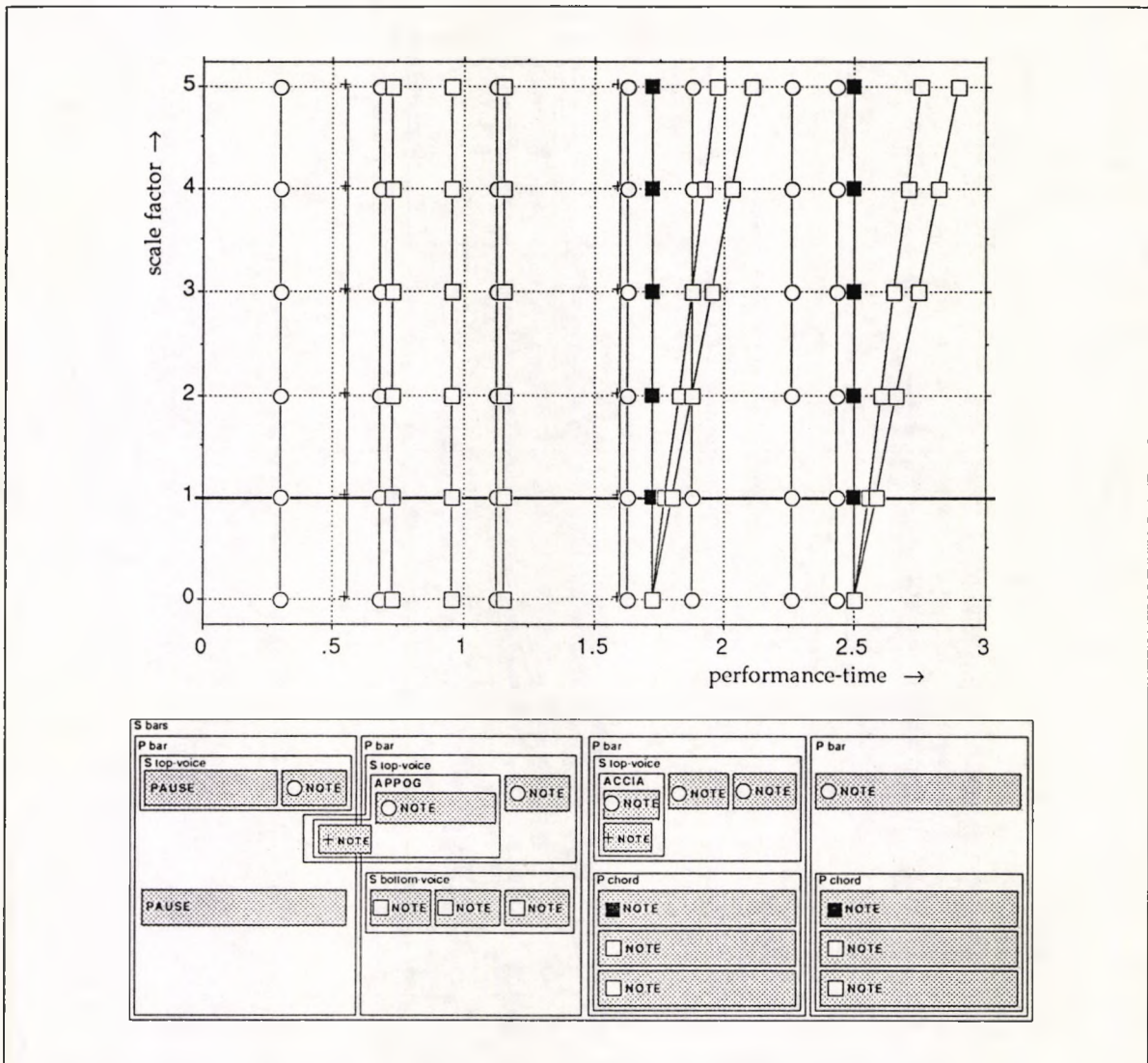


Figure 15. Scaling the expressive asynchrony of each chord in the Beethoven fragment. It shows another expressive transformation that exaggerates the chord spread, turning them almost into arpeggio's at high scale factors. At scale factor 0 the chord spread is completely removed. The timing of the rest of the fragment stays unaltered.



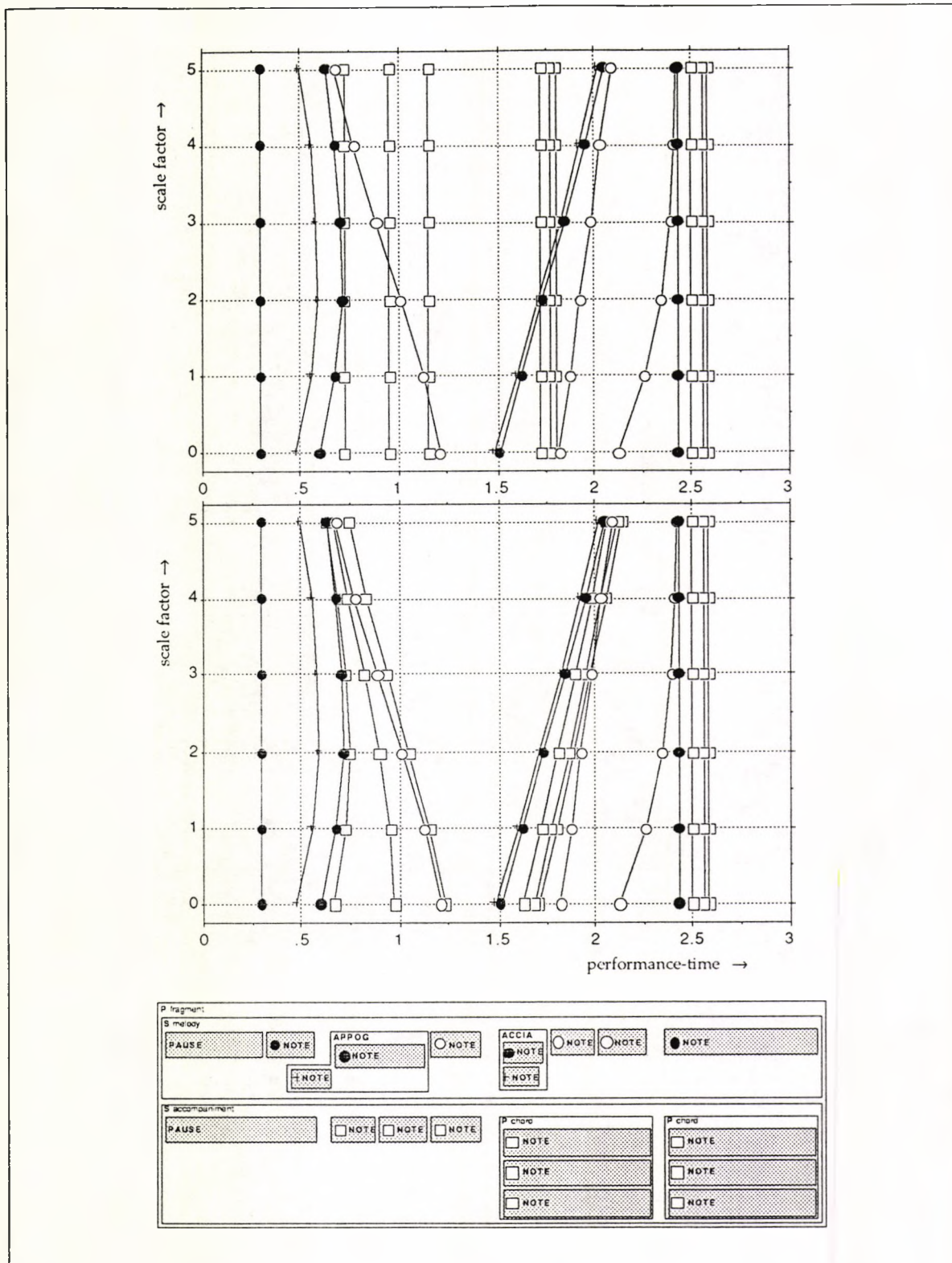


Figure 16. Scaling the expressive tempo of the melody in the Beethoven fragment, a) without and b) with "stretching" the accompaniment. It shows that the timing of each note of the melody becomes exaggerated with a higher scale factor. Here the accompaniment (lines marked with white squares) is not affected at all. Figure 16b, on the other hand, shows a musically more reasonable transformation: the accompaniment follows the movements of the transformed melody, e.g. slowing down when the tempo of the melody slows down. Here the accompaniment is kept consistent with respect to the original performance (compare with the onsets at scale factor 1). Note that note order can change between melody and accompaniment, because of the structural description in two parallel voices.

