Noname manuscript No.
(will be inserted by the editor)

On Validity in Modelization of Musical Problems by CCP

Camilo Rueda, Frank Valencia

[1] Universidad Javeriana-Cali, Colombia
e-mail: crueda@atlas.puj.edu.co
[2] Dept. of Information Technology, Uppsala University, Sweden
e-mail: fvalenci@brics.dk

Abstract

We show how the ntcc calculus, *a model of* temporal concurrent constraint programming *with the capability of modeling asynchronous and non-deterministic timed behavior, can be used for modeling real musical processes. In particular, we show how the expressiveness of ntcc allows to implement complex interactions among musical processes handling different kinds of partial information. The ntcc calculus integrates two dimensions of soft computing: a horizontal dimension dealing with partial information and a vertical one in which non determinism comes into play. This integration is an improvement over constraint satisfaction and concurrent constraint programming models, allowing a more natural representation of a variety of musical processes. We use the nondeterminism facility of ntcc to build weaker representations of musical processes that greatly simplifies the formal expression and analysis of its properties. We argue that this modeling strategy provides a "runnable specification" for music problems that eases the task of formally reasoning about them. We show how the linear temporal logic associated with ntcc gives a very expressive setting for formally proving the existence of interesting musical properties of a process. We give examples of musical specifications in ntcc and use the linear temporal logic for proving properties of a realistic musical problem.*

1 Introduction

From a broad perspective we view soft computing as those techniques pertaining to systems that can best be described as a collection of processes dealing with partial information. What "partial" means depends on the particular application. It can refer to being able to use partial knowledge of a state of affairs and to perform different kinds of guessing (bounded or unbounded non determinism, probabilistic choices, approximate answers). In this view, concurrency also belongs to this realm since it deals with partial information on the ordering of events. So does constraint programming which is based on the very idea of computing with *predicates* expressing different degrees of knowledge about variable values. Concurrent constraint (CC) process calculi ([9]) provide formal grounds to the integration of concurrency and constraints so that non trivial properties of concurrent systems can be expressed and proved. They are thus natural simulators to gain experience on different soft computing techniques.

We view musical experience as constructed from very complex interactions among a great number of concurrent processes acting on different musical dimensions. We share the view in [6] that as a result of those interactions musical structures such as rhythm and harmony *emerge*. Concurrent processes occurring in music exhibit a rich variety of synchronization schemes, calling into play different degrees of precision (i.e. partial information) about temporal or harmonic relations involving them. The complexity of musical phenomena poses a great challenge to any computational formalism. We think that a suitable CC process calculus should provide a convenient framework to get insights into the right models to cope with this challenge.

We thus borrow concepts and techniques from concurrent processes modeling to define suitable computational calculi and analyze their behavior in real musical settings. What we gain from this *low level* approach is twofold. One the one hand, we are able to ground the development of music composition tools on a very precise formal foundation and by this means proposing coherent higher level musical structures and operations. On the other hand, our model can give us clues for constructing formal proofs of interesting properties of a given musical process.

In [1], *PiCO*, a concurrent processes calculus integrating constraints and objects is proposed. Musical applications are programmed in a visual language having this calculus as its underlying model. Since there is no explicit notion of time in *PiCO* some musical processes, in particular those involving real time activity, are difficult to express. Moreover, reasoning about musical processes behavior in *PiCO* can be difficult since there is no formal logic associated with it.

We propose using a temporal non deterministic concurrent calculus (*ntcc*, see [7]) as a formal base to model timed

musical processes in such a way that their musical properties can be formally proved.

The ntcc calculus inherits ideas from the tcc model [8], a formalism for reactive concurrent constraint programming. In tcc time is conceptually divided into *discrete intervals (or time-units)*. In a particular time interval, a deterministic ccp process receives a stimulus (i.e. a constraint) from the environment, it executes with this stimulus as the initial store, and when it reaches its resting point, it responds to the environment with the resulting store. Also the resting point determines a residual process, which is then executed in the next time interval.

The tcc model is inherently deterministic and synchronous. Indeed, patterns of temporal behavior such as "the system must output pitch C *within* the next t time units" or "the three voices must eventually output the same note but *there is no bound* in the occurrence time" cannot be expressed within the model. It also rules out the possibility of choosing one among several alternatives as an output to the environment.

A very important benefit of being able to specify non-deterministic and asynchronous behavior arises when modeling the interaction among several components running in parallel, in which one component is part of the environment of the others. This is frequent in musical applications. These systems often need non-determinism and asynchrony to be modeled faithfully.

The ntcc calculus is obtained from tcc by adding *guarded-choice* for modeling non-determinism and an *unbounded but finite delay* operator for asynchrony. Computation in ntcc progresses as in tcc, except for the non-determinism and asynchrony induced by the new constructs. The calculus allows for the specification of temporal properties, and for modeling and expressing constraints upon the environment both of which are useful in proving properties of timed systems.

In this essay we are interested in showing how non trivial musical processes calling into action different forms of partial information can be modeled in ntcc. In particular, we implement systems in which the number of notes is a derived property. Systems of this kind are awkward to model using constraint satisfaction because the exact number of variables is not known in advance. We are also interested in modeling musical patterns resulting from the controlled interaction of independent musical agents, such as is the case in several forms of musical improvisation. In this type of systems, state changes in one process following particular local laws must be "partially" synchronized with state changes in other processes. This poses difficulties to CCP models not including state change constructs. In fact, two dimensions of soft computing issues can be identified: a "horizontal" dimension in which agents compute, share and accumulate partial information, and a "vertical" dimension in which they perform non deterministic choices. Very different interpretations of the same musical piece bear witness of the presence of both dimensions in a real score, particularly when improvisation is called into play. Most CCP models include only the horizontal dimension whereas most concurrent process calculi include only the vertical dimension. The ntcc calculus includes both. We show that music processes with non determinism, partial information and state change synchronization are naturally expressed in ntcc. We claim this is a clear advantage of the ntcc calculus.

