# Operational semantics of a domain specific language for real time musician-computer interaction

José Echeveste, Arshia Cont, Jean-Louis Giavitto, Florent Jacquemard

Discrete Event Dyn Syst (2013) 23:343–383
DOI 10.1007/s10626-013-0166-2

# Operational semantics of a domain specific language for real time musician–computer interaction

**José Echeveste · Arshia Cont · Jean-Louis Giavitto · Florent Jacquemard**

Received: 6 April 2012 / Accepted: 11 April 2013 / Published online: 26 April 2013


**Abstract** With the advent and availability of powerful personal computing, the computer music research and industry have been focusing on real-time musical interactions between musicians and computers; delegating human-like actions to computers who interact with a musical environment. One common use-case of this kind is *Automatic Accompaniment* where the system is comprised of a real-time machine listening system that in reaction to recognition of events in a score from a human performer, launches necessary actions for the accompaniment section. While the real-time detection of score events out of live musicians' performance has been widely addressed in the literature, score accompaniment (or the reactive part of the process) has been rarely discussed. This paper deals with this missing component in the literature from a formal language perspective. We show how language considerations would enable better authoring of time and interaction during programming/composing and how it addresses critical aspects of a musical performance (such as errors) in real-time. We sketch the real-time features required by automatic musical accompaniment seen as a reactive system. We formalize the timing strategies for musical events taking into account the various temporal scales used in music. Various strategies for the handling of synchronization constraints and the handling of errors are presented. We give a formal semantics to model the possible behaviors of the system in terms of *Parametric Timed Automata*.

**Keywords** Computer music · Score following · Automatic musical accompaniment · Synchronous programming · Synchronous language · Error handling

This work has been partly supported by the ANR project INEDIT (ANR-12-CORD-0009).

J. Echeveste (✉) · A. Cont · J.-L. Giavitto · F. Jacquemard
STMS Lab (IRCAM, CNRS, UPMC, INRIA)—RepMus/Mutant team,
1, place Igor-Stravinsky, 75004 Paris, France
e-mail: Jose.Echeveste@ircam.fr

## 1 Introduction

The need to specify music by a scoring system was manifested in most civilizations who knew writing. Western music has long since developed a formalism to specify fine temporal relationships involving timing, duration and sequencing between musical objects such as notes or chords. If this formalism is relatively stable for classical music, new and difficult problems arise in the context of *mixed music*. Here, this latter term denotes the association in live performance of human musicians and computer mediums, interacting in *real-time*, and the authoring of such interaction for both mediums (music composition).

A simple example of such interaction is *automatic accompaniment* for an existing piece of music for a solo musician and ensemble, where the solo musician is played by a human performer whereas the ensemble part is automatically accompanied by computer generated sounds. An automatic accompaniment system relies on an automatic real-time *score following* system that in reaction to the recognition of events in a score from a human performer, triggers the necessary computer actions for the accompaniment section. For contemporary electronic art repertoire, the accompaniment actions take the form of (real-time) programs whose computational nature vary from signal processing (sound transformations), to control (sound spatialization), transactional processing (concatenative synthesis on sound databases), and more.

While the real-time detection of score events out of live musicians' performance has been widely addressed in the literature, the reactive part of the score accompaniment has been poorly investigated. Indeed, the novelty introduced by mixed music w.r.t. the usual score, is that the performative dimension of the computer accompaniment must be explicitly introduced in the score: the "musical synchronization" between the part performed by the computer and the human performance must be specified in some way in a language processable by a computer as well as the nature of the computation for each action and the handling of errors.

*Contributions* In this paper, we give a semantics of an augmented score corresponding to a classical score extended with a *synchronous reactive language*. This language is dedicated to the specification of the relationships between the events performed by humans and the score events controlled by the computer.

This domain specific language has been developed as the *coordination language* of the *Antescofo* score following system.[1] The genericity and the expressiveness provided by the augmented score language make *Antescofo* a unique framework for the authoring of mixed music, supporting *musical time*, *synchronization strategies* and *error handling* both during composition (programming) and live performance. An objective of this paper is to introduce the motivations and the rationals behind these features in a manner accessible to a computer scientist without backgrounds in music. For the sake of the presentation, we focus on a core language which is paradigmatic of the constructions available in the full coordination language of *Antescofo*. This core language illustrate the major issues in the formalization of musical relationships: the management of musical phrases, tempi, relative duration and the handling of errors.

