Timing in music and modal temporal logic

ALAN MARSDEN

Lancaster Institute for the Contemporary Arts, Lancaster University

'In-time' representations of music in which the time represented is the same time as inhabited by the agent making or using the representation are contrasted with 'out-of-time' representations. Temporal logics with a similar 'in-time' perspective, and in particular those using operators S and U for 'since' and 'until', are explored as a means of representing musical situations, with particular reference to a paradigm 'triangle-player problem'. Illustrative implementations are given in the music software Pd. New versions of the operators S and U are defined to accommodate the musically important phenomena of regularly occurring events associated with metre, and to allow representations to reflect actual timings rather than relations of temporal order. Nesting of out-of-time representations within in-time representations then becomes possible and arises naturally as a way of representing certain kinds of musical situation.

Keywords: Representation; Musical time; Co-ordination; Temporal logic
AMS Subject Classification Codes: 03B44; 68N30; 91F99

1 Introduction

The representation of time in music is clearly important, but turns out often not to be simple. Different situations require different kinds of representations, discussed by a number of researchers in previous publications (for a summary, see [1]). Here consideration is given only to the situation where the representing agent inhabits the same flow of time as the music represented. This kind of representation is crucial in the modelling of listening processes, and in the design of automata to perform or create music in real-time set-ups. Relevant previous work exists in the theoretical formulations of Kunst [2–3] and Leman [4–6], with respect to the modelling of the listening experience, and in real-time software systems with a strong theoretical base such as HARP [7–8] and various systems based on Petri nets [9–10]. The objective of this paper is to enhance the mathematical rigour underlying such work by formulating concepts based on temporal logic which might provide for musical systems the kind of basis for proof and formal design which it provides for computer science (see [11]).

When one listens to a live performance, one must make some kind of mental representation of the music performed; otherwise one could not recognise a recurrence of a passage heard before. But, unlike someone transcribing a recorded performance, it is not

Correspondence: Alan Marsden. Lancaster Institute for the Contemporary Arts, Lancaster University, LA1 4YW, UK;
Tel. +44 1524 593774; Fax. +44 1524 593939; E-mail: A.Marsden@lancaster.ac.uk

possible for the listener to actually review some earlier stage in the performance to check that it is indeed the same passage of music as just heard; the only information the listener has about earlier occurrences is in the representation he or she has already made and is currently adding to. Similarly, one's representation of the future can only be in anticipations of events and plans for actions. A musician playing from a score operates simultaneously with two kinds of temporal representations. The score represents temporal relations in an abstract time which can be related differently to actual times in different actual performances. The performer's memory and anticipation, on the other hand, on which her or his performance relies, refers implicitly to the time in which the performer lives. These two kinds of temporal representation are referred to as 'in-time' representations for those cases where reference is made only through memory and anticipation, and 'out-of-time' representations for cases like a musical score.

These considerations apply not only to (human) musicians, but also to musical automata. An automaton behaving in a musical fashion can make use of one or more out-of-time representations, similarly to a musician making use of a score, but it must also have an in-time representation of its own past and, possibly, its own future. This representation of past and future need not be explicit in the automaton, and indeed it is common for it to be implicit in its structure and internal state. However, explicit representation of temporal relations is necessary for reasoning about them, such as in the design and proofs of correctness of computer software, and the task of this paper is to begin to identify which kinds of in-time temporal representations are suitable for music.

1.1 Paradigmatic triangle-player problem

To focus the discussion, a paradigmatic problem is proposed. Consider a triangle player who has one note **x** to play in a piece of music. This note must occur at the same time as another note **y** played by another performer in the ensemble. The triangle player has information about some of the notes **a**, **b**, **c** … the other player is to play before the note **y**. (See Figure 1.) The essential requirement of the problem, that when **y** is sounded **x** should also be sounded, **Y** → **X,** suggests a solution of a sort—wait until **y** and immediately play **x**—but this is clearly not satisfactory in either of the scenarios suggested above. It takes time to hit a triangle, and to recognise that a note has been played. A proper solution requires the triangle player to recognise that the other player has reached a point in his or her part sufficiently before the note **y** in time to initiate the playing of note **x** so that it sounds at the same time as **y**. On the other hand, the triangle player must be able to respond to changes in timing by the other player which will affect the timing of note **y**, and so must leave initiation of the process of playing note **x** as late as possible.

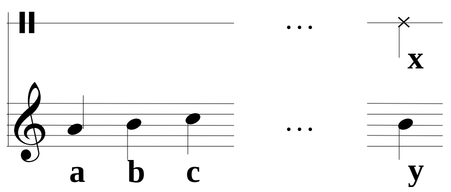

Figure 1. The 'triangle-player problem'; note x must be played at the same time as note y, using the timing information conveyed by the preceding notes a, b, c, ….

1.2 Modal temporal logic

Temporal logics which are here called 'modal' introduce into classical logic new connectives with an explicitly temporal meaning. Sometimes they re-use the modal connectives □

(necessity) and ◊ (possibility), to which can be added ○ for ‘next’ (or ‘previous’), but since these can be variously used to refer to the past or the future, and since other connectives with no modal equivalent can be used (such as those for ‘since’ and ‘until’), it is common to use upper case letters for the temporal connectives. (Descriptions of temporal logic can be found in [12–15].) Common connectives, including those used in this paper, are given in Table 1, together with their definitions in English and in first-order predicate logic. These definitions use a special term **now**, which functions like a constant but is properly an indexical, to refer to the time at which the truth-value of the statement is evaluated. The term $x < y$ is true when x precedes y, and *vice versa* for $x > y$. Precise definitions of the temporal connectives vary from one temporal logic to another, and logics vary in the ontology of time assumed (see [1] pp.23–53). For now, it is sufficient to note that the connectives Y and T are only valid in the case of time with discrete steps and $t + 1$ is the time immediately following time t, and that a logic with S and U (in terms of which the other connectives can be defined) is expressively complete and decidable for linear time (except in certain ontologies, see [11]).

Past		**Modal**		**Future**
$\mathrm{P}A =_{\mathrm{def}} \exists t((t < \mathbf{now}) \wedge A(t))$ *A* was true at some time in the Past.	P	◊	F	$\mathrm{F}A =_{\mathrm{def}} \exists t((t > \mathbf{now}) \wedge A(t))$ *A* will be true at some time in the Future.
$\mathrm{H}A =_{\mathrm{def}} \forall t((t < \mathbf{now}) \rightarrow A(t))$ *A* always Has been true.	H	□	G	$\mathrm{G}A =_{\mathrm{def}} \forall t((t > \mathbf{now}) \rightarrow A(t))$ *A* is always Going to be true
$\mathrm{Y}A =_{\mathrm{def}} \exists t((t = \mathbf{now} - 1) \wedge A(t))$ *A* was true Yesterday.	Y	○	T	$\mathrm{T}A =_{\mathrm{def}} \exists t((t = \mathbf{now} + 1) \wedge A(t))$ *A* will be true Tomorrow.
$\mathrm{S}(A, B) =_{\mathrm{def}} \exists t((t < \mathbf{now}) \wedge A(t) \wedge \forall s((t < s < \mathbf{now}) \rightarrow B(s)))$ *B* has been true Since *A*.	S		U	$\mathrm{U}(A, B) =_{\mathrm{def}} \exists t((t > \mathbf{now}) \wedge A(t) \wedge \forall s((t > s > \mathbf{now}) \rightarrow B(s)))$ *B* will be true Until *A*.