We also investigate ways in which properties of musical process can be formally proved. We are able to do this thanks to the logical nature of ntcc, which comes to the surface when we consider its relation with linear temporal logic: All the operators of ntcc correspond to temporal logic constructs. In constraint based musical applications, the existence or non existence of a musical structure enjoying given properties is discovered after a time consuming search. The main contributions of this paper are: 1) toshow how the expressiveness of the ntcc model allows simple descriptions of complex systems of interacting musical processes and 2) showing that by modeling a music process in ntcc one inherits a well defined logical inference system (see [7]) that can be used to prove its musical properties (or lack thereof).

2 The Calculus

In this section we present the syntax and an operational semantics of the ntcc calculus. First we recall the notion of constraint system.

Basically, a constraint system provides a signature from which syntactically denotable objects in the language called *constraints* can be constructed, and an entailment relation specifying interdependencies between such constraints. The underlying language $\mathcal{L}$ of the constraint system contains the symbols $\dot{\neg}, \dot{\wedge}, \dot{\Rightarrow}, \dot{\exists}$, `true` and `false` which denote logical negation, conjunction, implication, existential quantification, and the always true and always false predicates, respectively. *Constraints,* denoted by $c, d, \ldots$ are first-order formulae over $\mathcal{L}$. We say that c *entails* d in Δ, written $c \vdash_\Delta d$ (or just $c \vdash d$ when no confusion arises), if $c \dot{\Rightarrow} d$ is true in all models of Δ. For operational reasons we shall require $\vdash$ to be decidable.

Process Syntax. Processes $P, Q, \ldots \in Proc$ are built from constraints $c \in \mathcal{C}$ and variables $x \in \mathcal{V}$ in the underlying constraint system by the following syntax.

$$\begin{aligned} P, Q, \ldots ::= \ & \mathbf{tell}(c) \mid \sum_{i \in I} \mathbf{when}\, c_i \,\mathbf{do}\, P_i \\ & \mid P \parallel Q \mid \mathbf{local}\ x\ \mathbf{in}\, P \\ & \mid \mathbf{next}\, P \mid \mathbf{unless}\ c\ \mathbf{next}\, P \\ & \mid\, !\, P \mid \star P \ . \end{aligned}$$

The only move or action of process $\mathbf{tell}(c)$ is to add the constraint c to the current store, thus making c available to other processes in the current time interval. The guarded-choice

$$\sum_{i \in I} \mathbf{when}\ c_i\ \mathbf{do}\ P_i,$$

where I is a finite set of indexes, represents a process that, in the current time interval, must non-deterministically choose one of the P_j ($j \in I$) whose corresponding constraint c_j is entailed by the store. The chosen alternative, if any, precludes

the others. If no choice is possible then the summation is precluded. We use $\sum_{i\in I}\ P_i$ as an abbreviation for the "blind-choice" process $\sum_{i\in I}\mathbf{when}\,(\mathtt{true})\,\mathbf{do}\,P_i$. We use $\mathbf{skip}$ as an abbreviation of the empty summation and "+" for binary summations.

Process $P \parallel Q$ represents the parallel composition of P and Q. In one time unit (or interval) P and Q operate concurrently, "communicating" via the common store. We use $\prod_{i\in I} P_i$, where I is finite, to denote the parallel composition of all P_i. Process $\mathbf{local}\ x\ \mathbf{in}\ P$ behaves like P, except that all the information on x produced by P can only be seen by P and the information on x produced by other processes cannot be seen by P.

The process $\mathbf{next}\,P$ represents the activation of P in the next time interval. Hence, a move of $\mathbf{next}\,P$ is a unit-delay of P. The process

$$\mathbf{unless}\,c\,\mathbf{next}\,P$$

is similar, but P will be activated only if c cannot be inferred from the current store. The "unless" processes add (weak) time-outs to the calculus, i.e., they wait one time unit for a piece of information c to be present and if it is not, they trigger activity in the next time interval. We use $\mathbf{next}^n(P)$ as an abbreviation for

$$\mathbf{next}(\mathbf{next}(\ldots(\mathbf{next}\,P)\ldots)),$$

where $\mathbf{next}$ is repeated n times.

The operator ! is a delayed version of the replication operator for the $\pi-$calculus ([5]): $!\,P$ represents $P \parallel \mathbf{next}\,P \parallel \mathbf{next}^2 P \parallel \ldots$, i.e. unbounded many copies of P but one at a time. The replication operator is the only way of defining infinite behavior through the time intervals.

The operator $\star$ allows us to express asynchronous behavior through the time intervals. The process $\star\,P$ represents an arbitrary long but finite delay for the activation of P. For example, $\star\,\mathbf{tell}(c)$ can be viewed as a message c that is eventually delivered but there is no upper bound on the delivery time.

We shall use $!_I P$ and $\star_I P$, where I is an interval of natural numbers, as an abbreviation for processes $\prod_{i\in I}\mathbf{next}^i P$ and $\sum_{i\in I}\mathbf{next}^i P$, respectively. For instance, $\star_{[m,n]}P$ means that process P is eventually active between the next m and $m+n$ time units, while $!_{[m,n]}P$ means that P is always active between the next m and $m+n$ time units.

Operational Semantics.