[1] http://repmus.ircam.fr/antescofo.

Defining the semantics of a score following system faces original problems. One of them is the development of a framework able to encompass the temporal relationships specified by a composer, their actual instantiations by musicians and their relationships with the accompaniment actions. In this paper, the formal semantics of the core language is defined in terms of *Parametric Timed Automata* (Alur et al. 1993). The same automaton is used to give a semantics of an ideal performance (as written in the score) and of an actual performance (the musical interpretation of the score), just by changing the values of parameters.

Providing a semantics to the augmented score language is a first step in the formal modeling of the temporal relationships involved in mixed pieces. Our long term goal is to use this semantics as a firm basis to design more sophisticated synchronization mechanisms with human in the loop, to prove the correctness of some score transformations, to establish certain invariants (e.g., the preservation of some temporal constraints despite the variation of the human interpretation), the development of a relevant notion of observation and equivalence (e.g., to test such system), etc. In this perspective, we establish here a first simple result stating the determinism (in some sense) of the accompaniment for any interpretation.

*Related works* *Antescofo* is unique in the sense that it is an attempt to couple two literatures of real-time recognition and reactive systems, usually considered distinctively but whose integration make complete sense in a musical practice. To this respect, similar systems address either of the two issues. Among such, one can note the *Music-Plus-One* system by Raphael (2011) which undertakes automatic accompaniment for classical music repertoire. This system is similar to *Antescofo* in its machine listening features. However it does not allow musicians to arrange and to compose the accompaniment scores; moreover synchronicity is a result of the machine listening and does not address fault tolerance. A similar work to the reactive component of *Antescofo* is the *Iscore* project by Desainte-Catherine and Allombert (2005). This computer-assisted composition tool allows composers to build musical parts and structures their pieces, binding them with temporal logical relations and adding discrete interactive events to control the triggering of some actions. The temporal constraints rely on Allen (1983) relations but are far from customary musical vocabularies addressed in this paper. Furthermore, *Iscore* and similar approaches do not address the problem of dynamic binding and interactivity with outside environment.

The augmented score developed in *Antescofo* can be seen as a domain specific synchronous reactive language in which the accompaniment actions of a mixed score are specified. As a matter of fact, an analogy can be made between interactive music composition using *Antescofo* and synchronous programming. In a score, a composer indeed predicts and specifies the temporal evolution of processes and the events of the musical performance (inputs) on which they depend. In the same way that an engineer would use a synchronous language to specify commands to be sent to the actuators, depending on data acquired through sensors, the composer describes the musical responses of the system (the accompaniment actions) relative to its environment (the recognized events). And similar to reactive systems, an augmented score assumes the *strong synchrony hypothesis*: actions hooked directly to an event should occur in zero-time and in the right order. This hypothesis is unrealistic, however it is an excellent abstraction of musical behavior. In practice, the system

needs to be quick enough to preserve the auditory perception of simultaneity, which is on the order of 20 ms.

As a synchronous reactive language, the augmented score departs in several ways from common reactive languages such as Lucid Synchrone (Pouzet 2006), ReactiveML (Mandel and Plateau 2008) or Esterel (Berry and Gonthier 1992); yet it shares the notion of strong synchronization, instantaneous action and the declarative style of Lustre (Halbwachs et al. 1991), Signal (LeGuernic et al. 1991) and Lucid Synchrone. The specification of the score raises new issues and requires special extensional notations. The handling of multiple related tempi and the handling of errors are also specific to the musical application domain.

To the best of our knowledge, this work and (Echeveste et al. 2011) are the first attempts to give a formal semantics to the reactive part of an automatic accompaniment system: research in automatic accompaniment have focused instead on the first stage of music score alignment (Dannenberg and Raphael 2006). The parametric timed automaton approach developed in this work answers some problems raised by the denotational semantics in terms of timed traces given in Echeveste et al. (2011). It preserves some of the high-level structures of the augmented score. The structural relationships between the score and the actual interpretations are made explicit, relying on one common parametric automaton. This well definite object is associated to the family of possible interpretations as well as a given interpretation. In this setting, the specification of error handling strategies is more straightforward and relies on alternative paths in the automaton, instead of implicit computation in the traces. This framework opens the way to the analysis and design of alternative synchronization strategies. Finally, the timed automaton framework is more widespread and amenable to automated verifications using methods and tools based on this model.