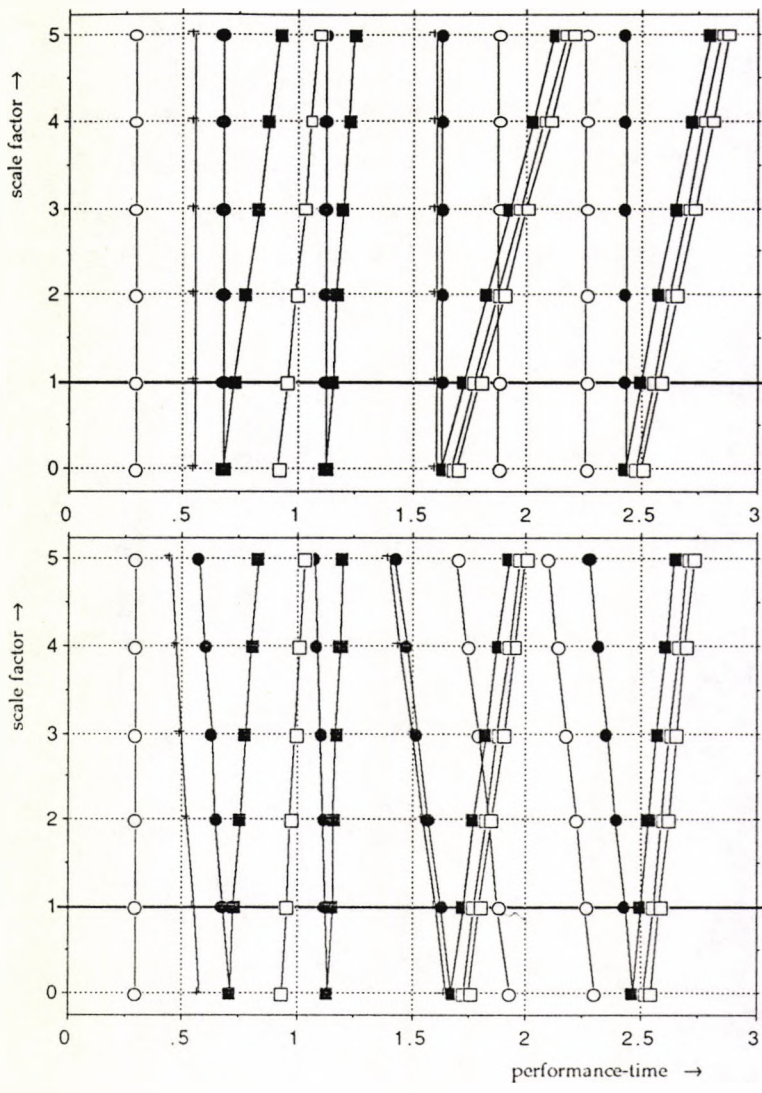
### **Keeping articulation consistent**

In the above examples we showed the scaling of onset times and neglected what happened to the offset times. But, as we showed before, this cannot just be ignored in musically relevant transformations. We can select one of the described types of articulation to keep consistent, but we do not show this here (see figure 12 for a simple example).

### **Scale intervoice expression**

When the expression between voices is scaled, two parameters are used. The first one selects a reference level of expression (0 designates the expression of the first, 1 designates the expression of the second, 0.5 is the mean of the two etc.). The second parameter determines in how far the voices are removed from that reference level (0 means completely on reference level, 1 means as in original performance, 2 means exaggerated with respect to the reference etc.).





P segment:									
S melody									
PAUSE	<input type="radio"/> NOTE	APPOG	<input checked="" type="radio"/> NOTE	<input checked="" type="radio"/> NOTE	ACCIA	<input checked="" type="radio"/> NOTE	<input type="radio"/> NOTE	<input type="radio"/> NOTE	<input checked="" type="radio"/> NOTE
		<input checked="" type="radio"/> NOTE			<input checked="" type="radio"/> NOTE				<input checked="" type="radio"/> NOTE
S accompaniment									
PAUSE	<input checked="" type="checkbox"/> NOTE	<input type="checkbox"/> NOTE	<input checked="" type="checkbox"/> NOTE	P chord			P chord		
				<input checked="" type="checkbox"/> NOTE	<input type="checkbox"/> NOTE	<input type="checkbox"/> NOTE	<input checked="" type="checkbox"/> NOTE	<input type="checkbox"/> NOTE	<input type="checkbox"/> NOTE
				<input type="checkbox"/> NOTE	<input type="checkbox"/> NOTE	<input type="checkbox"/> NOTE	<input type="checkbox"/> NOTE	<input type="checkbox"/> NOTE	<input type="checkbox"/> NOTE

Figure 17. Scaling the intervoice timing between the melody and the accompaniment in the Beethoven fragment, a) with the melody as reference, and b) with the mean of the melody and the accompaniment as reference. In figure 17a intervoice timing (one type of intervoice scaling) is scaled with the melody voice as the reference. It shows the scaling of the asynchrony between the accompaniment and the melody, as found in the performance (see the horizontal line where the scale factor is 1). Notes that are not synchronous (i.e. don't have the same score time) interpolate their change with respect to their starting performance onsets that are considered parallel (have the same score time). Note that the timing of the melody does not change because it is used as reference. In figure 17b the mean of the melody and accompaniment timing is used as reference, resulting in displacements (with respect to this invisible reference) of both voices. In both figures, the first event in the melody voice is unaffected since there is no measurable timing in the accompaniment (only a rest).

## CONCLUSION

In this paper we have presented a proposal for a calculus that enables expressive timing to be transformed on the basis of structural aspects. The program implementing the calculus, will hopefully prove to be a solid basis for formalised theories of music cognition. A micro version of this program is included in the appendix, open to further inquiry and immediate test. The proposed representation constructs allow for easy maintenance and extension. An object-oriented programming style proved a good choice for this kind of modelling. The algorithmic parts became reasonably simple, but the program can still be considered as quite complex, especially its elaborate knowledge representation. This algorithmic simplicity combined with structural complexity mirrors, in this respect, the widespread hypothesis that the complex expressive timing profiles found in musical performances are more readily explained as the product of a small collection of simple rules linked to a relatively complex structure, than as the result of a large collection of interacting rules, with hardly any structure.

This research again confirmed that music is a very rewarding field for experimentation with knowledge representation concepts.

## ACKNOWLEDGEMENTS

We are very grateful to Eric Clarke who made it possible for us to work for two years on research in expressive timing at City University in London, and the British ESRC for their financial support (grant A413254004) during this period.

## REFERENCES

- Bentley J. (1988) More Programming Pearls, Confessions of a Coder. Reading, MA: Addison-Wesley.
- Bregman, A. S. (1990) Auditory Scene Analysis: The Perceptual Organization of Sound. Cambridge, Mass.: Bradford books, MIT Press.
- Clarke, E. F. (1988) Generative principles in music performance. In J. A. Sloboda (Ed.) Generative processes in music. The psychology of performance, improvisation and composition. Oxford: Science Publications.
- Desain, P. & H. Honing (1988) LOCO: A Composition Microworld in Logo. Computer Music Journal 12(3): 30-42.
- Desain, P. & H. Honing (1991) Quantization of Musical Time: A Connectionist Approach. In P. M. Todd & G. J. Loy (Eds.) Music and Connectionism. Cambridge, Mass.: MIT Press.
- Desain, P. & H. Honing (in press, a) Tempo curves considered harmful. To appear in Contemporary Music Review.