Operationally, the currently available information is represented as a constraint $c \in \mathcal{C}$, so-called *store*. The operational semantics is given by considering transitions between *configurations* γ of the form $\langle P, c\rangle$. We define Γ as the set of all configurations. The formal definition (see [7] for details) introduces two reduction relations, one representing *internal transitions* and the other *observable transitions*.

The *internal transition* $\langle P, c\rangle \longrightarrow \langle Q, d\rangle$ should be read as "P with store c reduces, in one internal step, to Q with store d ". The *observable transition* $P \stackrel{(c,d)}{\Longrightarrow} Q$ should be read as "P on input c from the environment reduces, in one time unit, to Q and outputs d to the environment ". Such an observable transition is defined in terms of a sequence of internal transitionstransitions $\langle P, c\rangle \longrightarrow^* \langle Q', d\rangle$ starting in P with store c and ending in some process Q' with store d. Crudely speaking, to obtain Q we should remove from Q' what was meant to be executed only in the current time interval. Since Q is to be executed in the next time interval we should also "unfold" the sub-terms within $\mathbf{next}\,R$ expressions in Q'. As in tcc, the store does not transfer automatically from one interval to another.

To illustrate reductions in ntcc, consider a musical process, say $!\,P$, that continually outputs either C (MIDI=60) or E (MIDI=64) until another process (the conductor) Q signals the end. Process $!P \parallel Q$, for P and Q as defined below, models the example.

$$\begin{aligned} P &\stackrel{\text{def}}{=} \mathbf{when}\,(Go=1)\,\mathbf{do}\,(\mathbf{tell}\,(Note=60) \\ &\quad + \mathbf{tell}\,(Note=64)\,) \\ &\quad \parallel \mathbf{unless}\,End=1\,\mathbf{next}\,\mathbf{tell}\,(Go=1) \\ Q &\stackrel{\text{def}}{=} \mathbf{tell}\,(Go=1)\ \parallel\ \star\mathbf{tell}\,(End=1) \end{aligned}$$

Then there is a sequence of internal transitions

$$\begin{aligned} &\langle !\,P, Go=1\rangle \longrightarrow \langle P \parallel \mathbf{next}\,!\,P, Go=1\rangle \\ &\longrightarrow^* \langle \mathbf{tell}\,(Note=65) \parallel \mathbf{next}\,!\,P, Go=1\rangle \\ &\longrightarrow \langle \mathbf{next}\,!\,P, Note=65 \wedge Go=1\rangle \not\longrightarrow \ldots \end{aligned}$$

Initially the store contains constraint $Go=1$ (which, as described below, will be added to thestore by Q). Replicated process $!P$ then creates a copy of P and schedules itself for the next time unit. Process P chooses note E (the store gets $Go=1\wedge Note=65$). No further reductions are possible in the current time unit. Two processes, $!P$ and $\mathbf{tell}\ Go=1$ are scheduled for the next time unit. So, in the case $P \parallel Q$, for an arbitrary (number of time units) $n>1$, the following are possible transitions:

$$\begin{aligned} &\langle !P \parallel\ Q, true\rangle \longrightarrow^* \\ &\langle \mathbf{next}\,!P\ \parallel\ \mathbf{next}\,\mathbf{tell}\ Go=1 \\ &\parallel \mathbf{next}^n\mathbf{tell}(End=1), Go=1\wedge Note=64\rangle \end{aligned}$$

and

$$\begin{aligned} &!P \parallel Q \stackrel{(\mathbf{true}, Go=1\wedge Note=64)}{\Longrightarrow} \\ &\quad !P\ \parallel\ \mathbf{tell}\ Go=1 \parallel \mathbf{next}^{n-1}\mathbf{tell}(End=1). \end{aligned}$$

The first one is the internal transition relation, whereas the second is the observable transition. Thus $!P$ continually outputs either pitch C or E for an arbitrary number n of time units until the constraint $End=1$ is put in the store.

As mentioned before, an important feature of the ntcc model is that there is a logic associated with it. We describe next this logic.

3 A Logic of ntcc Processes

A relatively complete formal system for proving whether or not an ntcc process satisfies a linear-temporal property was introduced in [7]. In this section we summarize these results.

Temporal Logic.

We define a linear temporal logic for expressing properties of ntcc processes. The formulae $A, B, ... \in \mathcal{A}$ are defined by the grammar

$$A ::= c \mid A \Rightarrow A \mid \neg A \mid \exists_x A \mid \circ A \mid \Box A \mid \Diamond A.$$

The symbol c denotes an arbitrary constraint. The symbols $\Rightarrow$, $\neg$ and $\exists_x$ represent temporal logic implication, negation and existential quantification. These symbols are not to be confused with the logic symbols $\dot{\Rightarrow}$, $\dot{\neg}$ and $\dot{\exists}_x$ of the constraint system. The symbols $\circ$, $\Box$, and $\Diamond$ denote the temporal operators *next*, *always* and *sometime*. Given a property A (e.g. $x > 10$) the intended meaning of $\circ A$, $\Box A$ and $\Diamond A$ is that the property holds, in the next time unit, always and eventually, respectively. We use $A \vee B$ as an abbreviation of $\neg A \Rightarrow B$ and $A \wedge B$ as an abbreviation of $\neg(\neg A \vee \neg B)$.

We shall say that process P *satisfies* A iff every infinite sequence that P can possibly output satisfies the property expressed by A. A relatively complete proof system for assertions $P \vdash A$, whose intended meaning is that P satisfies A, can be found in [7]. We shall write $P \vdash A$ if there is a derivation of $P \vdash A$ in this system.