*Organization of this paper* This paper is structured in two parts. In the first part, we introduce the necessary musical backgrounds to understand the application domain and the requirement of an augmented score language (Section 2). Specifically, requirements for *musical synchronization* and the rational of error handling are discussed. Section 3 introduces the *Antescofo* automatic accompaniment systems and discusses the relevance of the synchronous framework for music accompaniment. For the convenience of the reader, the last section of this first part recalls some basic information on the timed automata framework.

In the second part, we present a core language for augmented score and its semantics. This language provides constructs like continuous processes, loops and parallel (polyphonic) phrasing. The specific syntax is exposed in Section 5. The Section 5.6 explicitly deals with synchronization strategies and error handling of this reactive paradigm in the context of a music performance. Section 6 introduces syntactic transformations used to translate the core language into a normalized form making the semantics more easy to express on a smaller equivalent language. This translation is defined inductively on the score grammar. Section 7 provides the semantics of the core language in term of *Parametric Timed Automata*. A simple result on the determinism of an interpretation is exposed. An example is developed throughout the second part to illustrate the constructions introduced. The paper ends with a conclusion and a discussion about future works.

## Part I: Background on real-time and musician–computer interaction

### 2 Musical backgrounds

The usual workflow of musical creation in a context of written music is constituted by two main phases: *Composition* and *Performance*. The role of the composer is to describe the temporal development of each instrument, ensuring correct superposition of different voices (parts) in time. Actions are described using a set of high-level structures issued from the western musical tradition, like notes, rhythms and nuances which allow the composer to author a musical score. During a live performance, musicians (four in case of Fig. 1) are expected to perform each voice in parallel and produce a coherent output despite variations common in musical interpretations.

*Composition* Figure 1 shows an excerpt of a typical music score, a string quartet. This example shows both the autonomy of the different voices (specific phrases, specific nuances...) and their strong temporal relationships (synchronization, simultaneity/polyphony). It is especially interesting to note the characteristics of the "musical time" in this compositional phase: the timing relationships (delay, duration, sequencing, simultaneity…) between events denoted in the score are *relative* (to each others), *virtual* (the timing relationships expressed at the level of the score will be instantiated during the performance) and *undetermined* (several performances comply with the same score).

*Performance* During live performance of a music score, musicians "instantiate" the high-level processes denoted in the score by musical gestures. At this point, the durations and delays become physical time (measurable in second). However, events with the same relative duration in a score (and in different positions) do not necessarily lead to the same duration during the executation and vice versa. Their value depends highly on the performance, individual performers and musical

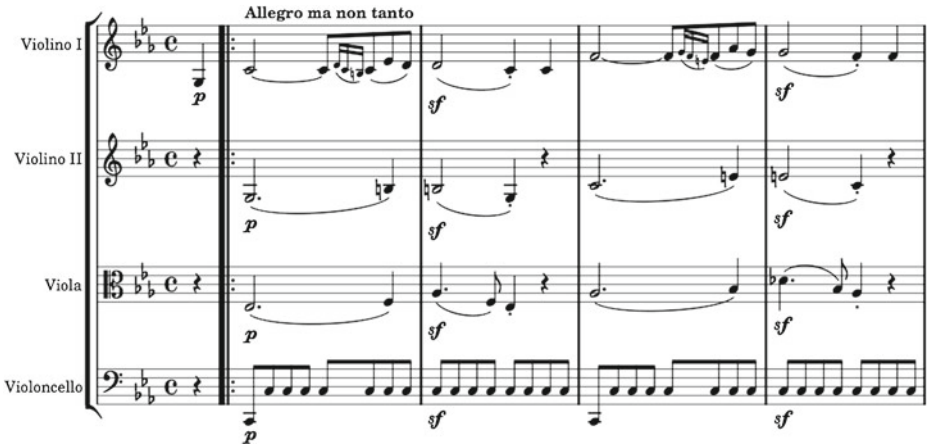



**Fig. 1** Score example for a string quartet, by Ludwig Van Beethoven, each line corresponds to a specific instrument

interpretation strategies such as stylistic features that are neither determined nor easily formalizable.

Despite the indeterminacy due to musical interpretation, the musicians can produce together a result which is deterministic in many respect: the order of the events in the score is preserved as well as the relative duration of events.[2]

We stress that various synchronization strategies are needed following the type of interactions along the performance: sometimes one instrument takes the lead and plays a melody while the others accompany and synchronize on it, or instruments group together to play in a "call and response" manner, each with its own pace but interleaving in a fluid way. To do so, musicians adopt various strategies such as synchronization points, head movement or breathing, but above all they constantly anticipate the events' happening with a shared knowledge of the *tempo* and overall structure of the music score.

