A Process Algebra for Stochastic Music Composition

BRIAN J. ROSS
Brock University
Department of Computer Science
St. Catharines, Ontario, Canada L2S 3A1
phone: (416) 688-5550 ext. 4284
fax: (416) 688-3255
email: bross@sandbanks.cosc.brocku.ca

May 2, 1995

Abstract:
A process algebra suitable for modelling stochastic music composition is presented. The algebra is based on Toft's weighted sequential calculus of communicating systems, which is a probabilistic version of Milner's SCCS. The algebra permits the design of stochastically-based automata which perform a variety of musically interesting behaviours, including reactive behavior between processes (horizontal composition), and simultaneity of musical events (vertical composition). The most notable feature of the algebra is the ability to mathematically model complex musical systems with an algebra having a rigorous but simple formal semantics. This permits formal analysis of musical systems, as well as a basis for deriving well-founded implementations. This algebra complements other models based in trace theory, Petri Nets, Markov processes, and other stochastic music programming languages.

Key words: process algebra, concurrency, computer music, stochastic composition

1 Introduction

The perspective of music as a concurrent activity is well established. Music models using Petri nets [HR88, HS92] and algebraic interleaving [CT88, Che90] are rooted in the concurrent computation paradigm to different extents. This paper introduces a process algebra model of music called the Musical Weighted Synchronous Calculus of Communicating Systems, or MWSCCS. Process algebra are popular contemporary formalisms for modelling and analyzing concurrent systems [Hoa85, Mil89]. They are widely accepted as effective models of concurrency largely because of the high degree of abstraction possible with them, along with their intuitive "programming language" feel.

MWSCCS considers music as comprised of streams of discrete, concurrent behaviors that can be observed and reacted upon by musical processes. These streams denote the horizontal structure of music – the relative order of events with respect to each other. Work in process algebra has shown that this level of abstraction is convenient for analyzing complex concurrent behavior, since it denotes the essence of a systems activity without being encumbered with structural details of the system itself. The vertical component of music – event simultaneity – is naturally denoted by the event domain of the synchronous algebra. A major feature of MWSCCS is its set of probabilistic primitives for representing events that adhere to stochastic event distributions. These stochastic primitives opens the possibility of novel techniques for automated stochastic composition.

2 The MWSCCS Process Algebra

An *agent* or *process* is an abstract mechanism whose behavior is characterised by discrete actions. Process algebra are mathematical formalisms for modelling processes. There are many varieties of process algebra in the literature (eg. [Hoa85, Hen88, Mil89]), which establish different perspectives on what precisely constitutes observable behavior. As we will show, process algebra have natural application to music, since musical events are observed (heard) and reacted upon by observers (musicians, audiences).

We call the algebra MWSCCS (Musical Weighted Sequential Calculus of Communicating Systems). It is an enhancement of Tofts' WSCCS algebra [Tof90], which is a probabilistic version of Milner's Synchronous Calculus of Communicating Systems [Mil89]. It adds to WSCCS by defining a musical action domain and useful musical operators. A comprehensive discussion of WSCCS or SCCS algebra is beyond the scope of this paper, and we refer the reader to the aforementioned sources for detailed treatments.

2.1 Domain of Musical Events

The most basic identifiable atomic event or communication signal from a process is a *particle.* The set of particles $\mathcal{A}$ is defined by an application, and particles themselves are usually named to indicate their function in a system. By convention, we do not use capital letters in particles. In the domain of music, a useful set of particles to include in $\mathcal{A}$ are the set of tones on a piano keyboard. We could denote this set $Notes \subseteq \mathcal{A}$ as

$$Notes = \{a_1,\ a_1^\sharp,\ a_1^\flat,\ b_1,\ b_1^\sharp,\ b_1^\flat,\ \cdots\}$$

where each note is indexed with its octave. Other examples of particles are the set of pitch frequencies, fundamentals, chords, and percussive hits. Each particle a has a corresponding *complement* $\overline{a}$. Informally, an overbarred particle represents an *output* communication, while corresponding non-overbarred equivalent is an *input* communication. We define $\overline{\overline{a}} = a$.

Particles are used to construct *actions*. An action is an event occurring at one moment in time. The universe of actions for a system is an Abelian group over the particles $\mathcal{A}$ (including complements), with an identity particle $\mathbf{1}$. We let $\{a, b, ...\}$ range over $\mathcal{A}$, and $\{\alpha, \beta, ...\}$ range over Act. Given an integer i, the notation a^i represent i copies of a. If i is negative, then a is complemented (ie. $a^{-2} = \overline{a}^2$). If $i = 0$, then a is an empty string. Then, for each action $\alpha \in Act$, a unique product

$$\alpha \ = \ a_1^{i_1} \cdots a_k^{i_k}$$

of non-zero powers of particles $a_i \in \mathcal{A}$ can be constructed. When $k = 0$ in the above, then $\alpha = \mathbf{1}$, which represents a silent delay. We ignore the order of particles in an action, which means that actions denotes bags of particles.