The following notion will be useful for discussing properties of our musical examples.

Definition 1 (Strongest Derivable Formulae) *A formula A is the strongest temporal formula derivable for P if $P \vdash A$ and for all A' such that $P \vdash A'$, we have $A \Rightarrow A'$.*

Note that the strongest temporal formula of a process P is unique modulo logical equivalence. We give now a constructive definition of such formula.

Definition 2 (Strongest Formula Function) *Let the function $stf : Proc \rightarrow \mathcal{A}$ be defined as follows:*

$$\begin{array}{ll}
stf(\mathbf{tell}(c)) & = c \\
stf(WHEN(c_i, P_i)) & = \left(\bigvee_{i\in I} c_i \wedge stf(P_i)\right) \\
 & \quad \vee \bigwedge_{i\in I} \neg c_i \\
stf(P \parallel Q) & = stf(P) \wedge stf(Q) \\
stf(\mathbf{local}\, x\, P) & = \exists_x stf(P) \\
stf(\mathbf{next}\ P) & = \circ\, stf(P) \\
stf(\mathbf{unless}\ c\ \mathbf{next}\ P) & = c \vee \circ stf(P) \\
stf(!\, P) & = \Box\, stf(P) \\
stf(\star\, P) & = \Diamond\, stf(P).
\end{array}$$

where the expression $WHEN(c_i, P_i))$ represents process $\sum_{i\in I} \mathbf{when}\,(c_i)\,\mathbf{do}\,P_i$.

From [7] it follows that $P \vdash stf(P)$. From this we have:

Proposition 1 *For every process P, $stf(P)$ is the strongest temporal formula derivable for P.*

Note that to prove that $P \vdash A$ is sufficient to prove that $stf(P) \Rightarrow A$. In addition, the proof system described in [7] gives extra mechanisms for carrying out proofs of process properties.

4 Musical examples

We introduce the idea of modeling music processes in ntcc by defining rhythm patterns which are synchronized in a variety of ways. In the examples given below we use *cells*. Cells are variables whose values can be updated in subsequent time units. Notation $x := exp$ means that cell x is to be updated with value exp in the next time unit. Notation $x : (v)$ initializes a cell x with value v. Cells are not extended constructs since they can be easily encoded in standard ntcc (see [7]). Let us first define a "metronome" process:

$$\begin{array}{l}
M[tick, count, \delta] \stackrel{\text{def}}{=} \\
\quad !(\mathbf{when}\,(count \bmod \delta) = 0\,\mathbf{do}\, tick := tick + 1 \\
\qquad\qquad\qquad\qquad\qquad\qquad || \, count := 1 \\
\quad +\mathbf{when}\,(count \bmod \delta) > 0\,\mathbf{do}\, count := count + 1)
\end{array}$$

One could think of $M[tick, count, \delta]$ as a process that beats time every δ time units. This process could interact with an acceleration process:

$$\begin{array}{l}
Accel_k[signal, \delta] \stackrel{\text{def}}{=} \\
\quad !\mathbf{when}\,(signal = 1) \wedge \delta > k\,\mathbf{do}\,\delta := \delta - k
\end{array}$$

This process speeds up the ticks of $M[tick, count, \delta]$ by a value of k if some other process, which we shall refer to as $Control_{(s,d,e)}[tick, signal]$, tells $signal = 1$:

$$\begin{array}{l}
Control_{(s,d,e)}[tick, signal] \stackrel{\text{def}}{=} \\
\quad !(\mathbf{when}\,(s \leq tick \leq e \wedge (tick - s) \bmod d = 0) \\
\qquad \mathbf{do} \quad signal = 1)
\end{array}$$

Process $Control_{(s,d,e)}[tick, signal]$ acts as a conductor signaling the start of a variety of events. Each type of event is represented by an instance of the controller using a different "signal" variable. A musical process P is controlled by one of these particular instances when P shares the signal variable of that controller. For example, if the $Accel_k[signal, \delta]$ process share its $signal$ variable with the one in the above $Control$ process, then it would accelerate tempo every d-th tick in the range between ticks s and e. Any controller could instead be made (bounded) non deterministic by triggering it within a summation

$$\sum_{t\in[s,e]} Control_{(t,d,e)[tick,signal]}$$

In this way, for example, tempo change could be made to start at some undetermined point between ticks s and e.

Since a process can launch an instance of any other process, very complex temporal patterns can be achieved. Furthermore, musical processes with local "clocks" can be easily

synchronized to some other global conducting process, as the following "ornament" process illustrate:

$$
\begin{array}{l}
Ornament_n[signal, \delta, signal_1, signal_2] \stackrel{\text{def}}{=} \\
\quad \textbf{when}\, signal = 1\, \textbf{do} \\
\quad (\textbf{local}\, count\, tick)\, (\\
\qquad tick : (0) \,||\, count : (0) \\
\qquad M[tick, count, (\delta \,\text{div}\, n)] \\
\qquad ||\, Control_{(1,1,\delta-1)}[tick, signal_1] \\
\qquad ||\, Control_{(0,1,\delta)}[tick, signal_2]\,)\,)
\end{array}
$$

The above process $Ornament$ is launched by some controller (the one setting its $signal$ parameter). When it starts, it launches a local metronome and two controllers. Variable $signal_1$ is thus set in n equally spaced ticks, starting in the current global one and ending (but not including) the next global tick (i.e the one occurring δ time units later). Variable $signal_2$ is similarly set, but starting after the current global tick and including the next global tick. These could be used to signal, for example, when different types of ornament notes have to be played in between base notes. The nondeterministic variety of $Control$ could be used to achieve more freely controlled ornamentations.