Table 1. Common temporal-logic connectives.

Temporal logic can function not only as quasi-mathematical problem-solving device, but also as a language for the description and design of automata and computer systems. Indeed, there have been substantial research efforts to develop temporal logics for use in the specification and design of real-time systems, especially where proof is required that the system will work safely. One of the best known is the Duration Calculus [16], a logic which is not always decidable, but which has been demonstrated to be decidable in likely realistic implementation scenarios [17]. While we can expect, therefore, that Duration Calculus could be used to specify a solution to the triangle-player problem, it is more general in its scope than is necessary for music-specific problems. An alternative approach is to use a concurrent process calculus, such as in [18–19]. This works well for situations where there is a ‘bird’s-eye view’ of interacting agents, but the intention here is to model the processing of a single agent whose only knowledge of the past is through memories of sensed events, and whose only control over the future is through its own behaviour. Such an agent is modelled in a network system capable of realising a ‘score’ in interaction with live events in [20].

My intention here, however, is not so much to design a comprehensive implementation logic, but rather to make a conceptual exploration of formalising the requirements of automata capable of musical timing. For this the simplicity of classical modal temporal logic is sufficient, and so it will be the basis of the formulations used here. A common approach when using temporal logic to define the behaviour of an automaton has been described as ‘declarative past, imperative future’ [21]: statements referring to the past recognise that a particular situation has occurred, and these imply statements referring to the future which are taken as a prescription of actions to perform in order to make certain propositions become true. For example, the basic functionality of a burglar alarm can be represented by the following two statements.

$\mathrm{S}(\mathbf{A} \wedge \mathbf{B}, \mathbf{A}) \rightarrow \mathrm{U}({\sim}\mathbf{A}, \mathbf{X})$ (1)

${\sim}\mathbf{A} \rightarrow {\sim}\mathbf{X}$ (2)

A is true when the alarm is set, **B** when there is an intrusion, and **X** when the alarm sounds. Formula (1) states that when there has been an intrusion at a time when the alarm is set, then throughout the time when the alarm remains set, the bell will sound. Formula (2) ensures that the bell does not sound when the alarm is turned off. Note the importance of using S formula (1): if on the left-hand side it had simply P(**A** ∧ **B**), then, after turning off the alarm, it would start to sound as soon as it was set again. Using S ensures that the alarm will sound only if it has been set continuously since the last intrusion. A mechanism which was certain to behave according to these formulae would be suitable as a burglar-alarm controller.

2 Representing a sequence in temporal logic

To sketch a similar logical description of a solution to the triangle-player problem, we begin by exploring the representation of events in the past in a temporal logic using the connectives described above. In these representations, logical propositions such as **A** will be considered true at all times that the note **a** is sounding. Conversely, ~**A** will be considered true at all times that the note **a** is not sounding. We will take the situation leading up to the triangle note to be as illustrated in Figure 2.

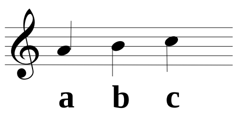

Figure 2. Simple sequence of notes.

We can represent situation in which the notes **a**, **b**, and **c** have occurred in that order by the statement

$\mathrm{P}(\mathrm{P}(\mathrm{P}\mathbf{A} \wedge \mathbf{B}) \wedge \mathbf{C})$ (3)

but this only requires that at some time in the past note **c** was sounding and in the past relative to that time note **b** was sounding, and in the past relative to that time note **a** was sounding. This is not a sufficient characterisation of the situation in Figure 2 for three reasons:

(a) Any time interval between **a**, **b**, and **c** is acceptable, but the notation in Figure 2 requires the intervals to be at least approximately equal, and in most contexts to be within a certain range.

(b) The formula does not require **b** to immediately follow **a** nor **c** to immediately follow **b**. Silences, other notes or even other occurrences of **a**, **b**, or **c** could occur in between and the formula would still be true.

(c) The formula does not preclude the possibility of other notes occurring in addition to **a**, **b**, and **c**.

Problem (c) raises the biggest questions. The universe of the formula should correspond to the focus of the triangle player, which will generally be on just a part of the score where a single line or voice of music is represented. Note occurring in other voices should not matter, but other notes in that voice should falsify the formula. Unfortunately, how musical voices are defined in relation to a whole piece remains the real mystery of music and temporal structures (see [22] for some discussion), and for the remainder of this paper it will be assumed that the notes of the voice in question can be unequivocally distinguished from other notes.

2.1 Representation in discrete time

To solve problem (a) requires some form of measurement or at least comparison of time intervals. In discrete domains this is easy to achieve by using the connective Y to effectively count the number of time quanta between an event in the past and the present. If the notes of Figure 2 are to be four quanta long, for example, and the triangle player needs two time quanta to prepare to strike the triangle, a solution to the triangle-player problem could be represented by formula (4).

$\mathrm{YY(YYYY(YYYY}\mathbf{A} \wedge \mathbf{B}) \wedge \mathbf{C}) \rightarrow \mathrm{TT}(\mathbf{X} \wedge \mathbf{Y}) \wedge \mathbf{INITIATE\_STRIKE}$ (4)

However, this leaves problem (b) unsolved, allowing gaps and intervening notes. Formula (5) remedies this by requiring notes **a**, **b**, and **c** to sound in each time quantum.

$\mathrm{Y(Y(Y(Y(Y(Y(Y(Y(Y(Y}\mathbf{A} \wedge \mathbf{A}) \wedge \mathbf{A}) \wedge \mathbf{A}) \wedge \mathbf{B}) \wedge \mathbf{B}) \wedge \mathbf{B}) \wedge \mathbf{B}) \wedge \mathbf{C}) \wedge \mathbf{C}) \rightarrow$
$\mathrm{TT}(\mathbf{X} \wedge \mathbf{Y}) \wedge \mathbf{INITIATE\_STRIKE}$ (5)

However, this is still not an acceptable solution to the problem because it requires timing more precise than is generally produced in actual performance situations. This is illustrated by example software[*], shown in Figure 3, written in Pd [23], a free graphical data-flow language for MIDI and audio similar to Max/MSP. To emphasise the anticipation time required, the 'triangle' object here actually plays a roll with a crescendo culminating in a final strike at the required time. The length of this roll is here set to 250 milliseconds. The 'noteInput' object causes the software to listen for input on MIDI channel 1, also requested for the three 'note' objects which detect the notes A, B, and C (MIDI numbers 69, 71 and 72). These emit 1 when a note starts, and 0 when it is replaced by another note on the same channel or if it has been silent for at least a second. (This is to accommodate the realities of keyboard playing: players often play notes shorter than their notated duration, in a detached or staccato style of playing, or, when playing legato, hold a note for longer than its notated duration so that it actually overlaps the following note.) The 'Y' object emits 1 or 0 at each tick (sent on a background channel by the 'ticker' object), according to whether or not it last received 1 or 0 on its inlet. Pd objects normally emit output on receiving input on the left inlet, but the 'and' object (and a corresponding 'or' object) have been written to emit output whenever input is received on either inlet. (These objects are shown in Figure 5.) The output is 1 whenever the input is 1 and the previous input on the other inlet was also 1. The object 'select 1' effectively listens for a 1 on its inlet, and when it is received (i.e., the preconditions for the triangle note are true), it sends a 'bang' to cause the triangle to sound. The 'metronome' object causes a click to sound every four ticks, allowing the user to hear the time interval expected between notes for the triangle to sound. It is quite difficult to play the notes A, B and C at just the right tempo and evenly enough to make the triangle sound, even with a time quantum as coarse as 125 milliseconds (one eighth of a second).