Given an action $\alpha \ = \ a_1^{z_1} \cdots a_k^{z_k}$, whenever a particle and its complement reside in α, they absorb with one another and disappear. For example, the action $a\overline{a}bcd^3\overline{d}^2\overline{e}$ is equivalent to $bcd\overline{e}$. Each absorption that occurs represents the occurence of a synchronizing communication amongst two or more processes. Note how this conforms to the standard interpretation of arithmetic product:

$$aa^{-1}bcd^3d^{-2}e^{-1} \ = \ bcde^{-1}$$

An action that is useful in music is one that denotes termination. Being able to identify the termination of musical activity – be it a melodic line, a major movement of a symphony, or the end of a whole musical composition – is critical in music. Without such a termination signal, other musicians cannot know for certain that a colleague has finished playing; the absence of playing from that person is not foolproof, since he or she may be taking an extended dramatic break (or may have forgot their place). A general solution adopted here is to establish a conventional signal for when musical activity ends [HM88]. The action $\surd$ represents the termination of a musical process. When $\overline{\surd}$ is transmitted from a process, it is the final action seen from that process, and no further activity will be exhibited from it. However, the process should remain silently alive, rather than literally terminate. This technical requirement will be clearer later, as it will be shown that literal termination is inherited amongst multiple musical events, while passive but active silence permits multiple musicians to remain playing. Processes that use this termination convention are said to be *well-behaved.* Note that not all processes may terminate. As long as a well-behaved process does not emit $\overline{\surd}$, then it has not terminated.

2.2 Action Frequency and Priority

A major feature of MWSCCS is the ability to prescribe relative frequencies and priorities to processes. Process expressions may be prefixed with the term

$$n\omega^k$$

where n and k are natural numbers, with $n > 0$. This denotes that the expression has a relative frequency of n and a priority level of k. We let the set of weights for a system be $\mathcal{W} = \cup_i\{w_i\}$. The expression $n\omega^0$ is abbreviated as n, and $1\omega^k$ as ω^k. For example, the term $2\omega a\overline{b}$ has a relative frequency of 2 with respect to other processes it is being selected from, and has a priority of 1. We distinguish the frequency number 1 from the identity action **1** mentioned in section 2.1. For convenience, we denote $1\omega^0 a = 1a$ as just a.

The following rules are used when adding and multiplying weights. Assuming $k > k'$,

$$\begin{aligned} n\omega^k + m\omega^{k'} &= n\omega^k = m\omega^{k'} + n\omega^k \\ n\omega^k + m\omega^k &= (n+m)\omega^k = m\omega^k + n\omega^k \\ n\omega^k * m\omega^{k'} &= (nm)\omega^{k+k'} = m\omega^{k'} * n\omega^k \end{aligned}$$

We also use the following to extract frequency and priority values from weight terms:

$$\mathcal{N}(n\omega^k) = n \qquad \Omega(n\omega^k) = k$$

2.3 Algebra Syntax and Semantics

The language $\mathcal{E}$ of agent expressions is defined by the grammar

$$\mathcal{E} \ ::= \ \mathbf{0} \mid X \mid \alpha.E \mid E\lceil A \mid E\backslash A \mid E[f] \mid \sum_{i\in I} w_i E_i \mid E \times F \mid \mu_i \tilde{X}\tilde{E}$$

where $E, F \in \mathcal{E}$, X is a process variable, $a \in \mathcal{A}$, $w_i \in \mathcal{W}$, $\alpha \in \mathcal{A} \cup \mathcal{W}$, $A \subseteq \mathcal{A}$, and f is a particle renaming function $f : a \to b$ for $a, b \in \mathcal{A}$.

Tofts defines the semantics of WSCCS using natural deduction-style rules of inference (figure 1). These transition rules are sequents in which the expression below the line can be inferred when the premise expressions above it hold [Plo81, Hen90]. The rules define two least relations, $\rightarrow \subseteq \mathcal{E} \times Act \times \mathcal{E}$ and $\mapsto \subseteq \mathcal{E} \times \mathcal{W} \times \mathcal{E}$. Using the inference rules, transitions such as $E \xrightarrow{\alpha} E'$ are derivable, which denotes the transformation of an expression E into a derivative E', yielding the action α in the process. This α represents a simultaneous event, which in music may be a collection of notes from different musicians playing together. We therefore expect to hear multiple actions emitted sequentially in a composition:

$$P \xrightarrow{\alpha_1}\xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_k} \cdots$$