The above scheme defines temporal patterns of conductors *signals*. Any musical process, be it temporal, melodic or harmonic could be synchronized according to those signals. These , in turn, could interact "diagonally" by means of constraints coupled with non determinism:

$$
\begin{array}{l}
Amb_{low,high}[signal, bound] \stackrel{\text{def}}{=} \\
\quad \textbf{when}\, signal = 1\, \textbf{do} \\
\qquad \sum_{n \in [low,high]} low \leq bound \leq n
\end{array}
$$

$$
\begin{array}{l}
Notes_S[signal, bound_1, bound_2, note] \stackrel{\text{def}}{=} \\
\quad !(\textbf{when}\, signal = 1\, \textbf{do} \\
\qquad \sum_{pitch \in S} \textbf{when}\, bound_1 \leq pitch \leq bound_2 \\
\qquad\qquad \textbf{do}\, note = pitch\,)
\end{array}
$$

Process Amb constrains all pitches output from processes synchronized in that tick to be not higher than a non deterministically chosen value $bound$ in the range $[low, high]$. Process $Notes$ non deterministically selects a note from the set S, provided it lays within the allowable range.

The system described above could be instantiated as follows:

$$
\begin{array}{l}
System \stackrel{\text{def}}{=} \\
\quad (\textbf{local}\, tick\, \delta\, count\, signal\, signal_1, signal_2\, signal_a \\
\qquad note\, note_1\, note_2\, bound_1\, bound_2) \\
\quad (count : (0) \,||\, tick : (0)\, \delta : (2) \\
\quad ||\, M[tick, count, \delta] \,||\, Accel_2[signal_a, \delta]|| \\
\quad ||\, Control_{(0,60,120)}[tick, signal_a] \\
\quad ||\, Control_{(0,30,120)}[tick, signal] \\
\quad ||\, Control_{(0,20,120)}[tick, signal] \\
\quad ||\, Control_{(0,12,120)}[tick, signal] \\
\quad ||\, Ornament_4[signal, signal_1.signal_2] \\
\quad ||\, Amb_{(48,55)}[signal, bound_1] \\
\quad ||\, Amb_{(60,72)}[signal, bound_2] \\
\quad ||\, Notes_{[48,72]}[signal, bound_1, bound_2, note] \\
\quad ||\, Notes_{\{60,64,67,71\}}[signal_1, 60, 72, note_1] \\
\quad ||\, Notes_{\{48,50,55,62\}}[signal_2, 48, 62, note_2]\,)
\end{array}
$$

The above system performs a rhythm addition of two eight notes ($d = 30$), a triplet ($d = 20$) and a quintuplet ($d = 12$). Tempo is doubled each quarter note. Base notes are taken in a range supplied by $bound_1$ and $bound_2$, computed by process Amb. Ornament notes before and after the beat are selected from given sets of pitches.

4.1 Conrolled improvisation

This examples models a non trivial controlled improvisation musical system. An interesting feature of this example is that the number of events (notes) to be computed is not known in advance. This makes it very difficult to implement as a constraint satisfaction model. Contrary to this, the problem admits a simple ntcc model. The system can be described as follows: there is a certain number m of musicians (or *voices*), each playing blocks of three notes. Each of them is given a particular pattern (i.e. a list) of allowed delays between each note in the block. A musician can freely choose any permutation of this pattern. For example, given a pattern $p = [4, 3, 5]$ a musician can play his block with time gaps of 5 then 4 and then 3 between the notes. Once a musician has finished playing his block of three notes, he must wait for a signal of the *conductor* indicating that the other musicians have also finished their respective blocks. Only after this he can start playing a new block. The exact time in which he actually starts playing a new block is not specified, but it is constrained to be no later than the sum of the durations of all patterns. For example, given three musicians and patterns $p_1 = [3, 2, 2], p_2 = [4, 3, 5]$ and $p_3 = 3, 3, 4]$ no delays between blocks greater than 29 time units is allowed. The musicians keep playing this way until all of them play a note at the same time. After this, all musicians must stop playing.

In order to model this example we assume that constant sil in the constraint system represents some note value for "silence". Process $M_i, i < m$, models the activity of the i-th musician. When ready to start playing ($start_i = 1$), the i-th musician chooses a permutation (j, k, l) of his given pattern

p_i. Then M_i spawns a process $Play^i_{(j,k,l)}$, thus playing a note at time j (after starting), then at time $j+k$ and finally at time $j+k+l$. Constraint $c_i[note_i]$ specifies some value for $note_i$, different from sil. After playing a block, the i-th musician signals termination by setting cell $flag_i$ to 1. Furthermore, upon receiving the $go = 1$ signal, the i-th musician eventually starts a new block no later than pdur which is a constant representing the sum of the durations of all patterns.

$$
\begin{array}{l}
M_i \stackrel{\text{def}}{=} \\
\quad !(\mathbf{when}\, start_i = 1\, \mathbf{do} \\
\quad\quad \sum_{(j,k,l)\in perm(p_i)} (Play^i_{(j,k,l)} \\
\quad\quad\quad\quad || \, \mathbf{next}^{j+k+l} \\
\quad\quad\quad\quad\quad (flag_i = 1 \\
\quad\quad\quad\quad\quad || \, *[0, pdur]\mathbf{tell}\,(start_i = 1)))
\end{array}
$$

$$
\begin{array}{l}
Play^i_{(j,k,l)} \stackrel{\text{def}}{=} \\
![0, j-1]\mathbf{tell}\,(note_i = sil) \,||\, \mathbf{next}^j\mathbf{tell}(c_i[note_i]) \\
||\, ![j+1, j+k-1]\mathbf{tell}\,(note_i = sil) \\
\quad\quad\quad\quad\quad\quad ||\, \mathbf{next}^{j+k}\mathbf{tell}(c_i[note_i]) \\
||\, ![j+k+1, j+k+l-1]\mathbf{tell}\,(note_i = sil) \\
\quad\quad\quad\quad\quad\quad\quad\quad ||\, \mathbf{next}^{j+k+l}\mathbf{tell}(c_i[note_i])
\end{array}
$$