*Tempo* The tempo is a specific clock which defines to the execution speed or pace of a musical piece. Music event duration is commonly described in fractions of a "beat", a measurement unit relative to the clock (tempo) and defining musical rhythms. The tempo (or speed) of a score is commonly expressed as "beats-per-minute". Therefore, a fast tempo determines closer beats while a slow tempo determines longer beats. If explicitly expressed, the tempo of a piece is just an indication and its evolution in time is left at the discretion of the performer(s). For example, the score of Fig. 1 describes the general tempo of the piece as *Allegro ma non tanto* which literally refers to *fast, quickly and bright* (Allegro) but *not so much* (non tanto). As the musicians are aware of the tempo's value, they are able to foresee the future, for synchronize the end of phrases together or to attack synchronously a new beginning.

*Augmented scores for automatic accompaniment* In the context of automatic accompaniment, a composer would want to add additional voices to the score corresponding to computer actions which will be performed by a computer. These actions may control a virtual instrument as well as all devices present on stage (light management, mixing, etc.). At the composition phase, the composer needs a language to specify the temporal development of electronic voices relative to the instrumental voices. In particular, she must be able to specify the fine temporal relationships that structure the interactions between the computer and the human performers. The score of the instrumental voices together with the score of the electronic voices and the specification of the temporal relationships between these two kinds of voice, constitute an *augmented score*. An example of augmented score is pictured in Fig. 2.

During the performance, the electronic part must be played musically, consistently and synchronously with the other voices.

*Errors in performance* Furthermore the musicians may make local errors that should neither halt the process nor interfere with the global coherence of the performance. In a live performance situation, different errors may be encountered: the listening module could confuse an event with another, miss an event, or produce

[2] At least locally, that is, in a bounded time window and for a specific voice.

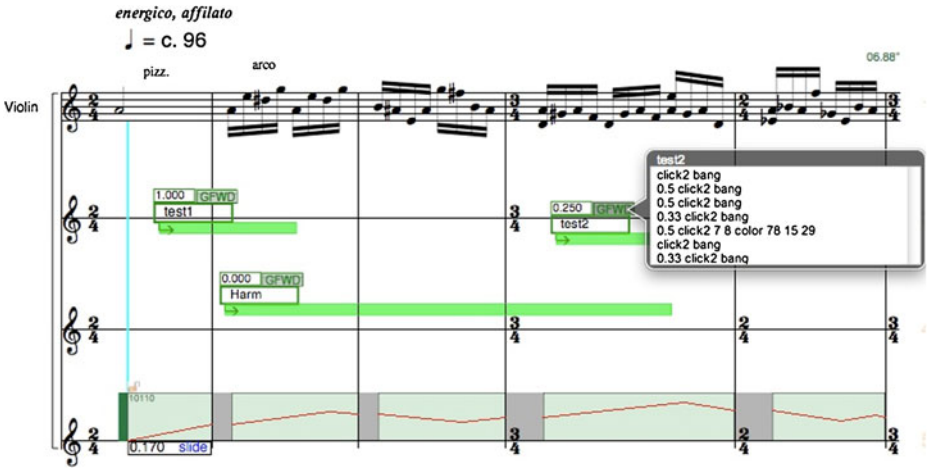



**Fig. 2** Augmented score for an interactive piece; the *top* staff contains the *instrumental* section whereas lower staves correspond to specific computer music commands. This score features three discrete-sequence groups and a continuous trajectory (*bottom* staff) as accompaniment actions of one instrument voice (*upper* staff), all living within a single score framework. Section 5 describes the constructions available in the *Antescofo* augmented score language. The authoring of *Antescofo* augmented scores directly on a graphical representation such as this one is possible thanks to the integration within the *NoteAbility* score editor

a false-alarm. Additionally, musicians might introduce performance errors that can affect the accompaniment results. In all cases, we expect not only that the system continues to work, but also that it reacts as musically as possible. A similar requirement holds for critical embedded systems such as those on airplanes where in-flight errors can result to extreme human casualties. Even if human lives are not at stake in an interactive computer music performance, a crash or halt during a concert should be avoided!

