Proving Musical Properties using a temporal Concurrent Constraint Calculus

Camilo Rueda, Frank Valencia
Universidad Javeriana-Cali and **BRICS**, University of Aarhus, Denmark
email:crueda@atlas.puj.edu.co, 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. We use the non-determinism 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 real musical problem.*

1 Introduction

Constraint programming and constraint satisfaction procedures have found an important place in music applications. Various languages and tools using these notions have been defined ((Assayag, Rueda, Laurson, Agon, and Delerue 1999), (Pachet and Roy 1995), (Laurson 1996)). These tools extend general purpose programming languages with new interfaces and search engines which are deemed suitable to compute musical structures defined by some given set of rules. Recently, more flexible search engines based on soft constraints have been proposed to be able to handle over-specified musical problems ((Truchet, Agon, and Codognet 2001)). The tendency is thus to provide tools that can offer approximate solutions when exact solutions do not exist or are extremely difficult to find.

Our work belongs to the constraint programming framework but our approach is entirely different. We envision computer supported music composition from a model of computation perspective. We thus look for a reduced set of simple objects and behavior that should be a minimal base to construct meaningful musical processes. The base concept we choose to bootstrap from is that of concurrent process. We regard music performance and composition as a complex task of defining and controlling interaction among concurrent activities. 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 (Alvarez, Diaz, Quesada, Rueda, Tamura, Valencia, and Assayag 2001), *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.

In this paper we propose using a temporal non deterministic concurrent calculus (*ntcc*, see (Palamidessi and Valencia 2001)) as a formal base to model

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

The ntcc calculus inherits ideas from the tcc model (Saraswat, Jagadeesan, and Gupta 1994), 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 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 allowing the specification of 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 (Rueda and Valencia 2001) we took advantage of the expressiveness of *ntcc* to program a music improvisation process. In this paper we are interested in the possibility of formally proving the properties of a musical process. 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 contribution of this paper is to show that by modeling a music process in ntcc one inherits a well defined logical inference system (see (Palamidessi and Valencia 2001)) that can be used to prove its musical properties (or lack thereof). We apply this approach to a real musical problem taken from (Chemillier 1995).

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.

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

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 ((Milner 1999)): $!\,P$ 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 the natural numbers, as an abbreviation for $\prod_{i\in I}\mathbf{next}^i P$ and $\sum_{i\in I}\mathbf{next}^i P$, respectively. For instance, $\star_{[m,n]}P$ means that 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.

2.0.1 Operational Semantics.

Operationally, the current 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 (Palamidessi and Valencia 2001) 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* $P \xrightarrow{(c,d)} Q$ 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:

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

and

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

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 (Palamidessi and Valencia 2001). In this section we summarize these results.

3.0.2 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 (Palamidessi and Valencia 2001). 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:*

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

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

From (Palamidessi and Valencia 2001) 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 (Palamidessi and Valencia 2001) gives extra mechanisms for carrying out proofs of process properties.

4 Musical examples

We introduce a broad idea for modeling music process in *ntcc* by means of a well known problem: Constructing a chromatic series containing all notes and all intervals. For simplicity we assume that notes are numbered $1..12$ and likewise for intervals, $0..11$, expressed in half tones. We define 12 ntcc processes each taking care of choosing one note (among those still available considering previous choices) and then informing the others about its choice. A note is chosen so that intervals are not repeated.

It will be convinient to specify our processes by using (possibly recursive) definitions. In ntcc it is possible to encode recursive definitions of the form $A \stackrel{\text{def}}{=} P_A$, where A is the process name and P_A. The intended behavior of a call A is that of P_A. A precise encoding of recursion in ntcc is given in (Palamidessi and Valencia 2001). In what follows we rely on the usual intuitions concerning procedure calls in a programming language.

$$
\begin{array}{l}
CHOOSE_{me} \stackrel{\text{def}}{=} \\
(\mathbf{unless}\, Token = me \\
\ \mathbf{next}\,(CHOOSE_{me}) \\
\ \| \ \mathbf{when}\, Token = me\, \mathbf{do} \\
\ \ \sum_j \mathbf{when}\,(Prev = j)\,\mathbf{do} \\
\ \ \ \ \sum_i \mathbf{unless}\,(Nchoice \neq i \\
\ \ \ \ \ \ \ \ \vee Ichoice \neq int(j,i)) \\
\ \ \ \ \ \ \mathbf{next}\,(!\,\mathbf{tell}\,(Nchoice \neq i) \\
\ \ \ \ \ \ \ \ \ \ \| !\,\mathbf{tell}\,(Ichoice \neq int(j,i)) \\
\ \ \ \ \ \ \ \ \ \ \| \ \mathbf{tell}\,(Token = me + 1) \\
\ \ \ \ \ \ \ \ \ \ \| \ \mathbf{tell}\,(Prev = i) \\
\ \ \ \ \ \ \ \ \ \ \| !\,\mathbf{tell}\,(Note_{me} = i) \\
\ \ \ \ \ \ \ \ \ \ ) \\
SERIES \stackrel{\text{def}}{=} \\
(\mathbf{tell}\,(Token = 1) \ \| \ \mathbf{tell}\,(Prev = 1) \\
\| \ CHOOSE_1 \ \| \ \ldots \ \| \ CHOOSE_{12}
\end{array}
$$

In process $CHOOSE_i$ summations range over note values. Each process $CHOOSE_i$ nondeterministically selects a certain note value unless it has already been chosen or the interval formed with the previous note has already appeared. Variable $Prev$ represents the value taken by the previous note in the sequence. Summation over j serves to find what that specific value was so as to use it for testing the constraint over the intervals. Function $int(j,i)$ computes the interval between notes i and j. Once a note successfully selected, the process states this fact forever and propagates forward new constraints to forbid further choices of this note (**tell** $Nchoice \neq i$) and also of the corresponding interval (**tell** $Ichoice \neq int(j,i)$). A token is passed so that all note processes are run sequentially.