$$
\begin{array}{cc}
\overline{\alpha.E \xrightarrow{\alpha} E} & \overline{\sum_{i\in I} w_i E_i \overset{w_i}{\mapsto} E_i} \\[2ex]
\dfrac{E \xrightarrow{\alpha} E' \qquad F \xrightarrow{\beta} F'}{E \times F \xrightarrow{\alpha\beta} E' \times F'} & \dfrac{E \overset{w}{\mapsto} E' \qquad F \overset{v}{\mapsto} F'}{E \times F \overset{wv}{\mapsto} E' \times F'} \\[2ex]
\dfrac{E \xrightarrow{\alpha} E' \qquad \alpha \in L}{E\lceil L \xrightarrow{\alpha} E'\lceil L} & \dfrac{E \overset{w}{\mapsto} E' \qquad does_A(E')}{E\lceil L \overset{w}{\mapsto} E'\lceil L} \\[2ex]
\dfrac{E \xrightarrow{\alpha} E' \qquad \alpha \in A}{does_A(E)} & \dfrac{E \overset{w}{\mapsto} E' \qquad does_A(E')}{does_A(E)} \\[2ex]
\dfrac{E \xrightarrow{\alpha} E'}{E[f] \xrightarrow{f(\alpha)} E'[f]} & \dfrac{E \overset{w}{\mapsto} E'}{E[f] \overset{w}{\mapsto} E'[f]} \\[2ex]
\dfrac{E_i\{\mu_i\tilde{x}.\tilde{E}/\tilde{x}\} \xrightarrow{\alpha} E'}{\mu_i\tilde{x}.\tilde{E} \xrightarrow{\alpha} E'} & \dfrac{E_i\{\mu_i\tilde{x}.\tilde{E}/\tilde{x}\} \overset{w}{\mapsto} E'}{\mu_i\tilde{x}.\tilde{E} \overset{w}{\mapsto} E'}
\end{array}
$$

Figure 1: Transitional semantics

This $\rightarrow$ operative represents a *synchronous global clock*. In other words, the algebra does not specify the absolute timing between the succession of events α_i and α_{i+1}, but rather, their relative order. It is reasonable to assume that this ordering would commonly correspond to some fixed time signature. The $\mapsto$ relation is similar to $\rightarrow$, except that a weight term is generated instead. Generally speaking, $\mapsto$ transitions are of importance when analyzing and manipulating expressions with weights, while $\rightarrow$ expressions are the means by which visible phenomena of interest are transmitted to observers.

We briefly overview the intuitive semantics of the operators, and give alternative equational characterizations of the transitional semantics for some of the operators. The *null* process $\mathbf{0}$ is the process that does nothing. The *prefix* operator in $a.E$ represents the process that can perform the action or weight α, thereafter becoming the process E:

$$\alpha.E \ = \ \alpha.(E)$$

We sometimes omit the "." notation, and shorten $\alpha.\mathbf{0}$ to α.

The *permission* operator in $E\lceil A$ denotes the process that performs the particles that are members of the set A. In effect, the permission operator prunes all actions not in A:

$$(a.E)\lceil A = \begin{cases} a.(E\lceil A) & if \;\; a \in A \\ \mathbf{0} & otherwise \end{cases}$$

When a weighted summation is involved, then only the terms that produce particles from A are permitted:

$$(\sum_{i\in I} w_i E_i)\lceil A \;=\; \sum_{j\in J} w_j(E_j\lceil A) \quad where \;\; J=\{i\in I:(E_i\lceil A)=a.E'\}$$

The *restriction* operator in $E\backslash A$ is the converse of permission, except that it lists the actions which cannot be generated by E. In fact, $E\backslash A \;=\; E\lceil(\mathcal{A}-A)$.

The *relabelling* operator in $E[f]$ renames particles according to f, while leaving weights alone:

$$(wa.E)[f] \;=\; w\, f(a).(E[f])$$

For example, $E[b/a]$ replaces particle a with b in each action emitted by E.

The *choice* operator $\sum_{i\in I} w_i E_i$ represents the choice of execution of a set of processes E_i indexed by a finite set I. This expression may be rewritten

$$w_{i_1}E_{i_1}+w_{i_2}E_{i_2}+\cdots+w_{i_k}E_{i_k}$$

where $I=\cup_k\{i_k\}$. The E_i chosen is influenced by the frequency $\mathcal{N}(w_i)$ and priority $\Omega(w_i)$ of its weight term w_i. Assuming all the processes under consideration have the same priority, then a process $w_jE_j(j\in I)$ is chosen with probability

$$\frac{\mathcal{N}(w_j)}{\sum_{i\in I}\mathcal{N}(w_i)}.$$

Note that frequencies are preserved by constant factors,

$$\sum_i \mathcal{N}(w_i) \;=\; \sum_i c\,\mathcal{N}(w_i)$$

for some constant c. This means that a is chosen with probability $\frac{2}{5}$ in both $2a+3b$ and $6a+9b$. Process priority $\Omega(w)$ comes into consideration in that only the processes with maximum priority values k in $\Omega(w_i)$ are considered, *providing* these processes have permission to execute from the environment. Therefore, in the expression

$$(2\omega^2a+3\omega^3b+3\omega^2c+4\omega^3d+\omega^3e)\lceil\{a,b,c,d\}$$

the only choices considered are

$$3\omega^3b+4\omega^3d$$

since they are permissible, and their priority of 3 is the highest.