- Desain, P. & H. Honing (in press, b) Time functions function best as functions of multiple times.  
To appear in Computer Music Journal.
- Desain, P. (1990) Lisp as a second language. Perspectives of New Music 28(1).
- Honing, H. (1990) POCO: An Environment for Analysing, Modifying, and Generating Expression in Music. In Proceedings of the 1990 International Computer Music Conference. San Francisco: Computer Music Association.
- Honing, H. (1991) Issues in the Representation of Time and Structure in Music. In Proceedings of the 1990 Music and the Cognitive Sciences Conference, edited by I. Cross and I. Deliège. Contemporary Music Review. London: Harwood Press. (forthcoming).
- Keene, S. E. (1989) Object-Oriented Programming in Common Lisp: A Programmer's Guide to CLOS. Reading, MA: Addison-Wesley.
- Longuet-Higgins, H. C. (1976) The Perception of Melodies. Nature 263: 646-653.
- Rasch, R. A. (1979) Synchronisation in performed ensemble music. Acoustica 43, 121-131.
- Serafine, M.L. (1988) Music as Cognition: The Development of Thought in Sound. New York: Columbia University Press.
- Steele, G. L. (1990) Common Lisp, the Language. Second edition. Bedford, MA: Digital Press.
- Vos, J. & R. A. Rasch (1981) The perceptual onset of musical tones. Perception & Psychophysics 29(4): 323-335.

## MICROWORLD EXPRESSION CALCULUS

```

;*****
;* A CALCULUS FOR MUSIC PERFORMANCE EXPRESSION *
;* (c) 1991, Peter Desain & Henkjan Honing *
;* *
;* in CLOS (Common Lisp), uses loop macro *
;*****

;*****
;*****
; MUSICAL OBJECTS
;*****
;*****
; abstract classes of musical objects

(defclass musical-object ()
  ((name :reader name :initarg :name :initform 'no-name :type symbol)
   (score-onset :reader score-onset :type rational :initform 0)
   (left :reader left :initform nil)
   (right :reader right :initform nil))
  (:documentation "Musical Object"))

(defclass structured (musical-object)
  ((score-offset :reader score-offset :type rational))
  (:documentation "Structured Musical Object"))

(defclass multilateral (structured)
  ((components :reader components :initarg :components))
  (:documentation "Multilateral Musical Object"))

(defclass collateral (structured)
  ((main :reader main :initarg :main)
   (ornament :reader ornament :initarg :ornament))
  (:documentation "Ornamented Musical Object"))

(defclass successive (structured)
  ()
  (:documentation "Successive Musical Object"))

(defclass simultaneous (structured)
  ()
  (:documentation "Simultaneous Musical Object"))

(defclass basic (musical-object)
  ((score-offset :reader score-offset :type rational :initarg :score-dur))
  (:documentation "Basic Musical Object"))

;*****
; instantiatable classes of musical objects

(defclass S (multilateral successive) () (:documentation "Sequential"))
(defclass P (multilateral simultaneous) () (:documentation "Parallel"))
(defclass ACCIA (collateral simultaneous) () (:documentation "Acciaccature"))
(defclass APPOG (collateral successive) () (:documentation "Appoggiature"))

(defclass NOTE (basic)
  ((dynamic :accessor dynamic :type float :initarg :dynamic)
   (perf-onset :accessor perf-onset :type float :initarg :perf-onset :initform nil)
   (perf-offset :accessor perf-offset :type float :initarg :perf-offset :initform nil))
  (:documentation "Note"))

(defclass PAUSE (basic) () (:documentation "Rest"))

```

```

;*****
; creators for musical objects

(defun S (name &rest components)
  (make-instance 'S :name name :components components))

(defun P (name &rest components)
  (make-instance 'P :name name :components components))

(defun ACCIA (name ornament main)
  (make-instance 'ACCIA :name name :ornament ornament :main main))

(defun APPOG (name ornament main)
  (make-instance 'APPOG :name name :ornament ornament :main main))

(defun NOTE (&key name perf-onset perf-offset score-dur (dynamic 1))
  (make-instance 'NOTE :name name
                  :perf-onset perf-onset
                  :perf-offset perf-offset
                  :score-dur score-dur
                  :dynamic dynamic))

(defun PAUSE (&key name score-dur)
  (make-instance 'PAUSE :name name :score-dur score-dur))

;*****
; extra access functions for musical objects

(defmethod components ((object basic)) nil)
(defmethod components ((object collateral))
  (list (ornament object) (main object)))

(defmethod all-notes ((object musical-object))
  (loop for component in (components object) append (all-notes component)))

(defmethod all-notes ((object note)) (list object))

(defun has-name? (&rest names)
  #'(lambda (object &rest ignore) (member (name object) names)))

(defmethod find-parts ((object musical-object) pred)
  (if (funcall pred object)
      (list object)
      (loop for component in (components object)
            append (find-parts component pred))))

;*****
; initialization of score times and context of musical objects

(defmethod initialize-instance :after ((object musical-object) &rest ignore)
  (object-check object)
  (initialize-score-times object)
  (initialize-context object))

(defmethod object-check ((object musical-object)) nil)

;*****
; initialization of score-onset and offset of musical objects

(defmethod initialize-score-times ((object basic)))

(defmethod initialize-score-times ((object P))
  (setf (slot-value object 'score-offset)
        (slot-value (first (components object)) 'score-offset)))

```

```

(defmethod initialize-score-times ((object S))
  (loop with onset = 0
    for component in (components object)
    do (shift-score component onset)
    (setf onset (slot-value component 'score-offset))
    finally (setf (slot-value object 'score-offset) onset)))

(defmethod initialize-score-times ((object collateral))
  (setf (slot-value object 'score-offset)
    (slot-value (main object) 'score-offset)))

(defmethod initialize-score-times :after ((object APPOG))
  (shift-score (ornament object)
    (- (slot-value (ornament object) 'score-offset))))

(defmethod shift-score ((object musical-object) shift)
  (incf (slot-value object 'score-onset) shift)
  (incf (slot-value object 'score-offset) shift)
  (loop for component in (components object) do (shift-score component shift)))

;*****
; initialization of context of musical objects

(defmethod initialize-context ((object musical-object))

(defmethod initialize-context ((object S))
  (loop for component in (components object)
    for next-component in (rest (components object))
    do (set-contexts component next-component)))

(defmethod initialize-context ((object APPOG))
  (set-context (ornament object) (main object) 'right))

(defmethod set-contexts ((left musical-object) (right musical-object))
  (set-context left right 'right)
  (set-context right left 'left))

(defmethod set-context ((object musical-object) (context musical-object) dir)
  (setf (slot-value object dir) context))

(defmethod set-context :after ((object P) (context musical-object) dir)
  (loop for component in (components object)
    do (set-context component context dir)))

(defmethod set-context :after ((object S) (context musical-object) dir)
  (if (eql dir 'left)
    (set-context (first (components object)) context dir)
    (set-context (last-element (components object)) context dir)))

(defmethod set-context :after ((object collateral) (context musical-object) dir)
  (set-context (main object) context dir))

(defmethod set-context :after ((object ACCIA) (context musical-object) dir)
  (when (eql dir 'left)
    (set-context (ornament object) context dir)))