The conductor process is always checking (listening) whether all the musicians play a note exactly at the same time

$$\bigwedge_{i\in[1,m]} (note_i \neq sil).$$

If this happens it sets cell $stop$ (initially set to 0) to 1. At the same time, it waits for all flags to be set to 1, and then resets the flags and gives the signal $go = 1$ indicating all musicians can start a new block, unless all of them have output a note at te same time (i.e. $stop = 1$).

$$
\begin{array}{l}
Conductor \stackrel{\text{def}}{=} \\
\quad \mathbf{wait}\, \bigwedge_{i\in[1,m]}(note_i \neq sil)\, \mathbf{do}\, stop := 1 \\
\quad ||\mathbf{when}\, \bigwedge_{i\in[1,m]}(flag_i = 1) \wedge (stop = 0) \\
\quad\quad \mathbf{do}\, (\mathbf{tell}(go = 1) \,||\, \Pi_{i\in[1,m]} flag_i := 0)
\end{array}
$$

Initially the m flag cells are set to 0, the M_i are given the start signal $start_i = 1$ and, as mentioned above, the cell $stop$ is set to 0. The system (i.e. the performance) is just a parallel execution of all the M_i musicians controlled by the $Conductor$ process.

$$
\begin{array}{l}
Init \stackrel{\text{def}}{=} \\
\quad \Pi_{i\in[1,m]}(flag_i : (0) \,||\, \mathbf{tell}\,(start_i = 1)) \,||\, stop : (0)
\end{array}
$$

$$
\begin{array}{l}
System \stackrel{\text{def}}{=} \\
\quad Init \,||\, Conductor \,||\, \Pi_{i\in[1,m]} M_i
\end{array}
$$

The temporal logic and the proof system of ntcc can then be used to formally specify and prove termination properties of this system. For example, we may wonder whether the assertion

$$System \vdash \diamond stop = 1$$

holds. This assertion expresses that the musicians eventually stop playing at all regardless their choices. We may also wonder whether there exists certain choices the musicians can make in order that they eventually stop playing definitely. For this, we can verify whether the assertion

$$System \vdash \Box stop = 0$$

does not hold, i.e., there is a run of the system for which at some time unit all the notes are different from sil.

4.2 modeling a harmonic problem

We model next a musical process described in [2]. It deals with harp music from Nzakara people of Central African republic. Two voices are constructed in such a way that the second one reproduces the first (up to transposition) with a time gap of p. The upper and lower voices play notes in the sets $\{64, 67, 70\}$ and $\{60, 62, 64\}$, respectively. A transposition function $f(64) = 60$, $f(67) = 62$, $f(70) = 64$ gives for each upper voice note the lower note that is to be played p time units later. Additional constraints state that time units that are either contiguous or separated by p units should not play the same chord. Finally, all chords thus formed must belong to the set $\{(60, 64), (60, 67), (62, 67), (62, 70), (64, 70)\}$.

The four $ntcc$ processes shown below model this problem.

$$
\begin{array}{l}
NOTES_{(midi,p)} \stackrel{\text{def}}{=} \\
\quad \sum_v (\, \mathbf{when}\, errors2 = v\, \mathbf{do} \\
\quad\quad \mathbf{unless}\, chord(PN_U, PN_L, CN_L, BN_L, midi) \\
\quad\quad\quad\quad \mathbf{next}\,(\mathbf{tell}\, errors2 = v + 1)\ \| \\
\quad\quad (\mathbf{when}\, chord(PN_U, PN_L, CN_L, BN_L, midi) \\
\quad\quad \mathbf{do} \\
\quad\quad\quad (\mathbf{tell}\,(CN_U = midi) \\
\quad\quad\quad \| \ \mathbf{next}\,(\,\mathbf{tell}\, PN_U = midi\,) \\
\quad\quad\quad \| \ \mathbf{next}^p\,(\,\mathbf{tell}\, CN_L = f(midi) \\
\quad\quad\quad\quad\quad\quad \| \ \mathbf{tell}\ BN_U = midi \\
\quad\quad\quad\quad\quad\quad \| \ \mathbf{next}\,(\,\mathbf{tell}\, PN_L = f(midi)\,)\,) \\
\quad\quad\quad \| \ \mathbf{next}^{2p}\,(\,\mathbf{tell}\, BN_L = f(midi)\,)\,) \\
\quad\quad\quad \| \ \mathbf{next}\,(\mathbf{tell}\, errors2 = v)\,)\,)
\end{array}
$$

$$
\begin{array}{l}
CHOOSE_{(p)} \stackrel{\text{def}}{=} \\
\quad NOTES_{(64,p)} \ + \ NOTES_{(67,p)} \\
\quad + NOTES_{(70,p)}
\end{array}
$$

$$
\begin{array}{l}
COUNTER \stackrel{\text{def}}{=} \\
\quad \sum_v (\, \mathbf{when}\, errors1 = v\, \mathbf{do} \\
\quad\quad (\mathbf{when}\, wrong(CN_L, CN_U)\, \mathbf{do} \\
\quad\quad\quad \mathbf{next}\,(\,\mathbf{tell}\, errors = v + 1\,) \\
\quad\quad \| \ (\mathbf{unless}\, wrong(CN_L, CN_U) \\
\quad\quad\quad\quad \mathbf{next}\,(\mathbf{tell}\, errors1 = v\,)
\end{array}
$$