The *parallel composition* operator in $E\times F$ forces concurrent processes E and F to synchronously communicate with one another. We let $E\times F=F\times E$. If both E and F are unweighted, then a new action is formed by their combined actions:

$$a.E\times b.F \;=\; ab.(E\times F)$$

If only one is weighted, then that weight is applied to the combined results of their derivative:

$$a.E \times \sum_{i \in I} w_i F_i \;=\; \sum_{i \in I} w_i (a.E \times F_i)$$

If both are weighted, then a new weight is calculated for their composite result using the multiplication equation:

$$\sum_{i \in I} w_i E_i \times \sum_{j \in J} v_j F_j \;=\; \sum_{(i,j) \in I \times J} w_i v_j (E_i \times F_j)$$

Here, $I \times J$ refers to the Cartesian product of index sets I and J, which means that there can be a combinatorially large number of possible behaviors from the expression. Fortunately, the permission and restriction operators permit control of these expressions. When executing such a process, only one of these behaviors will be generated.

Wherever a process variable X is found, the $\mathcal{E}$ expression bound to it is used. Process equations $\mu_i \tilde{X} \tilde{E}$ represent the least fixpoint of the mutually recursive equations $\tilde{E}$ using process variables $\tilde{X}$. An operator $\stackrel{\text{def}}{=}$ is used to define processes with equational expressions over E, which is akin to defining a subroutine in a traditional programming language. The μ_i notation is normally dropped in favour of using the process name defined in a process equation. An example process definition is:

$$P(X,Y) \stackrel{\text{def}}{=} (X \times Y) \backslash \{c_0, c_1, c_2, c_3, ...\}$$

This process lets X and Y play together, but forbids the playing of any C note.

Given two WSCCS expressions, it is important to be able to compare them with respect to their behaviors. For example, we may like to check if a simpler musical expression performs the same type of composition as a larger, more abstruse one. Of the number of equivalence schemes possible for WSCCS, we will use *relative bisimulation.* Two processes P and Q are relative bisimular equivalent, written $P \approx Q$, if:

1. For all $w \in \mathcal{W}$ and integers c_1, c_2, $P \stackrel{w}{\mapsto} P'$ iff $Q \stackrel{w}{\mapsto} Q'$, $c_1 w = c_2 v$ and $P' \approx Q'$.

2. For all $a \in Act$, $P \stackrel{a}{\to} P$ iff $Q \stackrel{a}{\to} Q'$ and $P' \approx Q'$.

The first case makes sure that processes are equivalent with respect to their relative frequency, and the second case assures that equivalent processes generate the same actions. It can be shown that $\approx$ is a congruence relation.

2.4 Useful Definitions

One convenience is the ability to freely use schematic expressions over actions. For example, we might be interested in denoting a note that is two semi-tones higher than a given note α. We will denote this as $\overline{\alpha + 2}$. Similarly, the expression

$$P(\alpha_k) \stackrel{\text{def}}{=} \alpha_k \overline{\alpha_{k+12}}$$

represents a process that hears a note α, and then plays the note one octave higher.

Sometimes it is useful to model asynchronous processes. Unlike the processes in the previous section, asynchronous processes may nondeterministically wait or stall before elliciting their observed behaviour. We can represent an asynchronous process by the following meta-operator:

$$\bullet(P) \stackrel{\text{def}}{=} P + \mathbf{1}.\bullet(P)$$

Here, P can either execute, or wait. As can be seen in this recursive definition, an asynchronous process may very well stall forever, although this is statistically unlikely.

The strict sequential execution of musical events is often required. To play sequentially, one musician's musical solo should end before the next musician's solo commences. A sequential composition operator ";" is useful for this purpose:

$$X \; ; \; Y \stackrel{\text{def}}{=} (X[c/\surd] \times \bullet(c.Y)) \setminus \{c\}$$

where c is a new particle not generated by X or Y. The ; operator takes two processes X and Y as arguments, where X must be well-behaved. Here, X will play until it is done, at which time it generates the termination signal $\overline{\surd}$. This is renamed to c when it is seen, and Y has this same c prefixed to it. When used with the restriction of c, the expression disallows c from appearing. Fortunately, when $\bar{c}$ is finally generated by the left-side, it synchronizes with c on the right, and that particle disappears (ie. $\bar{c}c = 1$). Then Y may proceed to play. Note that we do not want X to literally die (reduce to $\mathbf{0}$), or else the whole expression will quit, since $\mathbf{0} \times E = \mathbf{0}$. Rather, X should remain silently alive in order for the composite expression to remain active. For example, X may be equivalent to:

$$Silent \stackrel{\text{def}}{=} \mathbf{1}.Silent$$

2.5 More advanced extensions

Previous discussions have considered musical events to be discrete actions running synchronously with an unspecified clock. This view of music is adequate for discussing important semantic aspects of the algebra, but is unrealistic for modelling real music. For example, in

$$P \stackrel{\bar{c}}{\rightarrow}\stackrel{\bar{e}}{\rightarrow}\stackrel{\bar{g}}{\rightarrow} P'$$

the notes c, e and g are taken as occuring one after another, with unspecified but finite durations. In reality, music is much more complex: notes have different durations, dynamics, and timbres. Many formal models of these phenomena exist in the literature. For example, [Bel92, CT88] suggest means for representing temporal properties of music.

Enhanced musical activity can be modelled by suitably enriching the action domain. A basic enhancement that permits note duration is to use explicit *Note on* and *Note off* signals for each note [CT88], which has the advantage of mapping to MIDI. For example, let a represent an on-note, and a' an off-note. With this notation, if we presume that the global clock is running at a quarter-note duration, then the transition

$$P \xrightarrow{\overline{c}}\xrightarrow{\mathbf{1}}\xrightarrow{\mathbf{1}}\xrightarrow{\mathbf{1}}\xrightarrow{\overline{ec'}}\xrightarrow{\overline{e'g}}\xrightarrow{\mathbf{1}}\xrightarrow{\overline{g'}} P'$$

represents a sequence consisting of a whole note c, quarter note e, and half-note g. The wait action $\mathbf{1}$ is used to designate timing intervals. This process is written in MWSCCS as:

$$P \stackrel{\mathrm{def}}{=} \overline{c}.\mathbf{1}.\mathbf{1}.\mathbf{1}.\overline{ec'}.\overline{e'g}.\mathbf{1}.\overline{g'}.P'$$

Another possibility mentioned in [CT88] is to use *Note on* and *Note continue* actions. Then the above passage is denotable as:

$$P \xrightarrow{\overline{c}}\xrightarrow{\overline{c'}}\xrightarrow{\overline{c'}}\xrightarrow{\overline{c'}}\xrightarrow{\overline{e}}\xrightarrow{\overline{g}}\xrightarrow{\overline{g'}} P'$$

Other enhancements to the domain space permit additional important musical features to be represented. To represent multitimbral compositions, a unique $Notes_i$ set can be created for each instrument of interest. Each particle in these sets is designated with an appropriate voice label (akin to MIDI's *channel number*). Actions can also denote other aural and stylistic properties in as much detail as desired. In fact, MIDI data itself would be perfectly acceptable as action particles.

3 Examples

3.1 Simple Melody and Harmony

Consider the expression

$$1\overline{ca} + 2\overline{da}$$

which represents a simple stochastic fragment that plays the chord $\{c, a\}$ with a probability of $1/3$, and $\{d, a\}$ with a probability $2/3$. We can make a nonterminating melody M by defining:

$$M \stackrel{\mathrm{def}}{=} 1\overline{ca}.M + 2\overline{da}.M$$

and then invoking M.

Next, we define an expression that harmonizes with M:

$$Composition \stackrel{\mathrm{def}}{=} (\ M[x/c, y/d]\ \times\ (x\overline{cf}.H + y\overline{dg}.H)\ )\ \backslash\{x, y\}$$

The intension is that H should play f whenever it hears c, and play g whenever d is heard. In order to hear c and d, the expression relabels them so H can hear them, and then play the original notes as played by M. By convention, non-overbarred actions are input or "hearing" actions, and therefore the hearing actions x and y from H are restricted in the final output.

The following shows the final behaviors possible from Composition. It is derived by using the transitional inference rules and derived laws for the MWSCCS operators:

$$
\begin{array}{lll}
& Composition & \\
\approx & (\ M[x/c, y/d]\ \times\ (x\overline{cf}.H + y\overline{dg}.H)\ )\ \backslash\{x,y\} & : definition\ \ Composition \\
\approx & (\ (1\overline{ca}.M + 2\overline{da}.M)[x/c, y/d]\ \times\ (x\overline{cf}.H + y\overline{dg}.H)\ )\ \backslash\{x,y\} & : definition\ \ M \\
\approx & (\ (1\overline{xa}.M + 2\overline{ya}.M)\ \times\ (x\overline{cf}.H + y\overline{dg}.H)\ )\ \backslash\{x,y\} & : relabelling \\
\approx & (\ 1\overline{xa}x\overline{cf}.(M \times H)\ +\ 1\overline{xa}y\overline{dg}.(M \times H) & \\
& \quad +\ 2\overline{ya}x\overline{cf}.(M \times H)\ +\ 2\overline{ya}y\overline{dg}.(M \times H)\ )\ \backslash\{x,y\} & : parallel\ \ composition \\
\approx & (\ 1\overline{acf}.(M \times H)\ +\ 1\overline{xa}y\overline{dg}.(M \times H) & \\
& \quad +\ 2\overline{ya}x\overline{cf}.(M \times H)\ +\ 2\overline{adg}.(M \times H)\ )\ \backslash\{x,y\} & : simplify\ \ particles \\
\approx & 1\overline{acf}.(M \times H)\ +\ 2\overline{adg}.(M \times H) & : restriction
\end{array}
$$