```



```

;*****
;*****
; MAPS
;*****
;*****
; abstract classes of maps

(defclass map ()
  ((sections :accessor sections :initarg :sections)
   (:documentation "Expression Map"))

(defclass multilateral-map (map) ())
(defclass collateral-map (map) ())
(defclass simultaneous-map (map) ())
(defclass successive-map (map) ())

;*****
; instantiable classes of maps

(defclass P-map (multilateral-map simultaneous-map) ())
(defclass S-map (multilateral-map successive-map) ())
(defclass ACCIA-map (collateral-map simultaneous-map) ())
(defclass APPOG-map (collateral-map successive-map) ())

;*****
; creator for maps

(defun make-map (sections)
  (let ((ordered-sections (sort sections #'< :key #'score-onset)))
    (cond ((null ordered-sections) nil)
          ((and (same-section-type? ordered-sections)
                (not-overlapping? ordered-sections))
           (make-instance (section-to-map (first ordered-sections))
                          :sections ordered-sections))
          (t (error "attempt to merge incompatible sections into expression map")))))

;*****
; sections of maps
;*****
; abstract classes of sections of maps

(defclass section ()
  ((all-score-times :accessor all-score-times :initarg :all-score-times)
   (all-expressions :accessor all-expressions :initarg :all-expressions))
  (:documentation "Expression Section"))

(defclass multilateral-section (section) ())
(defclass collateral-section (section) ())
(defclass successive-section (section) ())
(defclass simultaneous-section (section) ())

;*****
; instantiable classes of sections of maps

(defclass S-section (successive-section multilateral-section) ())
(defclass P-section (simultaneous-section multilateral-section) ())
(defclass ACCIA-section (simultaneous-section collateral-section) ())
(defclass APPOG-section (successive-section collateral-section) ())

;*****
; compatibility relation between musical objects, expression maps and sections thereof

(defmethod object-to-section ((object musical-object))
  (third (find (class-name (class-of object)) (object-network) :key #'first)))

(defmethod section-to-map ((section section))
  (second (find (class-name (class-of section)) (object-network) :key #'third)))

```

```

(defun object-network ()
  '(S S-map S-section)
  (P P-map P-section)
  (ACCIA ACCIA-map ACCIA-section)
  (APPOG APPOG-map APPOG-section)))

;*****
; creators for sections of maps

(defun make-section (section-class all-score-times all-expressions)
  (make-instance section-class
    :all-score-times all-score-times
    :all-expressions all-expressions))

(defmethod make-new-section ((section section) expressions)
  (make-section (class-of section)
    (snoc (score-times section) (score-offset section))
    (snoc expressions (next-expression section))))

(defmethod make-new-section-from-pairs ((section section) pairs)
  (make-section (class-of section)
    (snoc (mapcar #'first pairs) (score-offset section))
    (snoc (mapcar #'second pairs) (next-expression section))))

;*****
; extra accessors for sections of maps

(defmethod score-onset ((section section))
  (first (all-score-times section)))

(defmethod score-offset ((section section))
  (last-element (all-score-times section)))

(defmethod expressions ((section section))
  (butlast (all-expressions section)))

(defmethod next-expression ((section section))
  (last-element (all-expressions section)))

(defmethod score-times ((section section))
  (butlast (all-score-times section)))

(defmethod score-onset ((section collateral-section))
  (score-main section))

(defmethod main-expression ((section collateral-section))
  (second (all-expressions section)))

(defmethod ornament-expression ((section collateral-section))
  (first (all-expressions section)))

(defmethod score-main ((section collateral-section))
  (second (all-score-times section)))

(defmethod score-ornament ((section collateral-section))
  (first (all-score-times section)))

(defun same-section-type? (sections)
  (every #'(lambda (section) (class-of section)) sections))

(defun not-overlapping? (sections)
  (loop for section in sections
    for next-section in (rest sections)
    never (> (score-offset section) (score-onset next-section))))

```

```

;*****
; find section (containing score time) in expression map

(defmethod lookup-section-containing ((map map) score-time)
  (loop for section in (sections map)
        when (<= (score-onset section) score-time (score-offset section))
        do (return section)))

;*****
; lookup expression value (via score time) in expression map

(defmethod lookup-defined-expression ((map map) score-time)
  (lookup-defined-expression (lookup-section-containing map score-time) score-time))

(defmethod lookup-defined-expression (section score-time)
  (and section
    (loop for expression in (all-expressions section)
          for map-score-time in (all-score-times section)
          when (= map-score-time score-time)
          do (return expression))))

(defmethod lookup-expression ((map successive-map) score-time)
  (lookup-expression (lookup-section-containing map score-time) score-time))

(defmethod lookup-expression (section score)
  (and section
    (loop for expression in (all-expressions section)
          for expression-next in (rest (all-expressions section))
          for score-time in (all-score-times section)
          for score-time-next in (rest (all-score-times section))
          while (> score score-time-next)
          finally (return (interpolate score-time score score-time-next
                                       expression expression-next))))))

;*****
; lookup score time in a monotone rising expression map

(defmethod in-section-inverse? ((section section) expression)
  (and expression (<= (first (expressions section))
                     expression
                     (or (next-expression section)
                         (last-element (expressions section))))))

(defmethod lookup-inverse ((map S-map) expression)
  (loop for section in (sections map) thereis (lookup-inverse section expression)))

(defmethod lookup-inverse ((section section) expression)
  (and (in-section-inverse? section expression)
    (loop for expression-next in (rest (expressions section))
          for score-time in (score-times section)
          for score-time-next in (rest (score-times section))
          while (> expression expression-next)
          finally (return (list score-time score-time-next)))))

;*****
; mapping through expression maps

(defmethod map-map (fun (map map))
  (make-map (loop for section in (sections map) collect (funcall fun section))))

;*****
; mapping through filtered expression maps

(defmethod with-filtered-null-expression (fun (map map))
  (unfilter-null-expression (funcall fun (filter-null-expression map))
    (filter-null-expression-out map)))

(defmethod filter-null-expression ((map map))
  (map-map #'filter-null-expression map))

```

```

(defmethod filter-null-expression ((section section))
  (make-new-section-from-pairs section
    (loop for expression in (expressions section)
          for score-time in (score-times section)
          when expression
          collect (list score-time expression))))

(defmethod filter-null-expression-out ((map map))
  (mapcar #'filter-null-expression-out (sections map)))

(defmethod filter-null-expression-out ((section section))
  (loop for expression in (expressions section)
        for score-time in (score-times section)
        for index from 0
        unless expression
        collect (list index score-time)))

(defmethod unfilter-null-expression ((map map) rejections)
  (make-map (mapcar #'unfilter-null-expression (sections map) rejections)))

(defmethod unfilter-null-expression ((section section) removed)
  (if removed
    (make-new-section-from-pairs section
      (loop with expressions = (expressions section)
            with score-times = (score-times section)
            for index from 0
            while (or score-times removed)
            when (and removed (= index (caar removed)))
            collect (list (second (pop removed)) nil)
            else collect (list (pop score-times)
                               (pop expressions))))
    section))

```



```

;*****
;*****
; EXPRESSION
;*****
;*****

(defclass expression ())

;*****
; nil and rests carry no expression, nil expressions and sections are not set

(defmethod get-expression ((object null) (expression expression)) nil)
(defmethod get-next-expression ((object null) (expression expression)) nil)

(defmethod get-expression ((object PAUSE) (expression expression)) nil)
(defmethod set-expression ((object PAUSE) (expression expression) value) nil)

(defmethod set-expression ((object musical-object) expression value-or-section) nil)
(defmethod get-next-expression ((object musical-object) (expression expression))
  (get-expression (right object) expression))

;*****
; get expression of notes

(defmethod get-notes-expression ((object musical-object) (expression expression))
  (loop for note in (all-notes object)
        collect (fetch-expression note expression)))

(defmethod set-notes-expression ((object musical-object) (expression expression) values)
  (loop for note in (all-notes object)
        for value in values
        do (set-expression note expression value)))

;*****
; propagate expression (interpolated, truncating-shift and shift)

(defmethod propagate-interpolated ((object S)
  old-begin new-begin old-end new-end expression)
  (loop for component in (components object)
        do (propagate-interpolated component
  old-begin new-begin old-end new-end expression)))

(defmethod propagate-interpolated ((object P)
  old-begin new-begin old-end new-end expression)
  (loop for component in (components object)
        do (propagate-truncating-shift component
  (save-- new-begin old-begin) new-end expression)))

(defmethod propagate-interpolated ((object collateral)
  old-begin new-begin old-end new-end expression)
  (let* ((ref (fetch-expression (main object) expression))
        (shift (save-- (interpolate old-begin ref old-end new-begin new-end) ref)))
    (propagate-interpolated (main object) old-begin new-begin old-end new-end expression)
    (propagate-shift (ornament object) shift expression)))

(defmethod propagate-interpolated ((object NOTE)
  old-begin new-begin old-end new-end expression)
  (set-expression
  object expression
  (interpolate old-begin (fetch-expression object expression)
  old-end new-begin new-end)))

(defmethod propagate-interpolated ((object PAUSE)
  old-begin new-begin old-end new-end expression))