## 3 Score following and the Antescofo system

The *Antescofo* system is an attempt to address both the compositional (programming) and the performative (real-time evaluation and synchronization) dimensions in mixed music as described in Section 2. *Antescofo* achieves this goal relying on two subsystems (see Fig. 3): a *listening machine* that tracks in real-time the position of the musicians in the augmented score and a *reactive engine* relying on a *dedicated synchronous language* for the real-time coordination of the electronic part. This paper focuses on the domain specific language (DSL) of *Antescofo*. This DSL is a simple language which permits to specify augmented scores, containing both instrumental and electronic parts and the instructions for their coordination during a performance (i.e. the specification of the events to recognize in real-time and of the actions triggered by these), following the ideas presented in Section 2. This section gives a bird's-eye view on *Antescofo*. The augmented score language, its syntax and semantics, are studied in the second part of the paper.

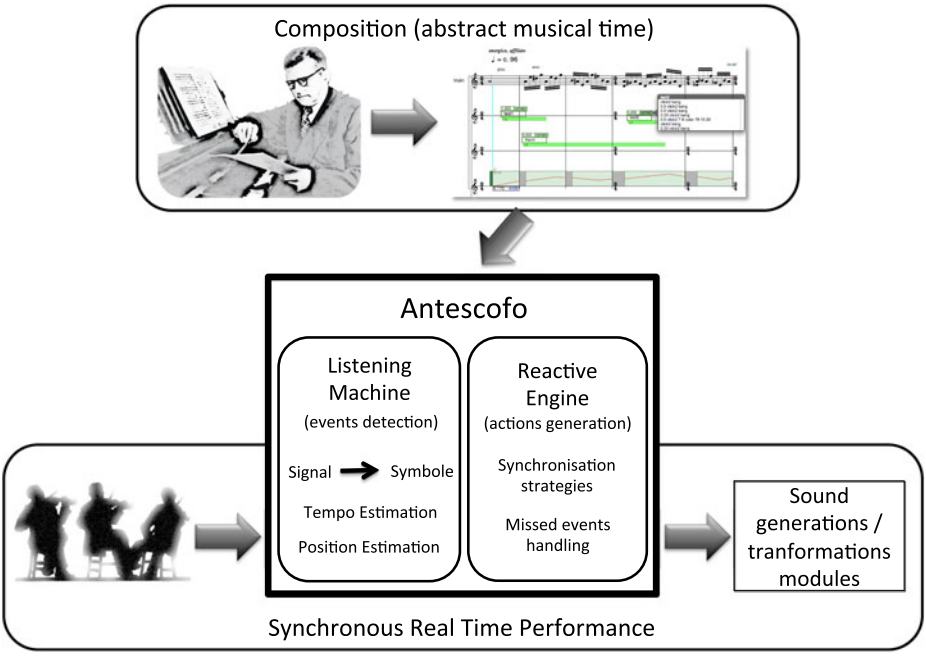



**Fig. 3** General architecture of the *Antescofo* system. The augmented score is an input of both the listening machine and the reactive engine. The events recognized by the listening machine are signaled to the reactive engine which schedule in turn the corresponding actions at the right time

*Score following* Traditionally score following is defined as the real-time automatic alignment of an audio stream played by one or more musicians, into a symbolic musical score. It was first invented in the 80s by Dannenberg (1984) and Vercoe (1984) employing symbolic inputs and extended a decade later to audio inputs, following advances in real-time signal processing. Late 90s saw the integration of probabilistic methods for score following improving the robustness of such systems. *Antescofo* (Cont 2008) extends the score following paradigm by integrating in the same environment a polyphonic listening machine (Cont 2010) with the management of the accompaniment actions, see Fig. 3. The role of *Antescofo* in real-time is to decode the position and tempo of the performer, and use that information within a synchronous reactive system to best interpret the accompaniment voice.

One of the most important and unique features of *Antescofo* is its dynamic time management. The listening machine constantly decodes the tempo and position of the live performer and feed them to the reactive system for synchronous performance. This is achieved by explicit time models inspired by cognitive models of musical synchrony in the brain (Large and Jones 1999) which provide both the tempo of the musician in real-time and also the *anticipated* position of future events used for real-time scheduling. To this respect, *Antescofo* imitates human musicians who constantly base their actions on current events and also future belief of incoming events. In this paper, we focus on the reactive system. Readers curious on the implementation details of the listening machine can refer to Cont (2010).