Note how restriction prevents the emmision of terms containing the particles we are using for communication. Also note how the final frequency numerals remain faithful to the stochastic model of the initial expression.

An enhancement of the composition is:

$$Composition \times (Composition[b/c, f^\sharp/a])$$

Here, we play *Composition* alongside a variation of itself that transposes c with b, and a with $f^\sharp$. We could in fact play limitless variations of *Composition* with itself:

$$Composition \times Composition \times (Composition[b/c, f^\sharp/a]) \times \cdots$$

Section 2.5 discusses how to assign a different voice or instrument to each process in this expression.

3.2 A Conductor

Consider again the nonterminating melody M above. We would like to rewrite M so that it can be terminated if desired. Here is one approach:

$$M' \stackrel{\text{def}}{=} 1\overline{ca}.M' + 2\overline{da}.M' + \omega fini.\overline{da}.\overline{ceg}$$

fini is a control action we introduce to force M to quit. The priority of this term is set to $\Omega(\omega) = 1$, which makes it a higher priority than the other terms, whose priorities are 0 $(= \Omega(\omega^0))$. We next define a *conductor*,

$$Conductor \stackrel{\text{def}}{=} \mathbf{1.1.1.1.1.1}.\omega\overline{fini}$$

Then, in the expression,

$$(M'\ \times\ Conductor) \setminus \{fini, \overline{fini}\}$$

The conductor permits M' only 6 phrases, after which it forces it to to quit by expressing a prioritized $\omega\overline{fini}$ action. On the other hand, if we run M' without a conductor,

$$M' \setminus \{fini\}$$

it behaves as before.

3.3 Grammatical composition structure

Using the notion of termination, WSCCS can duplicate the expressiveness of regular grammars, and therefore permits grammatical description of composition [Roa79] For example, we might like to structure a composition as:

$$\begin{array}{rcl}
\mathit{Tune} & \stackrel{\text{def}}{=} & \mathit{Prelude} \;;\; \mathit{Main} \;;\; \mathit{Finale} \\
\mathit{Prelude} & \stackrel{\text{def}}{=} & 2\overline{ccdce}.\overline{\surd}.\mathit{Silent} \;+\; \overline{cdcee}.\overline{\surd}.\mathit{Silent} \\
\mathit{Main} & \stackrel{\text{def}}{=} & \mathit{FirstPart} \;;\; \mathit{SecondPart} \;;\; \mathit{Climax} \;;\; \mathit{Resolution} \\
\mathit{FirstPart} & \stackrel{\text{def}}{=} & 2\overline{ccdce}.\mathit{FirstPart} \;+\; \overline{cdfee}.\mathit{FirstPart} \;+\; \overline{g}.\overline{\surd}.\mathit{Silent} \\
\ldots & & \\
\mathit{Finale} & \stackrel{\text{def}}{=} & \overline{d}.\overline{d}.\overline{b}.\overline{d}.\overline{ceg}.\overline{\surd}
\end{array}$$

Here, there are one of two possible preludes possible, followed by the main body and finale. *FirstPart* has the regular expression form $(a+b)^*c$.