```

```

;*****
; propagate-truncating-shift

(defmethod propagate-truncating-shift :around ((object musical-object)
                                             shift end expression)
  (when shift (call-next-method)))

(defmethod propagate-truncating-shift ((object multilateral) shift end expression)
  (loop for component in (components object)
        do (propagate-truncating-shift component shift end expression)))

(defmethod propagate-truncating-shift ((object collateral) shift end expression)
  (propagate-shift (ornament object) shift expression)
  (propagate-truncating-shift (main object) shift end expression))

(defmethod propagate-truncating-shift ((object NOTE) shift end expression)
  (set-expression object
    expression
    (save-min (save+ (fetch-expression object expression) shift) end)))

(defmethod propagate-truncating-shift ((object PAUSE) shift end expression))

;*****
; propagate-shift

(defmethod propagate-shift :around ((object musical-object) shift expression)
  (when shift (call-next-method)))

(defmethod propagate-shift ((object structured) shift expression)
  (loop for component in (components object)
        do (propagate-shift component shift expression)))

(defmethod propagate-shift ((object basic) shift expression)
  (set-expression object
    expression
    (save+ (fetch-expression object expression) shift)))

;*****
; onset timing
;*****

(defclass expressive-timing (expression) ())
(defclass onset-timing (expressive-timing) ())
(defclass basic-asynchrony (onset-timing) ())
(defclass basic-tempo (onset-timing) ())

(defclass estimate-onset-timing (onset-timing estimate-mixin) ())

;*****
; get expressive timing

(defmethod get-expression ((object NOTE) (expression onset-timing))
  (perf-onset object))

(defmethod get-expression ((object S) (expression onset-timing))
  (get-expression (first (components object)) expression))

(defmethod get-expression ((object P) (expression onset-timing))
  (loop for component in (components object)
        when (get-expression component expression)
        minimize it))

(defmethod get-expression ((object collateral) (expression onset-timing))
  (get-expression (main object) expression))

```

```

;*****
; set expressive timing

(defmethod set-expression ((object NOTE) (expression onset-timing) value)
  (setf (perf-onset object) value))

(defmethod set-expression ((object S) (expression onset-timing) (section S-section))
  (loop for new-expression in (expressions section)
        for next-new-expression in (snoc (rest (expressions section))
                                         (next-expression section))
        for component in (components object)
        do (propagate-interpolated component
                                   (fetch-expression component expression)
                                   new-expression
                                   (fetch-expression (right component) expression)
                                   next-new-expression
                                   expression)))

(defmethod set-expression ((object P) (expression onset-timing) (section P-section))
  (loop for new-expression in (expressions section)
        for component in (components object)
        do (propagate-truncating-shift component
                                       (save-- new-expression
                                             (fetch-expression component expression))
                                       (get-next-expression object expression)
                                       expression)))

(defmethod set-expression ((object ACCIA)
                          (expression onset-timing)
                          (section ACCIA-section))
  (propagate-shift (ornament object)
                  (save-- (ornament-expression section)
                        (fetch-expression (ornament object) expression))
                  expression))

(defmethod set-expression ((object APPOG)
                          (expression onset-timing)
                          (section APPOG-section))
  (propagate-interpolated (ornament object)
                        (fetch-expression (ornament object) expression)
                        (ornament-expression section)
                        (fetch-expression (right (ornament object)) expression)
                        (main-expression section)
                        expression))

;*****
; scale expressive-timing

(defmethod scale-expression ((section P-section)
                           (expression basic-asynchrony)
                           factor)
  (if (expressions section)
      (make-new-section
       section
       (scale-P-expression-points (expressions section) factor))
      section))

(defmethod scale-expression ((section S-section)
                           (expression basic-tempo)
                           factor)
  (cond ((and (expressions section) (next-expression section))
        (scale-S-section-] section factor))
        ((rest (expressions section))
         (scale-S-section-> section factor))
        (t section)))

```



```

(defmethod scale-S-section] ((section section) factor)
  (make-new-section section (scale-S-expression-points
    (snoc (score-times section) (score-offset section))
    (snoc (expressions section) (next-expression section))
    factor)))

(defmethod scale-S-section-> ((section section) factor)
  (make-new-section section
    (scale-S-expression-points (score-times section)
      (expressions section)
      factor)))

(defmethod scale-expression ((section ACCIA-section)
  (expression basic-asynchrony) factor)
  (make-new-section section
    (scale-ACCIA-points (main-expression section)
      (ornament-expression section)
      factor)))

(defmethod scale-expression ((section APPOG-section) (expression basic-tempo) factor)
  (make-new-section section
    (scale-APPOG-points (ornament-expression section)
      (main-expression section)
      (next-expression section)
      (score-ornament section)
      (score-main section)
      (score-offset section)
      factor)))

;*****

(defun scale-P-expression-points (perf-onsets factor)
  (let* ((perf-begin (apply #'min perf-onsets))
    (perf-iois (mapcar #'(lambda (onset) (- onset perf-begin)) perf-onsets))
    (raw-new-perf-iois (mapcar #'(lambda (perf) (scale-expression-lin perf factor))
      perf-iois))
    (shift (- (apply #'min raw-new-perf-iois)))
    (new-perf-onsets (mapcar #'(lambda (ioi) (+ ioi shift perf-begin))
      raw-new-perf-iois)))
    new-perf-onsets))

(defun scale-S-expression-points (score-times perf-times factor)
  (let* ((perf-iois (mapcar #'- (rest perf-times) perf-times))
    (score-iois (mapcar #'- (rest score-times) score-times))
    (perf-begin (first perf-times))
    (perf-end (last-element perf-times))
    (raw-new-perf-iois (mapcar #'(lambda (score perf)
      (scale-velocity score perf factor))
      score-iois
      perf-iois))
    (new-perf-iois (normalise raw-new-perf-iois (- perf-end perf-begin)))
    (new-perf-times (integrate new-perf-iois perf-begin)))
    new-perf-times))

(defun scale-ACCIA-points (main-expression ornament-expression factor)
  (let* ((expression-interval (- main-expression ornament-expression))
    (new-expression-ornament (- main-expression
      (scale-expression-lin expression-interval factor))))
    (list new-expression-ornament main-expression)))

```



```

(defun scale-APPOG-points (ornament-expression main-expression next-expression
                          score-ornament score-main score-end
                          factor)
  (let* ((score-ornament-ioi (- score-main score-ornament ))
         (expression-ornament-ioi (- main-expression ornament-expression))
         (score-main-ioi (- score-end score-main))
         (expression-main-ioi (- next-expression main-expression))
         (ornament-tempo (/ score-ornament-ioi expression-ornament-ioi))
         (main-tempo (/ score-main-ioi expression-main-ioi))
         (relative-tempo (/ ornament-tempo main-tempo))
         (new-ornament-tempo (* main-tempo (expt relative-tempo factor)))
         (new-expression-ornament (/ score-ornament-ioi new-ornament-tempo))
         (new-expression-ornament-ioi (- main-expression new-expression-ornament-ioi)))
    (list new-expression-ornament main-expression next-expression)))

;*****
; expression scale methods

(defun scale-velocity (score perf factor)
  "Exponential scaling"
  (/ score (expt (/ score perf) factor)
    ))

(defun scale-expression-lin (perf factor)
  "Linear scaling"
  (* perf factor))

;*****
; stretch expressive-timing

(defmethod stretch-expression ((section S-section)
                               (old S-map)
                               (new S-map)
                               (expression onset-timing))
  (make-new-section
   section
   (loop for perf-time in (expressions section)
        as (score-begin score-end) = (lookup-inverse old perf-time)
        collect (if (and score-begin score-end)
                    (interpolate (lookup-expression old score-begin)
                                 perf-time
                                 (lookup-expression old score-end)
                                 (lookup-expression new score-begin)
                                 (lookup-expression new score-end))
                    perf-time))))