This can be remedied by altering the specification of the preconditions to allow some flexibility in timing, as in formula (6) and Figure 4. With this version of the software, it is possible to play away in approximately the right tempo, and whenever the notes A, B and C are played as approximately crotchets, the triangle will sound.

$\mathrm{Y(Y((}$

$\mathrm{Y(Y(Y((Y(Y(Y}\mathbf{A} \wedge \mathbf{A}) \wedge \mathbf{A}) \vee \mathrm{Y(Y(Y(Y}\mathbf{A} \wedge \mathbf{A}) \wedge \mathbf{A}) \wedge \mathbf{A}) \vee \mathrm{Y(Y(Y(Y(Y}\mathbf{A} \wedge \mathbf{A}) \wedge \mathbf{A})$
$\wedge \mathbf{A}) \wedge \mathbf{A})) \wedge \mathbf{B}) \wedge \mathbf{B}) \wedge \mathbf{B}) \vee$

$\mathrm{Y(Y(Y(Y((Y(Y(Y}\mathbf{A} \wedge \mathbf{A}) \wedge \mathbf{A}) \vee \mathrm{Y(Y(Y(Y}\mathbf{A} \wedge \mathbf{A}) \wedge \mathbf{A}) \wedge \mathbf{A}) \vee \mathrm{Y(Y(Y(Y(Y}\mathbf{A} \wedge$
$\mathbf{A}) \wedge \mathbf{A}) \wedge \mathbf{A}) \wedge \mathbf{A})) \wedge \mathbf{B}) \wedge \mathbf{B}) \wedge \mathbf{B}) \wedge \mathbf{B}) \vee$

$\mathrm{Y(Y(Y(Y(Y((Y(Y(Y}\mathbf{A} \wedge \mathbf{A}) \wedge \mathbf{A}) \vee \mathrm{Y(Y(Y(Y}\mathbf{A} \wedge \mathbf{A}) \wedge \mathbf{A}) \wedge \mathbf{A}) \vee \mathrm{Y(Y(Y(Y(Y}\mathbf{A} \wedge$
$\mathbf{A}) \wedge \mathbf{A}) \wedge \mathbf{A}) \wedge \mathbf{A})) \wedge \mathbf{B}) \wedge \mathbf{B}) \wedge \mathbf{B}) \wedge \mathbf{B}) \wedge \mathbf{B})$

[*] The software examples may be downloaded from http://www.lancs.ac.uk/staff/marsdena/software/timing2007

$$) \wedge \mathbf{C}) \wedge \mathbf{C}) \rightarrow \mathrm{TT}(\mathbf{X} \wedge \mathbf{Y}) \wedge \mathbf{INITIATE\_STRIKE} \qquad (6)$$

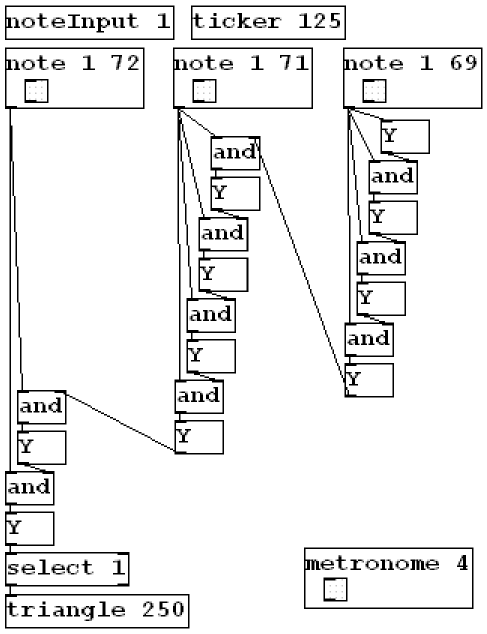

Figure 3. Pd implementation of the strict solution using Y.

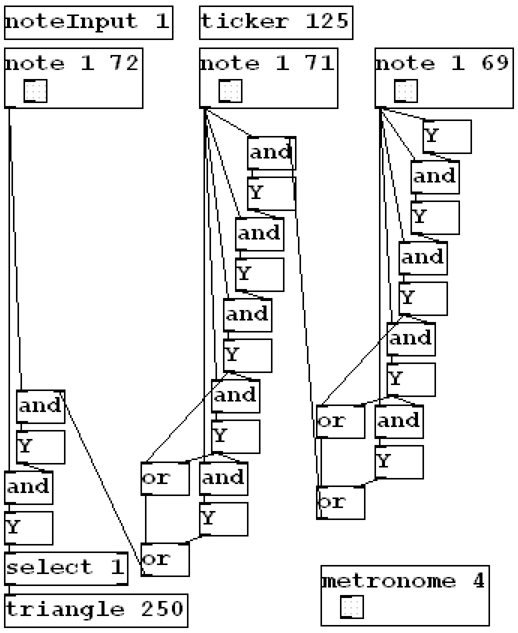

Figure 4. Pd implementation of a more realistic solution using Y.

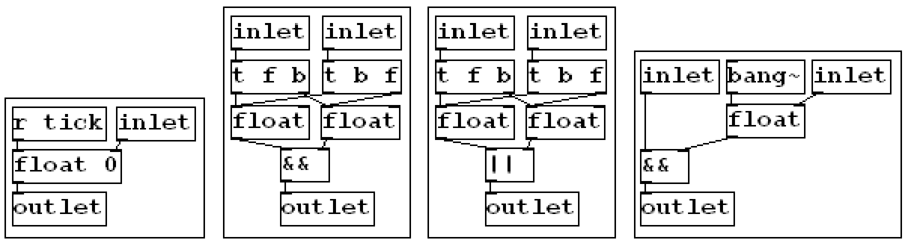

Figure 5. Pd implementations of Y, ‘and’, ‘or’ and S.

There remain three reasons, however, why this is not a good model of a general solution to the triangle-player problem.

(a) A specific solution must be related to a specific granularity of discrete time.

(b) Since timings are effectively specifically built into a solution, a complex set of disjunctions is required to mimic a realistic situation when the timings of the triangle player adapt to the timings used by the player of the other part, effectively building into the solution every possible specific timing that the triangle player has to respond to.

(c) A solution of this sort only requires the time intervals between the notes **a**, **b**, and **c** to be equal and to be within a certain range. In general, music notation also specifies that the notes must occur close to one of a certain set of times, corresponding to the beats of a piece of music. The notes **a**, **b**, and **c**, for example, might all occur between beats, which would not match the notation of Figure 2 as normally interpreted, but the triangle will still sound according to formula (6) and in the software of Figure 4.

In any case, it is preferable to express a solution in a logic of dense time and to leave the translation to the discrete time within which computer systems generally operate as an implementation issue.

2.2 Representation in dense time

In dense time, the operator Y is not available, but a continuous sequence of notes can be represented using the operator S, as in formula (7).

$$S(S(\mathbf{A}, \mathbf{B}), \mathbf{C}) \qquad (7)$$

The implementation of this operator in Pd is simple (shown in Figure 5), because of the way in which Pd reads inlets in a fixed order. The S object emits 1 whenever 1 is received on the left inlet and the value last received on the right inlet was also 1. Otherwise it emits 0 whenever any input is received on the left inlet.