3.4 A more complex example

$$\begin{array}{rcl}
\mathit{Music} & \stackrel{\text{def}}{=} & ((\mathit{Melody}\;\; [x/e_5, x'/e_5', b/\surd] \\
& & \;\times\;\; \mathit{Acc_7_9}(x, e5, b) \setminus \{\alpha, \overline{x}, \overline{x'}, \overline{b}\})\; [y/e^\flat_5, (y')/(e^\flat_5{}'), b/\surd] \\
& & \;\times\;\; \mathit{Acc_7_9}(y, c^\flat_5, b)) \setminus \{\alpha, \overline{y}, \overline{y'}, \overline{b}\} \\
\mathit{Melody} & \stackrel{\text{def}}{=} & (M1 \;;\; M2) \;;\; (M2 \;;\; M1) \;;\; \mathbf{0} \\
M1 & \stackrel{\text{def}}{=} & 2\omega(\overline{c_5e_5g_5}).\mathbf{1}.(\overline{c_5'e_5'g_5'\surd}).\mathit{Silent} \;+\; 1\omega(\overline{c^\flat_5e^\flat_5g_5}).\mathbf{1}.(\overline{c^\flat_5{}'e^\flat_5{}'g_5'\surd}).\mathit{Silent} \\
M2 & \stackrel{\text{def}}{=} & 2\omega(\overline{c_5e^\flat_5g_5}).\mathbf{1}.\mathbf{1}.(\overline{c_5'e^\flat_5{}'g_5'\surd}).\mathit{Silent} \;+\; 1\omega(\overline{c_5e^\flat_5g^\sharp_5}).(\overline{c_5'e^\flat_5{}'g^\sharp_5{}'\surd}).\mathit{Silent} \\
\mathit{Acc_7_9}(\alpha,\beta,\pi) & \stackrel{\text{def}}{=} & \omega^2(\pi\overline{\surd}).\mathit{Silent} \\
& & +\; 3\omega^1\alpha\overline{\beta}(\overline{\beta+7}).\mathit{Rel_7_9}(R, X, 7, \pi) \;+\; 2\omega^1\alpha\overline{\beta}(\overline{\beta-9}).\mathit{Rel_7_9}(R, X, -9, \pi) \\
& & +\; \omega^0\mathbf{1}.\mathit{Acc_7_9}(\alpha,\beta,\pi) \\
\mathit{Rel_7_9}(\alpha,\beta,\gamma,\pi) & \stackrel{\text{def}}{=} & \omega^2(\pi\alpha'\overline{\beta'}\overline{(\beta+\gamma)\surd}).\mathit{Silent} \\
& & +\; \omega^1(\alpha'\overline{\beta'(\beta+\gamma)}).\mathit{Acc_7_9}(\alpha,\beta,\pi) \;+\; \omega^0\mathbf{1}.\mathit{Rel_7_9}(\alpha,\beta,\gamma,\pi)
\end{array}$$

This example uses both grammatical structure, stochastic nondeterminism, and reactive communication. It also uses the more realistic *note on* and *note off* notation suitable for MIDI systems. The main musical process is *Music*, which plays a basic melodic line (*Melody*) along with two accompaniment processes (*Acc_7_9*). Melody uses two subprocesses *M1* and *M2*, which stochastically generate simple chords. *Music* then uses particle renaming when invoking *Acc_7_9*, and restricts the renamed particles so that they will be invisible to the audience. This relabelling events and restricting their emmission is a powerful technique used by process algebras to control communication. The calls to *Acc_7_9* make generous use of argument passing, to make that process as general as possible.

Acc_7_9 works as follows. The body of the process will do three things, which are ranked in priority as follows. Firstly, the ω^2 term checks if *Melody* has terminated, and if so, it quits as well. Next, it checks if *Melody* has generated the event α. If so, it chooses to generate either a note 7 semitones higher, or a note 9 semitones lower. A process *Rel_7_9* is used to release the accompanied tone previously generated. Finally, if neither of the above cases have occurred, *Acc_7_9* waits until the next communicated event.

The use of priorities in *Acc_7_9* and *Rel_7_9* serve two purposes. Firstly, they allow the processes to behave correctly. For example, if we used the same priority value in all the terms of *Acc_7_9*, it is possible that the silent waiting term might be selected when in fact *Melody* has terminated. This prevents the generation of the note off and termination signal actions. Secondly, from an implementation point of view, priorities can greatly reduce the need for search. The most critical actions to take can be given higher priority.

4 Discussion

As with other process algebra, constructing musical systems with MWSCCS benefits from its abstract view of observable behavior. A primary influence of its design is its use of a small set of powerful process construction operators, combined with an intuitive and flexible model of probability. Using its notion of bisimulation, different stochastic music systems can be compared. Other process algebra in the literature are equally applicable to music. Timed process algebra such as [BB91] are worth investigation, as they permit a more convenient representation of note duration and real-time system timing.

A working implementation of a music programming language based on MWSCCS is currently being undertaken. New issues must be addressed when implementing an algebra like MWSCCS. For example, special attention is required in ensuring that the composition operator ($\times$) is tractable, as a exponential amount of space and execution time can easily occur. We are also discovering new operators useful in a musical context. The hierarchical composibility of process algebra lend itself well to the definition of more powerful operators.

The recognition of music as a concurrent phenomenon is well established, and the application of formal models of concurrency towards music is resulting in novel and practical formalisms. There are three main formal models of concurrency in the literature: Petri nets, trace theory, and process algebra. All three approaches share a common mathematical basis, which is reflected by the fact that they essentially define different varieties of finite automata. As discussed below, both Petri nets and trace theory have been applied successfully to different extents, and this paper therefore completes the "triad" by modelling music with process algebra.