```

```

;*****
; mixin to estimate expression in case of absence, by linear inter- or extrapolation
;*****
(defclass estimate-mixin () ())

(defmethod fetch-expression :around ((object musical-object) (expression estimate-mixin))
  (or (get-expression object expression)
      (estimate-expression object expression)))

(defmethod fetch-expression ((object null) (expression expression)) nil)

(defmethod fetch-expression ((object musical-object) (expression expression))
  (get-expression object expression))

(defmethod get-next-expression :around ((object musical-object)
                                       (expression estimate-mixin))
  (cond ((call-next-method))
        ((right object)
         (estimate-expression (right object) expression))
        (t
         (estimate-next-expression object expression))))

(defmethod fetch-onset :around ((object musical-object) (expression estimate-mixin))
  (fetch-expression object (find-expression 'estimate-onset-timing)))

(defmethod estimate-expression ((object musical-object) (expression expression))
  (estimate-context (context-with-expression object expression #'left)
                   object
                   (context-with-expression object expression #'right)
                   expression
                   t))

(defmethod estimate-next-expression ((object musical-object) (expression expression))
  (let* ((left (context-with-expression object expression #'left))
         (lefter (and left
                      (left left)
                      (context-with-expression (left left) expression #'left))))
    (when (and left lefter)
      (interpolate (score-onset lefter)
                  (score-offset object)
                  (score-onset left)
                  (get-expression lefter expression)
                  (get-expression left expression))))))

(defmethod estimate-context (left object right (expression expression) first-try)
  (cond ((and left right)
        (interpolate (score-onset left)
                    (score-onset object)
                    (score-onset right)
                    (get-expression left expression)
                    (get-expression right expression)))
        ((and left (left left) first-try)
         (estimate-context (context-with-expression (left left) expression #'left)
                          object
                          left
                          expression nil))
        ((and right (right right) first-try)
         (estimate-context right
                          object
                          (context-with-expression (right right) expression #'right)
                          expression nil))
        (t nil)))

(defmethod context-with-expression ((object musical-object)
                                   (expression expression) direction)
  (cond ((get-expression object expression)
        object)
        ((funcall direction object)
         (context-with-expression (funcall direction object) expression direction))
        (t nil)))

```

```

;*****
; keeping articulation invariant: mixin for expressive timing expression
;*****

(defclass keep-articulation-mixin () ())
(defclass keep-overlap-articulation-mixin (keep-articulation-mixin) ())
(defclass keep-duration-articulation-mixin (keep-articulation-mixin) ())
(defclass keep-proportion-articulation-mixin (keep-articulation-mixin) ())

(defmethod articulation ((expression keep-overlap-articulation-mixin))
  (find-expression 'basic-overlap-articulation))

(defmethod articulation ((expression keep-duration-articulation-mixin))
  (find-expression 'basic-duration-articulation))

(defmethod articulation ((expression keep-proportion-articulation-mixin))
  (find-expression 'basic-proportion-articulation))

(defmethod set-map :around ((object musical-object)
                             map
                             (expression keep-articulation-mixin)
                             ground)
  (when map
    (let* ((parts (find-parts object ground))
           (articulation-collections
            (loop for part in parts
                  collect (get-notes-expression part (articulation expression)))))
      (call-next-method)
      (loop for part in parts
            for collection in articulation-collections
            do (set-notes-expression part (articulation expression) collection)))
    object)

;*****
; resource for expression instances

(defvar *expression-instances*)
(setf *expression-instances* nil)
(defvar *use-expression-resource*)
(setf *use-expression-resource* t)

(defun find-expression (class)
  (or (and *use-expression-resource*
          (cdr (assoc class *expression-instances*)))
      (make-expression-instance class)))

(defun make-expression-instance (class)
  (let ((instance (make-instance class)))
    (when *use-expression-resource*
      (push (cons class instance) *expression-instances*)))
  instance)

;*****
; averaging expression
;*****

(defclass averaging-expression-mixin () ())

;*****
; get averaging expression

(defmethod get-expression ((object multilateral) (expression averaging-expression-mixin))
  (loop for component in (components object)
        when (get-expression component expression)
        sum it into total
        finally (return (/ total (length (components object)))))

(defmethod get-expression ((object collateral) (expression averaging-expression-mixin))
  (get-expression (main object) expression))

```

```

;*****
; set averaging expression

(defmethod set-expression ((object multilateral)
                           (expression averaging-expression-mixin)
                           (section multilateral-section))
  (loop for component in (components object)
        for new-expression in (expressions section)
        do (propagate-shift component
                             (save-- new-expression
                                       (fetch-expression component expression))
                             expression)))

(defmethod set-expression ((object collateral)
                           (expression averaging-expression-mixin)
                           (section collateral-section))
  (propagate-shift (ornament object)
                   (save-- (ornament-expression section)
                           (fetch-expression (ornament object) expression))
                   expression))

;*****
; scale averaging expression

(defmethod scale-expression ((section multilateral-section)
                             (expression averaging-expression-mixin)
                             factor)
  (let* ((mean-expression (mean (expressions section)))
         (expression-deviations (mapcar #'(lambda (expression)
                                           (- expression mean-expression))
                                         (expressions section)))
         (new-expressions
          (mapcar #'(lambda (expression-deviation)
                    (+ mean-expression
                      (scale-expression-lin expression-deviation factor)))
                  expression-deviations)))
    (make-new-section section new-expressions)))

(defmethod scale-expression ((section collateral-section)
                             (expression averaging-expression-mixin)
                             factor)
  (let* ((expression-deviation (- (ornament-expression section)
                                  (main-expression section)))
         (new-ornament-expression
          (+ (main-expression section)
            (scale-expression-lin expression-deviation factor))))
    (make-new-section section
                      (list new-ornament-expression
                            (main-expression section)))))

;*****
; stretch averaging expression

(defmethod stretch-expression ((section S-section)
                               (old S-map)
                               (new S-map)
                               (expression averaging-expression-mixin))
  (make-new-section
   section
   (loop for expression in (expressions section)
         for score-time in (score-times section)
         as old-expression = (lookup-expression old score-time)
         as new-expression = (lookup-expression new score-time)
         as stretched-expression = (if (and old-expression new-expression expression)
                                       (+ expression (- new-expression old-expression))
                                       expression)
         collect stretched-expression)))

```



```

;*****
; ARTICULATION
;*****

(defclass offset-timing          (expressive-timing) ())
(defclass articulation          (offset-timing averaging-expression-mixin) ())
(defclass basic-overlap-articulation (articulation) ())
(defclass basic-duration-articulation (articulation) ())
(defclass basic-proportion-articulation (articulation) ())

;*****

(defmethod get-expression ((object NOTE) (expression offset-timing))
  (perf-offset object))

(defmethod fetch-onset ((object musical-object) (expression articulation))
  (get-expression object (find-expression 'onset-timing)))

;*****
; get articulation

(defmethod get-expression ((object NOTE) (expression basic-overlap-articulation))
  (when (right object)
    (save-- (perf-offset object)
            (fetch-onset (right object) expression))))

(defmethod get-expression ((object NOTE) (expression basic-duration-articulation))
  (- (perf-offset object)
     (fetch-onset object expression)))

(defmethod get-expression ((object NOTE) (expression basic-proportion-articulation))
  (when (and (fetch-onset object expression)
             (right object)
             (fetch-onset (right object) expression))
    (/ (- (perf-offset object)
          (fetch-onset object expression))
        (- (fetch-onset (right object) expression)
           (fetch-onset object expression)))))

;*****
; set articulation

(defmethod set-expression ((object NOTE) (expression basic-overlap-articulation) value)
  (when (and (right object) (fetch-onset (right object) expression))
    (setf (perf-offset object)
          (max (fetch-onset object expression)
               (+ (fetch-onset (right object) expression)
                  value)))))

(defmethod set-expression ((object NOTE) (expression basic-duration-articulation) value)
  (setf (perf-offset object)
        (+ (fetch-onset object expression)
           (max 0 value))))

(defmethod set-expression ((object NOTE)
                          (expression basic-proportion-articulation) value)
  (when (and (right object) (perf-onset (right object)))
    (setf (perf-offset object)
          (+ (fetch-onset object expression)
             (* (- (fetch-onset (right object) expression)
                  (fetch-onset object expression))
                (max 0 value))))))