More complex configurations of notes can also be represented using the S operator. (Indeed, the expressive completeness of the S and U operators implies that *any* configuration of notes could be represented.) A configuration in which **b** has followed **a** after an overlap (Figure 6A) could be represented as

$$S(S(\mathbf{A}, \mathbf{A} \wedge \mathbf{B}), \mathbf{B}) \qquad (8)$$

One might represent a situation in which note **d** is heard after the sequence **a**, **b** has been heard sounding together with the note **c** (Figure 6B), for example, might be represented as

$$S(S(\mathbf{A}, \mathbf{B}) \wedge \mathbf{C}, \mathbf{D}) \qquad (9)$$

but this formula is also satisfied in situations when note **a** has stopped before note **c** begins. The situation is more properly represented as

$$S(S(\mathbf{A} \wedge \mathbf{C}, \mathbf{B} \wedge \mathbf{C}), \mathbf{D}) \qquad (10)$$

Similarly, a configuration consisting of notes **a**, **b**, **c**, **d**, **e**, and **f** with the sequence **b**, **c** occurring in parallel with the sequence **d**, **e**, both following **a** and preceding **f** (Figure 6C), might be represented by

$$S(S(S(\mathbf{A}, \mathbf{B}), \mathbf{C}) \wedge S(S(\mathbf{A}, \mathbf{D}), \mathbf{E}), \mathbf{F}) \qquad (11)$$

but once again this representation is too loose. As with **a** and **c** in (9), it does not require **b** and **d** to sound together, and there could even be two occurrences of the note **a**. The solution to this is even more complex than (10) because it requires each alternative to be explicitly stated in a disjunction:

$$S(S(S(S(\mathbf{A}, \mathbf{B} \wedge \mathbf{D}), \mathbf{C} \wedge \mathbf{D}), \mathbf{C} \wedge \mathbf{E}) \vee S(S(\mathbf{A}, \mathbf{B} \wedge \mathbf{D}), \mathbf{C} \wedge \mathbf{E}) \vee$$
$$S(S(S(\mathbf{A}, \mathbf{B} \wedge \mathbf{D}), \mathbf{B} \wedge \mathbf{E}), \mathbf{C} \wedge \mathbf{E}), \mathbf{F}) \qquad (12)$$

Furthermore, the complexity of a formula like this would increase exponentially with increasing numbers of notes in indeterminate parallel sequences.

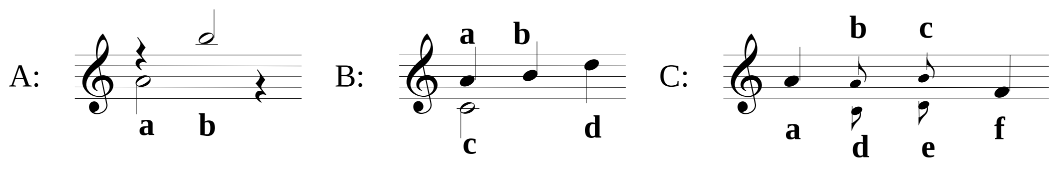

Figure 6. More complex patterns of notes.

Essentially, this kind of representation is capable only of representing sequences of events leading up to the present, where each event is a note, or a number of notes or segments of notes sounding together. It is possible for there to be branching of sequences in the past, as for example when two sequences occur in parallel and terminate at a common point (either **now** or in the past), but these branches cannot rejoin at some point in the past, and so nothing can be inferred about the co-ordination of events between such branching sequences. This is adequate for representing the past of a musical automaton—it might have reached this point by any one of a number of possible paths—and maybe it is entirely appropriate that where it is *required* that simultaneous sounding of notes has occurred, these should be represented explicitly as simultaneously-sounding-notes events with the form $A \wedge B$. When looking to the future, however, the situation is more complex. It might well be that we want to represent an indeterminate but constrained future in which diverging branches do rejoin at some specified event in the future: after first co-ordinating with another player, the triangle player might have a sequence of notes to play which terminate at some defined co-ordinating point with the part of the other player, but that player's part might involve some indeterminate ornamentation or improvisation in between.

2.3 S and U representations and 'nested-branching' representations

The nesting of S (or U) operators in representations of the kind exemplified above suggest a possible parallel with the class of representations described as 'nested-branching' in [1]. It was demonstrated there that fully indeterminate representations using time periods are intractable for such tasks as determining consistency. Some other representations, while tractable, are still too complex for use in real-time performance. For this, it is essential that the complexity of the performance task is linear in the size of the passage of music to be performed, otherwise there must inevitably be a size of passage which is too large for the performance task to keep up with the time of performance. Nested-branching representations can be sequences of symbols with nested occurrences of brackets { and } to indicate the divergence and convergence of branches respectively, with parallel sequences within brackets divided by slashes /. (The configuration of Figure 6C, for example, could be represented as **a** {**b c** / **d e**} **f**.) This kind of representation allows for some indeterminacy (e.g., in the co-ordination of notes **b**, **c**, **d** and **e** in the example above), of a kind commonly found in music, but can nevertheless be performed through an algorithm of linear complexity ([1] pp.214–215). As indicated above, the situation of hearing the note **x** in a sequence … **v w x y z** … can be represented as

$$S(S(S(\ldots, \mathbf{V}), \mathbf{W}), \mathbf{X}) \wedge U(U(U(\ldots, \mathbf{Z}), \mathbf{Y}), \mathbf{X}) \qquad (13)$$

but the power of nesting in the nested-branching representation, which allows us to replace a single event with a configuration of events, which might be a sequence or a set of parallel sequences, is not actually available here. It is possible to replace the first event with a sequence (e.g., the first ellipsis (…) in (13) could be replace by S(**A**, **B**)), and similarly the last event. Furthermore, the first or last event could be replaced by a parallel sequence of events represented as a conjunction of S terms (e.g., S(**A**, **B**) ∧ S(**C**, **D**)), and similarly for the last event. Other events, however, can only be replaced by sets of single coinciding events.

For example, $\mathbf{W}$ could be replaced by $\mathbf{A} \wedge \mathbf{B}$, to mean that notes **a** and **b** both follow **v** and precede **x**. However, to replace $\mathbf{W}$ by S(**A**, **B**) does not characterise a situation in which the sequence **a**, **b** follows **v** and precedes **x**. The formula S(**V**, S(**A**, **B**)) states that since **V** was true, S(**A**, **B**) has been true, but S(**A**, **B**) is true only at times when **B** has been true since **A** was true. If the sequence **v**, **a**, **b** has been heard, there have been times since **v** when **b** has not been sounding (i.e., while **a** was sounding), and hence S(**A**, **B**) has been false. We might try to redeem the situation by replacing **W** not by S(**A**, **B**) but by $\mathrm{S}(\mathbf{A}, \mathbf{B}) \vee \mathrm{U}(\mathbf{B}, \mathbf{A})$. While this formula is continuously true since **v** until the end of the sequence **v**, **a**, **b**, the formula $\mathrm{S}(\mathbf{V}, \mathrm{S}(\mathbf{A}, \mathbf{B}) \vee \mathrm{U}(\mathbf{B}, \mathbf{A}))$ does not properly characterise the situation of being at the end of the sequence **v**, **a**, **b**, because it is also true before the end (while **a** is sounding) and true at the end of the configuration {**v** / **a**} **b** (i.e., with **v** and **a** sounding together).