$$
\begin{array}{l}
PROCESS_{(n,p)} \stackrel{\text{def}}{=} \\
\quad !_{[0,p-1]}\,(\mathbf{tell}\, CN_L = 0 \wedge BN_U = 0 \wedge PN_L = 0) \\
\quad \| \ \mathbf{tell}\, errors1 = 0 \ \| \ \mathbf{tell}\, errors2 = 0 \\
\quad \| \ \mathbf{next}^p\,(\,\mathbf{tell}\, PN_L = 0\,) \\
\quad \| \, !_{[0,2p-1]}\,(\mathbf{tell}\, BN_L = 0) \\
\quad \| \, !_{[0,n-1]}\,(\,CHOOSE_{(p)} \ \| \ COUNTER\,)
\end{array}
$$

Variables PN_X, CN_X and BN_X represent the previous note, the current note and the back note played p time units before, respectively. Index X is either U (upper voice) or L (lower voice).

Process $CHOOSE_{(p)}$ models the nondeterministic selection of a note in the upper voice. Process $NOTES_{(midi,p)}$ verifies the chord constraints (except membership to the given set) and then outputs the current upper note (CN_U). It also sends its current upper and lower notes as previous and back notes for the future. When the chord constraint does not hold, variable $errors2$ is incremented in the next time unit.

Process $COUNTER$ keeps track of the number of output chords not belonging to the given set. Summation index v ranges from 0 to n (the number of notes). Finally, $PROCESS_{(n,p)}$ states that there cannot be lower, previous or back notes before the time gap of p, so these are set to zero (meaning a silence). It also initializes the number of errors and launches the other process for all time units from zero to the number of notes.

4.3 proving properties of the Nzakara process

The model given above is weaker than required for the Nzakara musical problem. Instead of asserting chord membership constraints, errors are simply counted. Giving weaker $ntcc$ models allows proving negative properties of the real problem, such as the fact that there is no solution.

We give first the *strongest temporal formula* for the process:

$$\begin{array}{l} stf(PROCESS_{(n,p)}) = \\ (\Box_{p-1}\, CN_L = 0 \wedge BN_U = 0) \\ \wedge(\Box_p\, PN_L = 0) \wedge (\Box_{2p-1} BN_L = 0) \\ \wedge errors1 = 0 \wedge errors2 = 0 \\ \wedge\, (\Box_{n-1} stf(CHOOSE_p) \\ \qquad \wedge stf(COUNTER)) \end{array}$$

where strongest temporal formulae for the above processes $CHOOSE_p$ and $COUNTER$ are

$$\begin{array}{l} stf(COUNTER) = \\ (\bigvee_v errors1 = v \wedge (\neg wrong \wedge \circ(errors1 = v)) \\ \quad \vee(wrong \wedge \circ(errors1 = v+1))\,) \end{array}$$

$$\begin{array}{l} stf(CHOOSE_{(p)}) = \\ \qquad (\bigvee_{i\in\{64,67,70\}} stf(NOTES_{(i,p)})) \end{array}$$

$$\begin{array}{l} stf(NOTES_{(i,p)}) = \\ (\bigvee_{w\in\{0..n\}} errors2 = w \\ \qquad \wedge(chord_{midi} \vee \circ(errors2 = w+1)) \\ \qquad \wedge chord_{midi} \wedge CN_U \wedge \circ(PN_U = midi) \\ \qquad \wedge \circ^p(CN_L = f(midi) \wedge BN_U = midi) \\ \qquad \wedge \circ^{p+1}(PN_L = f(midi)) \\ \qquad \wedge \circ(BN_L = f(midi)) \wedge \circ(errors2 = w)\,) \end{array}$$

In the above definitions we write $\circ^k A$ for k nested occurrences of $\circ$ in $\circ(\circ(\ldots(\circ A))\ldots)$. Similarly, we write $\Box_k A$ for $A \wedge \circ A \wedge \circ(\circ A) \wedge \cdots \wedge \circ^k A$.

The strongest temporal formula of $PROCESS$ can be used s for proving various musical properties. It is straightforward to prove from $stf(NOTES)$ that the value of $errors2$ never decreases and also that a chord constraint violation implies a non zero vale of $errors2$ in subsequent time units. That is,

$$\begin{array}{l} \Box(stf(NOTES_{(n,p)}) \wedge errors2 = w \\ \qquad \Rightarrow \circ\,(errors2 \geq w)\,) \end{array}$$

$$\begin{array}{l} stf(NOTES_{(n,p)}) \wedge errors2 = w \wedge \neg chord \\ \quad \Rightarrow \circ\,(errors2 > w) \end{array}$$

Similarly, from the strongest formula of $COUNTER$ we can prove that $errors1$ never decreases and that a single violation of the chord set membership constraint causes a non zero value of $errors1$ there on.