```

```

;*****
; empty expression (to recover only score times)
;*****

(defclass empty-expression (expression) ())

(defmethod get-expression ((object musical-object) (expression empty-expression)) nil)

;*****
; mixing instantiable classes of expression
;*****

(defmacro class-mixer (&rest class-cocktail-pairs)
  (list* 'progl t
    (loop for tuples on class-cocktail-pairs by #'caddr
      as name = (first tuples)
      as doc = (second tuples)
      as cocktail = (third tuples)
      collect `(defclass ,name ,cocktail ()
                (:documentation ,doc))))))

```

```

(class-mixer
  tempo " "
    (basic-tempo)

  asynchrony " "
    (basic-asynchrony)

  estimate-tempo " "
    (basic-tempo estimate-mixin)

  estimate-asynchrony " "
    (basic-asynchrony estimate-mixin)

  keep-overlap-articulation-tempo " "
    (basic-tempo keep-overlap-articulation-mixin)

  keep-duration-articulation-tempo " "
    (basic-tempo keep-duration-articulation-mixin)

  keep-proportion-articulation-tempo " "
    (basic-tempo keep-proportion-articulation-mixin)

  keep-overlap-articulation-estimate-tempo " "
    (basic-tempo keep-overlap-articulation-mixin estimate-mixin)

  keep-duration-articulation-estimate-tempo " "
    (basic-tempo keep-duration-articulation-mixin estimate-mixin)

  keep-proportion-articulation-estimate-tempo " "
    (basic-tempo keep-proportion-articulation-mixin estimate-mixin)

  keep-overlap-articulation-asynchrony " "
    (basic-asynchrony keep-overlap-articulation-mixin)

  keep-duration-articulation-asynchrony " "
    (basic-asynchrony keep-duration-articulation-mixin)

  keep-proportion-articulation-asynchrony " "
    (basic-asynchrony keep-proportion-articulation-mixin)

  keep-overlap-articulation-estimate-asynchrony " "
    (basic-asynchrony keep-overlap-articulation-mixin estimate-mixin)

  keep-duration-articulation-estimate-asynchrony " "
    (basic-asynchrony keep-duration-articulation-mixin estimate-mixin)

  keep-proportion-articulation-estimate-asynchrony " "
    (basic-asynchrony keep-proportion-articulation-mixin estimate-mixin)

  overlap-articulation " "
    (basic-overlap-articulation)

  duration-articulation " "
    (basic-duration-articulation)

  proportion-articulation " "
    (basic-proportion-articulation)

  estimate-overlap-articulation " "
    (basic-overlap-articulation estimate-mixin)

  estimate-duration-articulation " "
    (basic-duration-articulation estimate-mixin)

  estimate-proportion-articulation " "
    (basic-proportion-articulation estimate-mixin))

```

```

;*****
;*****
; EXTRACTING AND IMPOSING EXPRESSION MAPS OF MUSICAL OBJECTS USING EXPRESSION
;*****
;*****
; extracting a expression map

(defmethod get-map ((object musical-object) expression ground)
  (make-map (loop for part in (find-parts object ground)
                 collect (get-section part expression))))

(defmethod get-section ((object musical-object) expression)
  (make-section (object-to-section object)
                (snoc (mapcar #'score-onset (components object))
                      (score-offset object))
                (snoc (mapcar #'(lambda (component)
                                (fetch-expression component expression))
                          (components object))
                      (get-next-expression object expression))))

;*****
; impose a expression map

(defmethod set-map ((object musical-object) map expression ground)
  (loop for part in (find-parts object ground)
        for section in (sections map)
        do (set-expression part expression section)
        object)

```



```

;*****
;*****
; OPERATIONS ON EXPRESSION MAPS
;*****
;*****
; scale expression map

(defmethod scale-map ((map map) expression factor)
  (with-filtered-null-expression #'(lambda (filtered-map)
    (scale-filtered-map filtered-map expression factor)
    map))

(defmethod scale-filtered-map ((map map) expression factor)
  (map-map #'(lambda (section)
    (scale-expression section
      expression
      (get-parameter factor (score-onset section))))
    map))

;*****
; interpolate S-expression maps

(defmethod interpolate-maps ((map1 S-map) (map2 S-map) factor)
  (map-map #'(lambda (section) (interpolate-section section
    (filter-null-expression map2)
    factor))
    map1))

(defmethod interpolate-section ((section S-section) (map S-map) factor)
  (make-new-section
    section
    (loop for score-time in (score-times section)
      for expression in (expressions section)
      collect (in-between expression
        (lookup-expression map score-time)
        (get-parameter factor score-time))))

(defmethod monotonise-map ((map S-map))
  (map-map #'monotonise-section map))

(defmethod monotonise-section ((section S-section))
  (make-new-section
    section
    (loop for expression in (expressions section)
      when expression
      maximize expression into state
      and collect state
      else collect nil)))

;*****
; get S-expression maps at sync points

(defmethod get-sync-map ((map1 S-map) (map2 S-map))
  (map-map #'(lambda (section) (get-sync-section section map2)) map1))

(defmethod get-sync-section ((section S-section) (map S-map))
  (make-new-section-from-pairs section
    (loop for score-time in (all-score-times section)
      for expression in (all-expressions section)
      as new-expression = (and expression
        (lookup-defined-expression map score-time))
      when new-expression collect (list score-time expression))))

```

```

;*****
; stretch expression map

(defmethod stretch-map ((map successive-map)
                        (old successive-map)
                        (new successive-map)
                        expression)
  (let ((filtered-map (filter-null-expression map))
        (filtered-old (filter-null-expression old))
        (filtered-new (filter-null-expression new))
        (removed (filter-null-expression-out map)))
    (unfilter-null-expression
     (map-map
      #'(lambda (section)
          (stretch-expression section filtered-old filtered-new expression))
      filtered-map)
     removed)))

;*****
; TIME-CHANGING PARAMETERS
;*****

(defun get-parameter (factor score-time)
  (if (numberp factor)
      factor
      (funcall factor score-time)))

(defun make-ramp (x1 x2 y1 y2) ; as s-section ??
  #'(lambda (x) (interpolate x1 x x2 y1 y2)))

```

```

;*****
;*****
; TRANSFORMATIONS ON MUSICAL OBJECTS
;*****
;*****
; transfer expression transformation

(defmethod transfer ((object musical-object) expression foreground background)
  (let* ((foreground-map (get-map object expression foreground))
         (background-map (get-map object (find-expression 'empty-expression) background))
         (new-background-map (interpolate-maps background-map foreground-map 1)))
    (set-map object new-background-map expression background)
    object)

;*****
; scale expression transformation

(defmethod scale ((object musical-object) expression foreground background factor)
  (let* ((old-foreground-map (get-map object expression foreground))
         (new-foreground-map (when old-foreground-map
                               (scale-map old-foreground-map expression factor)))
         (old-background-map (when background
                               (get-map object expression background)))
         (new-background-map (when old-background-map
                               (stretch-map old-background-map
                                           old-foreground-map
                                           new-foreground-map
                                           expression))))
    (when new-foreground-map
      (set-map object new-foreground-map expression foreground))
    (when new-background-map
      (set-map object new-background-map expression background)))
  object)

;*****
; scale intervoice expression transformation

(defmethod scale-intervoice ((object musical-object) expression
                             voice1 voice2 factor ref)
  (let* ((map1 (get-map object expression voice1))
         (map2 (get-map object expression voice2)))
    (when (and map1 map2)
      (let* ((original-sync-map1 (get-sync-map map1 map2))
             (original-sync-map2 (get-sync-map map2 map1))
             (new-sync-map1 (monotonise-map (interpolate-maps
                                             original-sync-map1
                                             original-sync-map2
                                             (* ref (- 1 factor))))))
        (new-sync-map2 (monotonise-map (interpolate-maps
                                         original-sync-map2
                                         original-sync-map1
                                         (* (- 1 ref) (- 1 factor))))))
        (new-map1 (stretch-map
                   map1 original-sync-map1 new-sync-map1 expression))
        (new-map2 (stretch-map
                   map2 original-sync-map2 new-sync-map2 expression)))
      (set-map object new-map1 expression voice1)
      (set-map object new-map2 expression voice2)))
    object))