The essential problem is that we cannot define a formula using S and U which characterises a sequence of events and yet is continuously true throughout the interval of time occupied by that sequence. In fact, this is as it should be, because the idea of temporal logic formulae using S and U is that they should characterise a particular situation in time. The essence of a *sequence* of events is that the situation at the end of the sequence is not the same as the situation at the beginning, so we cannot expect a single formula with S and U to both characterise a sequence and to be true at both its end and its beginning. If we want to replace a single term in an in-time temporal-logic formula with a term representing a complete sequence of events, then that term must be an out-of-time representation. To stand as a single term, the change which constitutes the sequence of events cannot be reflected in a change in the truth value of the term, and so the time of the sequence is not the same as the time of the representation.

3 Temporal measurement in dense time

Modern approaches to human temporal behaviour (see [24], for example) favour models based on oscillators. The same idea can be incorporated into the definition of temporal connectives to add a dimension of measurement to the kinds of representations seen in formulae (7) to (12). It is assumed that there is a single oscillator, whose phase at time t is given by $\alpha(t)$ $(0 \leq \alpha(t) < 1)$. A new connective SY is like S in the sense that its second argument must have been continuously true since its first, and like Y in that this first argument must have been true in the previous time unit.

$$\begin{aligned}
\mathrm{SY}(A, B) =_{\mathrm{def}} {} & \exists t((t < \mathbf{now}) \wedge ((\alpha(t) \geq \mathbf{p_1}) \vee (\alpha(t) < \mathbf{p_2})) \wedge \\
& \quad \exists s((s < t) \wedge (\alpha(s) < \mathbf{p_1}) \wedge \forall u((s \leq u \leq t) \rightarrow A(u))) \wedge \\
& \quad \forall v(((t < v < \mathbf{now}) \rightarrow \alpha(v) \neq \mathbf{p_1}) \vee \\
& \qquad\qquad \exists w(((t < v < w) \rightarrow \alpha(v) \neq \mathbf{p_1}) \wedge ((w \leq v < \mathbf{now}) \rightarrow \alpha(v) \neq \mathbf{p_2}))) \wedge \\
& \qquad ((t < v < \mathbf{now}) \rightarrow B(v)))) \qquad (14)
\end{aligned}$$

The basic idea is that the oscillator marks the beats of a piece of music. Ideally a beat occurs each time the phase is 0, but practically the beat, or notes intended to co-ordinate with the beat, will fall within a range of phase about 0, determined by the constants $\mathbf{p_1}$ and $\mathbf{p_2}$ $(0 < \mathbf{p_2} < \mathbf{p_1} < 1)$. The definition states that SY(A, B) is true if, at the last time when the oscillator was at a phase between $\mathbf{p_1}$ and $\mathbf{p_2}$, A was true, and since then B has been continuously true. The definition for UT(A, B) is similar but looking into the future.

Now the behaviour of the triangle player required to play a note directly after the cue in Figure 2 can be defined by

$$\mathrm{SY}(\mathrm{SY}(\mathrm{SY}(\mathbf{A}, \mathbf{true}), \mathbf{B}), \mathbf{C}) \rightarrow \mathrm{UT}(\mathbf{X} \wedge \mathbf{STRIKE} \wedge \mathbf{Y}, \mathbf{C}) \qquad (15)$$

(Here **true** effectively stands for ‘don’t care what event happens’.)

The period of the oscillator marking the beats should be determined by the playing of the part(s) with which the triangle player must co-ordinate, or by a conductor, who can be considered to be another player whose 'playing' is silent but which still consists of events with which the triangle player must co-ordinate. An implementation of formula (15) in Pd which takes the beat from the length of the note A is given in Figure 7, and the objects SY, SY1 and UT in Figure 8. The innermost SY is represented by SY1 to fix the period of the oscillator from the length of the note A. The oscillator is implemented by the 'timerOsc' object, which sends 1 on a background 'beat' channel when the phase passes $\mathbf{p_1}$ (here set to 0.8) and then 0 when it passes $\mathbf{p_2}$ (here set to 0.2). It also regularly sends its current phase on another channel (used by the UT object). When the SY object receives 0 on its left inlet, it emits 0. If it receives 1, it emits 1 if the right inlet had previously received 1 and if this inlet had received 1 before the last beat (i.e., the previous note had to be sounding before the last beat) and if the beat channel has not received 0 since the last 1 (i.e., the beat has not 'passed'). If these conditions are not met, it emits 0. If they are met, it will emit 0 when the next beat passes (unless all the conditions are true once again on the next beat; note that these objects do not handle repeated notes well—to do so requires dealing not just with concepts of notes sounding but also of notes beginning).

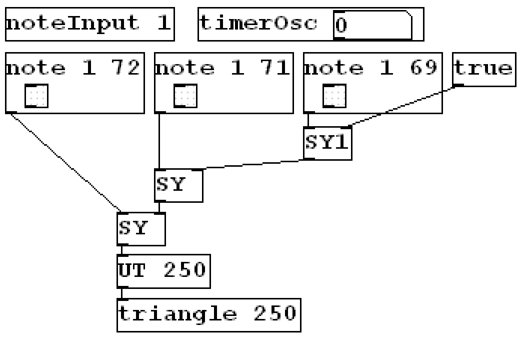

Figure 7. Pd implementation of a solution using SY and an oscillator.

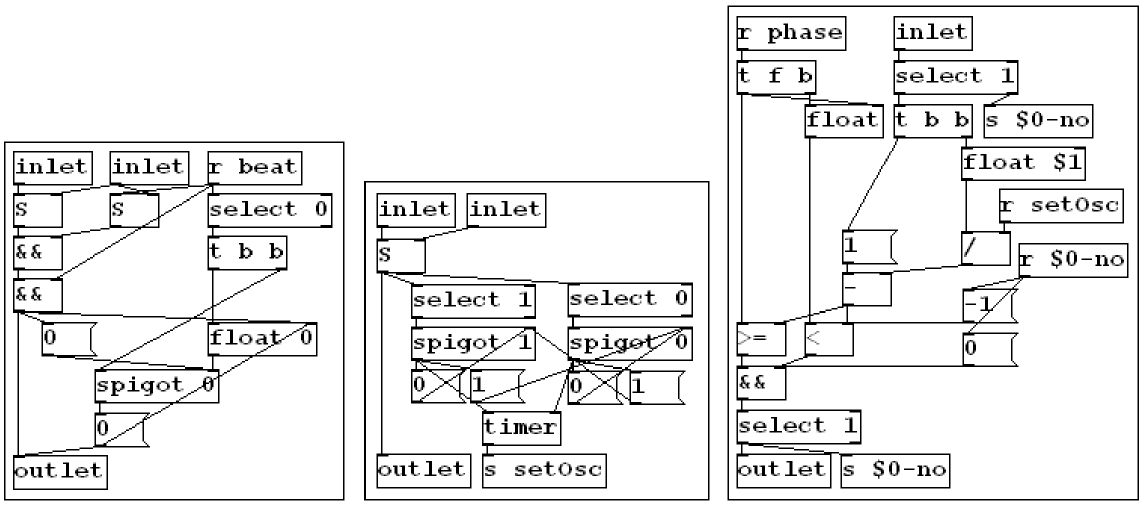

Figure 8. Pd implementations of SY, SY1, and UT.