The fact that this problem has a solution can be stated as follows:

$$\ulcorner SERIES \urcorner \vdash \Diamond \circ^{12} \bigvee_i (\, Note_{12} = i\,).$$

That is, eventually the twelve note in the twelve time unit has a concrete value. Since this can only happens when all the other notes have a concrete value, a solution is implied (we use $\circ^p$ as a shorthand for p nested occurrences of operator $\circ$. See below). One may also wonder whether solutions having certain interval patterns do exist. For instance, the logic could be used to prove that there are solutions with a third followed by a fifth in the middle of the series:

$$
\begin{array}{l}
\ulcorner SERIES \urcorner \vdash \\
\ \Diamond \circ^{12} (\bigvee_i (\, Note_{12} = i\,) \\
\qquad \wedge (|Note_6 - Note_5| = 3) \\
\qquad \wedge (|Note_7 - Note_6| = 7))
\end{array}
$$

4.1 modeling a harmonic problem

We model next a musical process described in (Chemillier 1995). 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.

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

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

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

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

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 $COUNTER$ counts 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.

4.2 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:

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

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

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

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

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

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 $A \wedge \circ A \wedge \circ(\circ A) \wedge \cdots \wedge \circ^k A$.

We can use the strongest formula of $PROCESS$ 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,

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

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

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:*

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

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

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

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

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.,*

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

from which it is easy to show

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

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

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 (Chemillier 1995))

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.,*

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

5 Related and Future Work

We have described an ongoing project concerning modeling musical processes with ntcc. The main advantages of this approach is that 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. It is of course true that complex musical processes, particularly those involving several musical dimensions, could be extremely difficult to model in such a "low level" mechanism. Our aim is not promote ntcc as a computer music language, but as a sort of "runnable specification" formalism of a variety of musical processes.

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 recent results show that 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, in particular those aiming at constructing material obeying well defined patterns, such as described in (Laurson and Kuuskankare 2001) or the metric modulations in (Nicolas 1990). Initial work in this area

has given us encouraging results. Second, extending ntcc to a probabilistic model following ideas in (Herescu and Palamidessi 2000). This is justified by the existence of rhythm patterns examples involving stochastic rules which cannot be faithfully modeled with non-deterministic behavior. Third, we have begun the implementation of an abstract machine for ntcc and plan to construct a music composition language on top of it.

References

Alvarez, G., J. Diaz, L. Quesada, C. Rueda, G. Tamura, F. Valencia, and G. Assayag (2001, January). Integrating constraints and concurrent objects in musical applications: A calculus and its visual language. *Constraints*.

Assayag, G., C. Rueda, M. Laurson, C. Agon, and O. Delerue (1999). Computer-assisted composition at ircam: From patchwork to openmusic. *Computer music journal 23*(3), 59–72.

Chemillier, M. (1995). Une esthetique perdue. In E. de Dampierre (Ed.), *La musique de la harpe*, Paris, pp. 99–208. Presses de l'Ecole normale Superieur.

Herescu, O. and C. Palamidessi (2000). Probabilistic asynchronous pi-calculus. *FoSSaCS*, 146–160.

Laurson, M. (1996). Pwconstraints reference manual. Available through IRCAM user's group.

Laurson, M. and M. Kuuskankare (2001). A constraint based approach to musical textures and instrumental writing. In *Proceedings of Workshop on musical constraints, CP2001*, Paphos, Cyprus, pp. 44–51.

Milner, R. (1999). *Communicating and Mobile Systems: the π-calculus*. Cambridge University Press.

Nicolas, F. (1990). Le feuilleté du temps, essaie sur les modulations métriques. *Entretemps* (9).

Nielsen, M. and F. Valencia (2001, February). *Temporal Concurrent Constraint Programming: Applications and Behavior*, Chapter 4, pp. 298–324. Springer-Verlag, LNCS 2300.

Pachet, F. and P. Roy (1995). Mixing constraints and objects: A case study in automatic harmonization. In *Proceedings of TOOLS Europe'95*, Versailles, France, pp. 119–126. Prentice-Hall.

Palamidessi, C. and F. Valencia (2001, 26 November). A temporal concurrent constraint programming calculus. In *Proc. of the Seventh International Conference on Principles and Practice of Constraint Programming*. Springer-Verlag, LNCS 2239.

Rueda, C. and F. Valencia (2001). Formalyzing timed musical processes with a temporal concurrent constraint calculus. In *Proceedings of Workshop on musical constraints, CP2001*, Paphos, Cyprus, pp. 44–51.

Saraswat, V., R. Jagadeesan, and V. Gupta (1994, 4–7 July). Foundations of timed concurrent constraint programming. In *Proc. of the Ninth Annual IEEE Symposium on Logic in Computer Science*, pp. 71–80.

Truchet, C., C. Agon, and P. Codognet (2001). A constraint programming system for music composition, preliminary results. In *Proceedings of Workshop on musical constraints, CP2001*, Paphos, Cyprus, pp. 34–43.