```

```

;*****
;*****
; LISP UTILITIES
;*****
;*****

(defun last-element (list)
  (first (last list)))

(defun snoc (list item)
  (append list (list item)))

(defun mean (numbers)
  (/ (apply #' + numbers) (length numbers)))

(defun save-min (&rest list)
  (let ((new-list (remove nil list)))
    (and new-list (apply #'min new-list))))

(defun save-max (&rest list)
  (let ((new-list (remove nil list)))
    (and new-list (apply #'max new-list))))

(defun save-- (&rest list)
  (and (notany #'null list)
    (apply #'- list)))

(defun save++ (&rest list)
  (apply #' + (remove nil list)))

(defun enforce-limits (minimum x maximum)
  (max minimum (min x maximum)))

(defun integrate (list start)
  (if (null list)
    (list start)
    (cons start
      (integrate (rest list) (+ (first list) start)))))

(defun normalise (list dur)
  (let ((factor (/ dur (apply #' + list))))
    (mapcar #'(lambda(item) (* factor item)) list)))

(defun interpolate (x1 x x2 y1 y2)
  (cond ((eql y1 y2) y1)
        ((eql x1 x2) nil)
        ((null x) nil)
        ((and x1 (= x x1)) y1)
        ((and x2 (= x x2)) y2)
        ((and x1 x2)
         (in-between y1 y2 (/ (- x x1) (- x2 x1))))
        (t nil)))

(defun in-between (y1 y2 a)
  (cond ((= a 0) y1)
        ((= a 1) y2)
        ((and y1 y2)
         (+ y1 (* a (- y2 y1))))
        (t nil)))

```



```

;*****
;*****
; EXAMPLES
;*****
;*****
#|

(defun metre-example ()
  (S 'bars
    (P 'bar
      (S 'melody
        (PAUSE :name 'pause :score-dur 1/4)
        (NOTE :name 64 :score-dur 1/8
          :perf-onset .30 :perf-offset 0.5 :dynamic .7))
      (S 'accompaniment
        (PAUSE :name 'pause :score-dur 3/8)))
    (P 'bar
      (S 'melody
        (APPOG 'appoggiatura
          (NOTE :name 64 :score-dur 1/8
            :perf-onset .550 :perf-offset .680 :dynamic .75)
          (NOTE :name 55 :score-dur 1/4
            :perf-onset .675 :perf-offset 1.133 :dynamic .7))
        (NOTE :name 55 :score-dur 1/8
          :perf-onset 1.125 :perf-offset 1.475 :dynamic .7))
      (S 'accompaniment
        (NOTE :name 38 :score-dur 1/8
          :perf-onset .725 :perf-offset .90 :dynamic .6)
        (NOTE :name 43 :score-dur 1/8
          :perf-onset .95 :perf-offset 1.2 :dynamic .6)
        (NOTE :name 47 :score-dur 1/8
          :perf-onset 1.150 :perf-offset 1.475 :dynamic .7)))
    (P 'bar
      (S 'melody
        (ACCIA 'acciaccatura
          (NOTE :name 59 :score-dur 1/16
            :perf-onset 1.600 :perf-offset 1.7 :dynamic .65)
          (NOTE :name 57 :score-dur 1/8
            :perf-onset 1.625 :perf-offset 1.880 :dynamic .7))
        (NOTE :name 55 :score-dur 1/8
          :perf-onset 1.880 :perf-offset 2.256 :dynamic .6)
        (NOTE :name 57 :score-dur 1/8
          :perf-onset 2.256 :perf-offset 2.647 :dynamic .65))
      (S 'accompaniment
        (P 'chord
          (NOTE :name 38 :score-dur 3/8
            :perf-onset 1.725 :perf-offset 2.500 :dynamic .7)
          (NOTE :name 42 :score-dur 3/8
            :perf-onset 1.775 :perf-offset 2.500 :dynamic .65)
          (NOTE :name 48 :score-dur 3/8
            :perf-onset 1.800 :perf-offset 2.500 :dynamic .7))))
    (P 'bar
      (S 'melody
        (NOTE :name 55 :score-dur 3/8
          :perf-onset 2.425 :perf-offset 4 :dynamic .7))
      (S 'accompaniment
        (P 'chord
          (NOTE :name 43 :score-dur 3/8
            :perf-onset 2.500 :perf-offset 4 :dynamic .6)
          (NOTE :name 47 :score-dur 3/8
            :perf-onset 2.550 :perf-offset 4 :dynamic .7)
          (NOTE :name 50 :score-dur 3/8
            :perf-onset 2.580 :perf-offset 4.5 :dynamic .65))))))

```

```

(defun background-example ()
  (P 'fragment
    (S 'melody
      (PAUSE :name 'pause :score-dur 1/4)
      (NOTE :name 64 :score-dur 1/8
        :perf-onset 0.3 :perf-offset 0.5 :dynamic .7)
      (APPOG 'appoggiatura
        (NOTE :name 64 :score-dur 1/8
          :perf-onset .550 :perf-offset .680 :dynamic .75)
        (NOTE :name 55 :score-dur 1/4
          :perf-onset .675 :perf-offset 1.133 :dynamic .7))
      (NOTE :name 55 :score-dur 1/8
        :perf-onset 1.125 :perf-offset 1.475 :dynamic .7)
      (ACCIA 'acciaccatura
        (NOTE :name 59 :score-dur 1/16
          :perf-onset 1.600 :perf-offset 1.700 :dynamic .65)
        (NOTE :name 57 :score-dur 1/8
          :perf-onset 1.625 :perf-offset 1.880 :dynamic .7))
      (NOTE :name 55 :score-dur 1/8
        :perf-onset 1.880 :perf-offset 2.256 :dynamic .6)
      (NOTE :name 57 :score-dur 1/8
        :perf-onset 2.256 :perf-offset 2.647 :dynamic .65)
      (NOTE :name 55 :score-dur 3/8
        :perf-onset 2.425 :perf-offset 4 :dynamic .7))
    (S 'accompaniment
      (PAUSE :name 'pause :score-dur 3/8)
      (NOTE :name 38 :score-dur 1/8
        :perf-onset .725 :perf-offset .90 :dynamic .6)
      (NOTE :name 43 :score-dur 1/8
        :perf-onset .950 :perf-offset 1.2 :dynamic .6)
      (NOTE :name 47 :score-dur 1/8
        :perf-onset 1.150 :perf-offset 1.475 :dynamic .7)
      (P 'chord
        (NOTE :name 38 :score-dur 3/8
          :perf-onset 1.725 :perf-offset 2.500 :dynamic .7)
        (NOTE :name 42 :score-dur 3/8
          :perf-onset 1.775 :perf-offset 2.500 :dynamic .65)
        (NOTE :name 48 :score-dur 3/8
          :perf-onset 1.800 :perf-offset 2.500 :dynamic .7))
      (P 'chord
        (NOTE :name 43 :score-dur 3/8
          :perf-onset 2.500 :perf-offset 4 :dynamic .6)
        (NOTE :name 47 :score-dur 3/8
          :perf-onset 2.550 :perf-offset 4 :dynamic .7)
        (NOTE :name 50 :score-dur 3/8
          :perf-onset 2.580 :perf-offset 4.5 :dynamic .65))))))

;data at factor 2 in figure 13
(scale (metre-example)
  (find-expression 'tempo) (has-name? 'bars) nil
  2)

;data at factor 2 in figure 14
(scale (metre-example)
  (find-expression 'asynchrony) (has-name? 'bar) nil
  2)

;data at factor 2 in figure 16a
(scale (background-example)
  (find-expression 'tempo) (has-name? 'melody) nil
  2)

;data at factor 2 in figure 16b
(scale (background-example)
  (find-expression 'tempo) (has-name? 'melody) (has-name? 'accompaniment)
  2)

|#

```