The UT object takes an argument to specify the amount of anticipation time required on its output (here 250 milliseconds, to match the 250 milliseconds of roll of the 'triangle' object). It listens for a 1 on its inlet, computes the phase of the oscillator at which to trigger output so that it will co-ordinate with the next occurrence of phase 0 (i.e., the ideal beat), and, provided that the inlet has not received 0, sends a bang when the phase of the oscillator passes the required value.

Unlike Figure 4, the software in Figure 7 does not require precise playing: the triangle sounds whenever the notes A, B and C are played in order and with approximately equal duration. However, this is not the behaviour normally required of a triangle player to sound the triangle after the cue in Figure 2. It might be, for example, that the other player sounds A, B and C as quavers (i.e., half the length of the beat), or that they are the correct duration but off the beat (i.e., syncopated). The mistake is in causing the period of the oscillator to be defined by the duration of the note A. If Figure 2 occurs at the beginning of the piece, one has no alternative, but if Figure 2 instead occurs later in the piece, the period of the beat will have already been established earlier. The events of notes **a**, **b** and **c** should *influence* the beat (the other player might be speeding up, for example), but they should not necessarily *define* it. In a more realistic implementation, the oscillator which models the beat of a performance will pick up its period from the playing (or conducting, if there is any, but recall that a conductor is effectively just like another player) and then adapt that period to changes in the course of the playing.

Software to model this process has been a topic of research for some decades (see [25]), and different models perform well in different situations. The example Pd software associated with this article[*] includes a crude beat tracker based on the basic idea of [26]. When used in place of the 'timerOsc' object in Figure 7, and when 'SY1' is replaced by 'SY', the result is the most satisfactory implementation of the solutions to the triangle-player problem offered here. It is not ideal, but the problems which remain are in the beat tracker (which can easily lose track of the proper beat) and in the degree of looseness or precision in where the beat falls (the triangle sometimes sounds when rhythms which clearly deviate from even crotchets have been played, but shortening the critical interval of the 'beat' requires the player to be unnaturally precise in timing). The software thus illustrates that the logic of SY associated with an adaptive beat tracker does allow a specification of a solution to the triangle-player problem.

3.1 Completeness and decidability of SY and UT

It is obvious from inspection of the definitions that SY(A, B) implies S(A, B), and UT(A, B) implies U(A, B). The opposite is not true, however, and indeed the whole point of introducing SY and UT was to restrict the truth of statements with these operators to situations where notes were co-ordinated with a beat. If one can know that every note starts and stops within a beat and no note extends over more than one beat, then SY does indeed become equivalent to S. The condition can be stated as follows:

$$\begin{aligned}&\forall X\exists s\exists t\exists u((s < t < u) \wedge ((\alpha(s) \geq \mathbf{p}_1) \vee (\alpha(s) \leq \mathbf{p}_2)) \wedge (\mathbf{p}_2 < \alpha(t) < \mathbf{p}_1) \wedge \\ &\qquad ((\alpha(u) \geq \mathbf{p}_1) \vee (\alpha(u) \leq \mathbf{p}_2)) \wedge {\sim}X(s) \wedge \\ &\quad \forall v(((s < v \leq t) \rightarrow ((\alpha(v) \neq \mathbf{p}_1) \wedge X(v))) \wedge ((t \leq v < u) \rightarrow ((\alpha(v) \neq \mathbf{p}_2) \wedge X(v)))) \wedge {\sim}X(u)) \quad (16)\end{aligned}$$

Subject to this condition, the completeness results for logics with S and U apply also to a logic with SY and UT also. Its decidability is also similarly contingent on the decidability of the condition. The condition quantifies over notes, but the number of notes in any finite formula of SY and UT will be finite and a decision procedure which examines each in turn is feasible. If the starting and finishing time of each note can be determined, then this condition is decidable in an ontology of linear time, and so is the entire logic. (Actually, for true equivalence to S/U, the condition needs to be weaker for notes which are referred to only in the first or second argument of an operator, restricting only the end or start time of those notes.)

[*] Downloadable from http://www.lancs.ac.uk/staff/marsdena/software/timing2007

SY and UT can also be related to Y and T. If the discrete times of the ontology for Y/T are associated with the time intervals between consecutive beats in SY/UT, then, subject to the same condition, SY implies Y and UT implies UT. The reverse is true also if we ensure that there are no gaps between notes at each beat.

In real music, notes do extend over more than one beat, and they do not all start and stop on a beat. The first of these situations can be accommodated by representing single notes as a sequence of separate tied notes, each lasting a beat. The second can be accommodated as described below, allowing a sequence of notes to occur between two beats.

4 Metre-based representations and nesting out-of-time representations

The problem identified in section 2.3 about incorporating nested-branching representations easily into this kind of modal framework can be solved in an ontology of time periods rather than time points. As before, the truth value of propositions depends on the time with respect to which they are evaluated, but this time is now a period rather than a point, and intuitively we can think of the situation described by the proposition holding over a period rather than applying at a point in time. The special time **now** must also be conceived as a period rather than a point, and we might think of it being short or long depending on how closely we wish to focus on the detail of events. Relations between time periods are more complex than those between points (which consist only of $<$, $=$ and $>$ and their disjunctions $\leq$, $\geq$ and $\neq$). In the definitions below, a symbolic notation for period relations is used ([1], pp.58–60), but each case is explained as necessary.

The operators SY and UT can be redefined with reference to a set of periods M which identifies a continuous sequence of (at least notionally) equal-duration beats. These periods are analogous to the intervals between successive instants when the oscillator of the definition (14) is at phase 0 (i.e., the periods of the oscillation), or to the beats of a piece of music.

$$\mathrm{SY}(A, B) =_{\mathrm{def}} \exists s \exists t \exists m((m \in M) \wedge (m \text{ -<=-< } \mathbf{now}) \wedge A(s) \wedge (s \text{ <<>> } m) \wedge B(t) \wedge (t \text{ =>- } m) \wedge (t \text{ -<= } \mathbf{now})) \quad (17)$$

Period relations used in these definitions have the following meanings: x -<=-< y is true when y occurs wholly within x, or when x and y are equal; x <<>> y is true if y follows x without a break; x =>- y is true if x and y start together but y does not end before x; and x -<= y (the reverse of x =>- y) is true if x and y end together but x does not start after y. For SY(A, B) to be true at the period **now**, first we must identify the period m, which we will call the 'current beat', which is the member of the set of periods M which covers **now**. Then there must be a period s at which A is true, meeting m, and a period t at which B is true, starting or equal to m (=>-) and ending with or equal to **now** (-<=). UT(A, B) is similar, but s is met by m and t starts with or is equal to **now** and ends or is equal to m. A diagrammatic representation of these relations is given in Figure 9.

The period **now** must not coincide with more than one of the periods in the set M of periods representing beats, otherwise the first part of the definition in (17) above can never be true: there will not exist a period m which is a member of M within which **now** is wholly contained. We must assume, therefore, that when the end of a beat occurs, the **now** period jumps to the beginning of the following beat. Indeed, the neatest interpretations will arise when the **now** periods coincide exactly with beats or subdivisions of beats (a topic discussed briefly below).