The proof of the non solvability of the Nkazara process can be carried out by showing that each possible chord in the given set leads to a chord constraint violation:

Proposition 2 *Each chord in the given set violates a chord constraint. For all $k \in 0..n-p$ we have:*

$$\begin{array}{l} stf(PROCESS_{(n,p)}) \Rightarrow \\ \Box\circ^{p+k}(CN_L = 60 \wedge CN_U = 64 \\ \qquad \wedge errors1 = 0 \Rightarrow errors2 > 0\,) \end{array}$$

$$\begin{array}{l} stf(PROCESS_{(n,p)}) \Rightarrow \\ \Box\circ^{2p+k}(CN_L = 60 \wedge CN_U = 67 \\ \qquad \wedge errors1 = 0 \Rightarrow errors2 > 0\,) \end{array}$$

$$\begin{array}{l} stf(PROCESS_{(n,p)}) \Rightarrow \\ \Box\circ^{3p+k}(CN_L = 62 \wedge CN_U = 67 \\ \qquad \wedge errors1 = 0 \Rightarrow errors2 > 0\,) \end{array}$$

$$\begin{array}{l} stf(PROCESS_{(n,p)}) \Rightarrow \\ \Box\circ^{3p+k+1}\,(\\ \qquad (((CN_L = 62 \wedge CN_U = 70) \\ \qquad \vee(CN_L = 64 \wedge CN_U = 70)) \\ \qquad \wedge errors1 = 0) \Rightarrow errors2 > 0\,) \end{array}$$

Since there are no options for chords other than those ruled out by the previous proposition, we have

Corollary 1 *There is always a chord constraint violation after three periods (Nzakara time gaps), i.e.,*

$$\begin{array}{l} stf(PROCESS_{(n,p)}) \Rightarrow \\ \quad \Box\circ^{3p+k+1}(\, errors1 = 0 \Rightarrow errors2 > 0\,) \end{array}$$

from which it is easy to show

Corollary 2 *There is no solution for a Nzakara musical process having 30 notes and a time gap of 6, i.e.,*

$$\begin{array}{l} stf(PROCESS_{(30,6)}) \Rightarrow \\ \quad \Box\circ^{30}(\, errors1 > 0 \vee errors2 > 0\,) \end{array}$$

The expressive power of linear time logic and the weaker implementation of the Nzakara process in ntcc allow us to infer many other interesting musical properties, such as (see [2])

Proposition 3 *There is a Nzakara musical process having 30 notes and a time gap of 6 with fewer than 7 wrong chords, i.e.,*

$$\begin{array}{l} stf(PROCESS_{(30,6)}) \Rightarrow \\ \quad \Diamond\circ^{30}(\, errors2 = 0 \wedge errors1 < 7\,) \end{array}$$

5 Related and Future Work

We have described an ongoing project concerning modeling musical processes with ntcc. From the soft computing perspective, the fact that ntcc integrates constraint programming and non deterministic asynchronous behavior allows defining in a natural way musical processes for which only partial information about their temporal or harmonic properties is known. In particular, this includes partial information about the number of events of a process.

In the ntcc approach the construction and manipulation of musical structures rests on the firm ground of a precise, simple, yet powerful "time aware" computational model. This allows us to better understand interactions among concurrent musical processes and thus having better clues for the development of coherent music composition tools, particularly for real time settings.

A strong point of the described approach is that the linear temporal logic associated with ntcc can be used to prove (or disprove) interesting musical properties of processes before running them. We showed how this can be achieved for non trivial musical problems. The expressiveness of the logic allows us to state interesting musical properties in a very compact way. Moreover a significant fragment of ntcc, which include all the applications examples in this paper, can be compiled into finite state systems. Finite state systems are amenable to automatic verification (of a program satisfying a temporal logic formula), since their observable behavior can be finitely represented.

We plan to pursue this line of work in three directions: first, modeling in ntcc and proving properties of a variety of rhythm processes. Second, extending ntcc to a probabilistic model following ideas in [3]. This is justified by the existence of rhythm patterns examples involving stochastic rules which cannot be faithfully modeled with non-deterministic behavior. Third, our research group has implemented an abstract machine (called LMAN) which is capable of running ntcc code in real time. We have used LMAN to efficiently control a LEGO robot ([4]) using ntcc processes. We are currently adding a Midi interface to LMAN so as to get a more direct feedback about the musical models defined in ntcc.

References

1. G. Alvarez, J.F. Diaz, L.O. Quesada, C. Rueda, G. Tamura, F. Valencia, and G. Assayag. Integrating constraints and concurrent objects in musical applications: A calculus and its visual language. *Constraints*, January 2001.
2. M. Chemillier. Une esthetique perdue. In E. de Dampierre, editor, *La musique de la harpe*, pages 99–208, Paris, 1995. Presses de l'Ecole normale Superieur.
3. O. Herescu and C. Palamidessi. Probabilistic asynchronous pi-calculus. *FoSSaCS*, pages 146–160, 2000.
4. R. Hurtado, P. Munoz, C. Rueda, and F. valencia. Lman, an abstract machine of the ntcc calculus for concurrent programming of lego robots (in spanish). *Epiciclos*, 4, 2003.
5. R. Milner. *Communicating and Mobile Systems: the π-calculus*. Cambridge University Press, 1999.
6. F. Pachet. Rhythms as emerging structures. In *ICMC2000*, Berlin, Germany, 2000.
7. C. Palamidessi and F. Valencia. A temporal concurrent constraint programming calculus. In *Proc. of the Seventh International Conference on Principles and Practice of Constraint Programming*, 26 November 2001.
8. V. Saraswat, R. Jagadeesan, and V. Gupta. Foundations of timed concurrent constraint programming. In *Proc. of the Ninth Annual IEEE Symposium on Logic in Computer Science*, pages 71–80, 4–7 July 1994.
9. V. Saraswat, M. Rinard, and P. Panangaden. The semantic foundations of concurrent constraint programming. In *POPL '91. Proceedings of the eighteenth annual ACM symposium on Principles of programming languages*, pages 333–352, 21–23 January 1991.