Chemillier's algebraic model of music explores the use of traces in musical modelling, analysis, recognition, and composition [Che87, CT88, Che90]. He defines a *solfege* semantics of music, which uses a free monoid algebra to denote traces of musical events. Action prefixing and simultaneity are denoted by string concatenation and set union respectively. Superposition, interposition, and literal shuffle operators are used to compose traces. The solfege, being founded in algebraic interleaving, is intimately related to process algebra. WSCCS and the solfege are both free monoids. In fact, the solfege superposition and literal shuffle operators have a very similar interpretation to this papers synchronous composition operator; its notion of simultaneity of actions via set composition corresponds to the Abelian groups used in the action domain of WSCCS; and Chemillier uses a equivalence relation similar to those in process algebra for comparing traces. WSCCS extends the solfege model by introducing a practical calculus for constructing systems of musical processes. The notions of nondeterministic choice, action renaming and restriction, synchronized communication, stochastic weights, and hierarchical process definition are useful tools for music modelling and composition. It would be interesting to apply some or all of these ideas with Chemillier's elegant algebra.

Haus *et al* have derived a powerful Petri net (PN) based model of music [HR88, HS92]. A PN is an automaton that denotes multiple simultaneous stimuli and reactions in a concurrent system. Haus defines a PN programming language for music, which is used for implementing concurrent music systems. By tweaking a PN's parameters, different varieties of compositions arise. The differences between the PN model and WSCCS result from the inherent differences between PNs and process algebra [Nie87]. PNs are said to model "true concurrency", in that they make explicit all the simultaneous causal relationships amongst components of a concurrent system. Process algebra view concurrency more abstractly, and causal relationships amongst components are not explicit denoted. The control mechanisms used by PNs and process algebra are also quite different in flavour. PN's use token passing to effect concurrent reactions. In process algebra, processes reacts directly with communications from other processes in the environment. Process algebra are generally considered to be more specification-oriented than PNs, which might be a point in their favour in a music composition setting.

It is also worth mentioning the similarity between WSCCS and other conventional music programming languages [LA85], especially those using object-orientation. The main advantage shared by WSCCS and Haus's PN language in comparison to other music languages are their well-defined mathematical foundation, which permits direct formal analyses of systems built with them.

Acknowledgement: Support through NSERC Operating Grant 0138467 is gratefully acknowledged.

References

[BB91] J.C.M. Baeten and J.A. Bergstra. Real time process algebra. *Journal of Formal Aspects of Computing Science*, 3(2):142–188, 1991.

[Bel92] B. Bel. Symbolic and Sonic Representations of Sound-Object Structures. In M. Balaban, K. Ebcioğlu, and O. Laske, editors, *Understanding Music with AI*, pages 64–109. MIT Press, 1992.

[Che87] M. Chemillier. Monoide Libre et Musique: Deuxierme Partie. *Theoretical Informatics and Applications*, 21(4):379–418, 1987.

[Che90] M. Chemillier. Solfege, Commutation Partielle et Automates de Contrepoint. *Math. Inf. Sci. Hum.*, 28(110):5–25, 1990.

[CT88] M. Chemillier and D. Timis. Toward a theory of formal musical languages. In *Proceedings ICMC*, pages 175–183, Cologne, 1988.

[Hen88] M. Hennessy. *Algebraic Theory of Processes.* MIT Press, 1988.

[Hen90] M. Hennessy. *The Semantics of Programming Languages – An Elementary Introduction Using Structural Operational Semantics.* John Wiley and Sons, 1990.

[HM88] E.C.R. Hehner and A.J. Malton. Termination Conventions and Comparative Semantics. *Acta Informatica*, 25:1–14, 1988.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice–Hall, 1985.

[HR88] G. Haus and A. Rodriguez. Music Description and Processing by Petri Nets. In G. Rozenberg, editor, *Advances in Petri Nets (LNCS 340)*, pages 175–199. Springer-Verlag, 1988.

[HS92] G. Haus and A. Sametti. ScoreSynth: A System for the Synthesis of Music Scores Based on Petri Nets and a Music Algebra. In D. Baggi, editor, *Computer-Generated Music*, pages 53–77. IEEE Computer Society Press, 1992.

[LA85] G. Loy and C. Abbott. Programming Languages for Computer Music Synthesis, Performance, and Composition. *Computing Surveys*, 17(2):235–265, June 1985.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice Hall, 1989.

[Nie87] M. Nielsen. CCS - and its Relationship to Net Theory. In W. Brauer, editor, *Petri Nets: Application and Relationship to Other Models of Concurrency (LNCS 255)*, pages 393–415. Springer-Verlag, 1987.

[Plo81] G. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Computer Science Department, Aarhus University, September 1981.

[Roa79] C. Roads. Grammars as Representations for Music. *Computer Music Journal*, 3(1):48–55, 1979.

[Tof90] C. Tofts. A Synchronous Calculus of Relative Frequency. In J.C.M. Baeten and J.W.Klop, editors, *CONCUR 90*, Amsterdam, The Netherlands, 1990. Springer-Verlag LNCS 458.