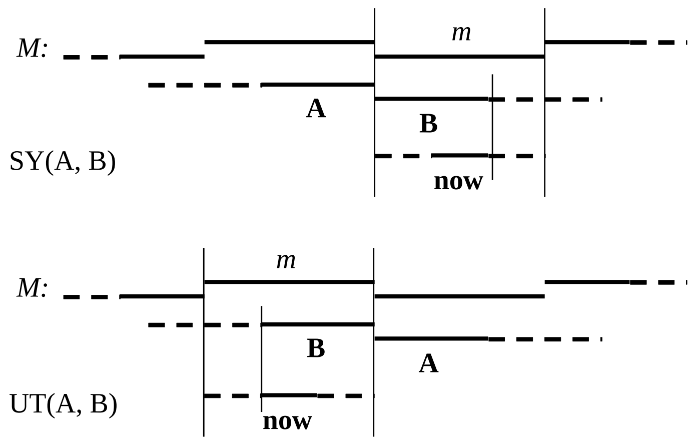

Figure 9. Diagrammatic representation of the relations SY(A, B) and UT(A, B) in period time.

For SY(*A*, *B*) to be true with respect to a time period **now** which is at the end of a beat *m*, then *B* must be true throughout that beat (and similarly if **now** is at the beginning of a beat in the case of UT(*A*, *B*)). This will always be the case for nested instances such as the inner term SY(*A*, *B*) in SY(SY(*A*, *B*), *C*) because, by definition (17), for SY(SY(*A*, *B*), *C*) to be true at time **now**, there must exist a time period *s* meeting the current beat *m* at which SY(*A*, *B*) is true. Since the set of beat-periods *M* is continuous, there must exist a beat immediately before *m* (let us call it *m*1) whose end (at least) coincides with the end of *s*, so, by the observation above, SY(*A*, *B*) must be true throughout the beat *m*1.

This is particularly important when it comes to nesting out-of-time representations within SY and UT operators because the nesting means that the truth of these nested representations is always evaluated with respect to a period which is an entire beat, facilitating the definition of the meaning of out-of-time representations in logical terms. In the definitions below, a propositional variable such as *A* can take a value which is either a simple proposition representing an event (such as the sounding of a note), a compound proposition using SY and/or UT, or an out-of-time representation with enclosing curly brackets {} (equivalent to the 'metrical event structures' described in [1] pp.99–100). In the case of a note-like event, it is natural to think that if a proposition is true with respect to a particular period, then it is also true with respect to each of its subperiods. The same is true of compound propositions whose last term is a simple proposition. The purpose of nested out-of-time representations, however, is precisely to represent situations which do not persist throughout a period but in which truth varies through the period. Thus a term '**simple**(*A*)' is employed in the first definition below which identifies either propositions which represent a single event or compound propositions whose last term is simple. (Recall that *A*(*t*) means '*A* evaluated with respect to time *t*'.)

$\mathbf{simple}(A) \rightarrow (A(t) \leftrightarrow \forall s((t \mathrel{\text{-<=-<}} s) \rightarrow A(s)))$ (18)

$\{A\}(t) \leftrightarrow A(t)$ (19)

$\{A \,/\, B\}(t) \leftrightarrow \{A\}(t) \wedge \{B\}(t)$ (20)

$\{A\ B\}(t) \leftrightarrow \exists r \exists s((t \mathrel{\text{=<<}} r) \wedge (r \mathrel{\text{<<>>}} s) \wedge (t \mathrel{\text{<<=}} s) \wedge (r =_d s) \wedge A(r) \wedge B(s))$ (21)

$\{A\ B\ C\}(t) \leftrightarrow \exists q \exists r \exists s((t \mathrel{\text{=<<}} q) \wedge (q \mathrel{\text{<<>>}} r) \wedge (r \mathrel{\text{<<>>}} s) \wedge (t \mathrel{\text{<<=}} s) \wedge (q =_d r =_d s) \wedge$
$A(q) \wedge B(r) \wedge C(s))$ (22)

…

Formula (18) defines the condition of persistence: if *A* is a simple proposition true at time *t*, then it is true at all subperiods of *t*, and inversely if it is true at all subperiods of a period *t*, then it is true at *t*. Formula (19) handles the redundant case of a single event (whether complex or simple). Formula (20) covers cases of events occurring in parallel. (It can be applied recursively for more than two events or sequences occurring in parallel.) Formulae

(21), (22) and the ellipsis cover the cases of successive events. Time periods for each constituent event are identified which follow each other immediately (identified by the '<<>>' relations) and of which the first starts ('=<<') and the last finishes ('<<=') the period of the overall sequence (period *t*), and which are all of equal duration (indicated by the relation '$=_d$'). Since, by the argument above, any representation of this type nested within the first argument of an operator SY or UT will be evaluated with reference to a beat-period in place of *t*, the effect of these definitions is that the interpretation is indeed equivalent to that of the metrical event structures described in [1]: each top level structure enclosed within brackets {} corresponds to a beat, and sequences of events or structures with brackets correspond to equal subdivisions of the beat.

With SY, UT and the definitions of out-of-time structures, we can now neatly represent situations in which notes are not all of the same duration. For example, **x** and **y** together following the sequence **a**, **b**, **c**, **d**, with the rhythm ♩ ♫ ♩ ♩) can be represented by the formula

$$\mathrm{SY}(\mathrm{SY}(\mathrm{SY}(\mathbf{A}, \mathbf{true}), \{\mathbf{B}\ \mathbf{C}\}), \mathbf{D}) \rightarrow \mathrm{UT}(\mathbf{X} \wedge \mathbf{STRIKE} \wedge \mathbf{Y}, \mathbf{D}) \tag{23}$$

On the other hand, a situation where notes **c** and **d** are the shorter notes (and the rhythm is ♩ ♩ ♫ ♩), should be defined without an out-of-time representation, since **now** falls within the subdivided period, and so we cannot assume that the left-hand side will be evaluated with respect to a period equivalent to an entire beat. The solution is to adapt the SY operator so that it can use not just the set of periods representing beats, but also sets of periods which represent subdivisions of beats. Thus a superscript is introduced to the SY (and UT) operators to represent by how much the period should be subdivided. This models the behaviour of a triangle player who, in a situation like this, directs attention not at the beats but at half beats. In the case of SY^n (or UT^n), the period of the oscillator is divided by *n*, or the periods of *M* are each subdivided into *n* equal-duration periods. SY^1 (or UT^1) is thus equivalent to SY (or UT). (*n* must be a constant and a natural number and will normally only have the value 1, 2, 3, or their multiples.) Requiring a triangle strike and the note **y** following the sequence **a**, **b**, **c**, **d**, with the rhythm ♩ ♩ ♫ ♩ can now be represented by

$$\mathrm{SY}^2(\mathrm{SY}^2(\mathrm{SY}(\mathrm{SY}(\mathbf{A}, \mathbf{true}), \mathbf{B}), \mathbf{C}), \mathbf{D}) \rightarrow \mathrm{UT}^2(\mathbf{X} \wedge \mathbf{STRIKE} \wedge \mathbf{Y}, \mathbf{D}) \tag{24}$$

Note that formula (24), and others like it, can be translated to involve only one value of superscript by splitting events occurring within the scope of operators:

$$\mathrm{SY}^2(\mathrm{SY}^2(\mathrm{SY}^2(\mathrm{SY}^2(\mathrm{SY}^2(\mathrm{SY}^2(\mathbf{A}, \mathbf{true}), \mathbf{A}), \mathbf{B}), \mathbf{B}), \mathbf{C}), \mathbf{D}) \rightarrow \mathrm{UT}^2(\mathbf{X} \wedge \mathbf{STRIKE} \wedge \mathbf{Y}, \mathbf{D}) \tag{25}$$

In fact, it will always be possible to express the equivalent of any formula by using as superscript to SY and UT the least common multiple of all the superscripts used in that formula. If one ignores ornaments (on which players are not generally required to co-ordinate), music notation always expresses durations in integer multiples and divisions (modulated by changes in tempo). Thus a lowest common multiple can always be determined from a pre-existing score, and so the completeness and decidability results adumbrated in section 3.1 above apply here also.

5 Conclusion

Three different levels of logical description of musical timing requirements have been proposed: one assuming discrete time determined by the representing mechanism, one using dense time and assuming the existence of an oscillator to mark out phases of time passing, and one using an ontology of time periods and assuming the existence of a set of beat periods with which events are coordinated. The three levels correspond to different levels of focus. The lowest is appropriate for the low-level design of musical automata where the mechanism imposes some quantisation of time (e.g., in the rate of an analogue-to-digital converter). The dense-time oscillator level is appropriate for higher-level implementation in, perhaps, a class of automata which operate with different actual quantisations of time. The period-time metre level is appropriate for describing human cognition of timing. The differences in complexity are seen where the most complex formulae are required at the lowest, discrete, level of representation, but the least needs to be assumed in the implementation mechanism. At the intermediate level, an oscillator mechanism is required for implementation, and at the highest level this mechanism needs to be very much more complex in a satisfactory implementation, but the representation formulae can be simplest. Translations between the levels of representation are defined, and they come eventually full circle where the metre-based representation can be seen to be equivalent to a discrete-time representation with a long time quantum.

Metre has not been demonstrated to be *essential* for coordination in music, but these formulations do suggest its significance in facilitating coordination. Perhaps this is why it is common in many kinds of music to have a layer of music which clearly defines the beat (e.g., a drum part). It is possible for timings in ensemble performance to be determined by other means. In Lutosławski's *Preludes and Fugue for 13 Solo Strings*, for example, players watch a conductor for a signal to stop repeating a segment of music. In Cage's *Two*^{2} for two pianos, the pianists must wait for each other to complete one segment before they start another. However, I do not know of any case where two musicians are required to cause sounds to happen *at the same time* without using metre to facilitate co-ordination. Furthermore, the data on production of polyrhythms (patterns of note occurring at more than one regular interval) [27] suggests that musicians produce such rhythms by subdividing a single beat, and that only simple subdivisions can be reliably produced.

References

[1] Marsden, A. (2000). *Representing Musical Time: A Temporal-Logic Approach*. Lisse: Swets & Zeitlinger.

[2] Kunst, J. (1978). *Making Sense in Music: An Enquiry into the Formal Pragmatics of Art*. Ghent: Communication & Cognition.

[3] Kunst, J. & Van den Bergh, H. (1984). The analysis of musical meaning: a theory and an experiment. *Interface*, *13*, 75–106.

[4] Leman, M. (1985). Dynamical-hierarchical networks as perceptual memory representations of music. *Interface*, *14*, 125–164.

[5] Leman, M. (1986). A process model for musical listening based on DH-networks. *CC-AI, Journal for The Integrated Study of Artificial Intelligence, Cognitive Science and Applied Epistemology*, *3*, 225–239.

[6] Leman, M. (1989). Adaptive dynamics of musical listening. *Contemporary Music Review*, *4*, 347–362.

[7] Camurri, A. (1993). Applications of artificial intelligence methodologies and tools for music description and processing. In G. Haus (ed.) *Music Processing,* Oxford: Oxford University Press, 233–266.

[8] Camurri, A., Frixione, M. & Innocenti, C. (1994). A cognitive model and a knowledge representation system for music and multimedia. *Journal of New Music Research*, *23*, 317–347.

[9] Camurri, A., Haus, G. & Zaccaria, R. (1986). Describing and performing musical processes by means of Petri nets. *Interface*, *15*, 1–23.

[10] Haus, G. & Rodriguez, A. (1993). Formal music representation; a case study: the model of Ravel's Bolero by Petri nets. In G. Haus (ed.) *Music Processing*. Oxford: Oxford University Press, 165–232.

[11] Gabbay, D.M., Finger, M. & Reynolds, M. (2000). *Temporal Logic: Mathematical Foundations and Computational Aspects (v.2)*. Oxford: Oxford University Press.
[12] Rescher, N. & Urquhart, A. (1971). *Temporal Logic*. Vienna & New York: Springer.
[13] Van Benthem, J.F.A.K. (1983). *The Logic of Time*. Dordrecht: D.Reidel.
[14] Galton, A. (1987). Temporal logic and computer science; an overview. In A. Galton (ed.) *Temporal Logics and their Applications*. London: Academic Press, 1-52.
[15] Gabbay, D.M., Hodkinson, I. & Reynolds, M. (1994). *Temporal Logic: Mathematical Foundations and Computational Aspects (v.1)*. Oxford: Oxford University Press.
[16] Zhou, C., Hoare, C.A.R. & Ravn, A.P. (1991). A calculus of durations. *Information Processing Letters*, *40*, 269–276.
[17] Fränzle, M. (1996). Synthesizing controllers from duration calculus. In B. Jonsson & J. Parrow, *Formal Techniques in Real-Time Fault-Tolerant Systems* (Lecture Notes in Computer Science, no. 1135), Berlin: Springer-Verlag, 168–187.
[18] Rueda, C. & Valencia, F. (2004). On validity in modelization of musical problems by CCP. *Soft Computing*, *8*, 641–648.
[19] Rueda, C., Assayag, G. & Dubnov, S. (2006). A concurrent constraints factor oracle model for music. XXXII Conferencia Latinoamericana de Informática CLEI 2006, Santiago, August 2006. http://mediatheque.ircam.fr/articles/textes/Rueda06a/, last accessed 9 May 2007.
[20] Desainte-Catherine, M. & Allombert, A. (2005). Interactive score: a model for specifying temporal relations between interactive and static events. *Journal of New Music Research*, *34*, 361–374.
[21] Gabbay, D.M. (1987). The declarative past and imperative future. In *Temporal Logic in Specification*, Berlin: Springer-Verlag, 407–448.
[22] Marsden, A. (1992). Modelling the perception of musical voices: a case study in rule-based systems. In A. Marsden & A. Pople (Eds.) *Computer Representations and Models in Music*, London: Academic Press, 239–263.
[23] Puckette, M. (n.d.). Pure Data (computer software), http://crca.ucsd.edu/~msp/software.html, last accessed 9 May 2007.
[24] Beek, P.J., Peper, C.E. & Daffertshofer, A. (2000). Timekeepers versus nonlinear oscillators: how the approaches differ. In P. Desain & L. Windsor (eds.), *Rhythm Perception and Production*, Lisse: Swets & Zeitlinger, 9–33.
[25] Dixon, S. (2001). Automatic extraction of tempo and beat from expressive performances. *Journal of New Music Research*, *30*, 39–58.
[26] Toiviainen, P. (2001). Real-time recognition of improvisations with adaptive oscillators and a recursive Bayesian classifier. *Journal of New Music Research*, 137–147.
[27] Summers, J.J. (2000). The learning and transfer of multifrequency patterns. In P. Desain & L. Windsor (eds.), *Rhythm Perception and Production*, Lisse: Swets & Zeitlinger, 69–80.