*Antescofo as a reactive system* During live performance, *Antescofo* is thus a *reactive system* reacting to data from the listening machine. Similar to classical reactive systems, the result is expected to be synchronous and deterministic (as described in a music score) despite real-time variations in the environment (human musicians' interpretations). Environment events and computer actions in a score share similar time scales (either relative or physical) which are linearly ordered.

The reactive system dynamically considers the tempo fluctuations for the interpretation of accompaniment actions. The possibility of dating the events and the actions relatively to the tempo, as in a classical score, is one of the main strengths of *Antescofo*. Within the augmented score language, the composer can thus decide to associate actions to certain events with delays (physical time or relative to the tempo), to group actions together, to structure the groups hierarchically and to have such groupings act in parallel as shown in Fig. 2. Several phrasing schemes for groups are available in *Antescofo* and allow the scope of such electronic phrases to go beyond inter-onsets of the instrumental score.

## 4 Networks of parametric timed automata

For the reader's convenience, we recall in this section some basic information on the timed automata framework, we follow (Alur and Dill 1994; Alur et al. 1993). Timed automata are well adapted to the expression of the timing constraints appearing in augmented scores because they are a powerful model for describing both the logical ordering of the events in such score and also the durations of events and the timing between these.

A timed automaton is a finite automaton with a finite set of real-valued variables called *clocks*. Each clock can be independently reset to 0 on a transition and keeps track of the elapsed time since the last reset. A transition is guarded by a constraint on the clock values: the transition may be taken only if the current values of the clocks satisfy the associated constraint. A timed automaton accepts *timed words*: sequences of symbols associated to their time of occurrence (dates are expressed as real values).

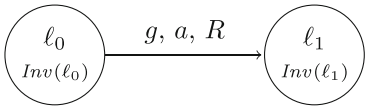
A *Parametric Timed Automaton* is defined as a tuple

$$A = (L, I, F, X, \Sigma, T, Inv)$$

where $L$ is a finite set of locations, $I \subseteq L$ is the set of initial locations, $F \subseteq L$ is the set of final (accepting) locations, $X$ is a finite set of clocks, $\Sigma$ is a finite alphabet of symbols, $T \subseteq L \times \mathcal{C}(X) \times \Sigma \times 2^X \times L$ is a finite set of transitions, and $Inv : L \to \mathcal{C}(X)$ assigns an invariant to each location, where $\mathcal{C}(X)$ is the set of *clock constraints*. Here we restrict $\mathcal{C}(X)$ to the conjunctions of atomic constraints of the form $x \bowtie d$ where $\bowtie \in \{\leq, <, =, >, \geq\}$, $x$ is a clock and $d$ is a constant in $\mathbb{Q}_+$ or a parameter. We use parameters to represent the delays between events; they will be instantiated by real values depending on the performance.

In words, a transition $(\ell_0, g, a, R, \ell_1)$ from the location $\ell_0$ to the location $\ell_1$ is guarded by a clock constraint $g$ and specifies an observed symbol $a$ and the set of

clocks $R \subseteq X$ to reset when the transition is taken. Such transition will be denoted $\ell_0 \xrightarrow{g,a,R} \ell_1$ and can be pictured graphically as follow:




We will use in the following the graphical syntax to define the automaton associated to an augmented score in Section 8.

*Example 1* We will use the same running example throughout the paper. This example is presented in Fig. 6 below.

Let us present the automaton which formalizes the series of events produced by the performer (i.e. the instrumental part). This instrumental part reduces to only two events (notes) $e_0$ (with an ideal duration specified in the score of 1.0 beat) and $e_1$ (with a duration of 1.0 beat too). The corresponding automaton starts in location start and awaits before either the launch of $e_0$ or signaling that $e_0$ is missing which is denoted $\overline{e_0}$. The crossing of these transitions does not require additional conditions on the clocks. After the occurrence of $e_0$, the clock $x$ is reset to 0. The automaton will stay in location prep $e_1$ during the duration of the event $e_0$, this location is therefore constrained by the invariant $x \leq \gamma_0$. Variable $\gamma_0$ is a parameter: it will be instantiated to 1.0 to obtain the automaton specifying an ideal performance and by the actual duration of the event $e_0$ during an actual performance to obtain the description of this performance. At the end of this delay $\gamma_0$, the automaton proceed with the handling of the event $e_2$.


*Timed languages* A *run* of a timed automaton without parameters $A = (L, I, F, X, \Sigma, T, Inv)$ is a finite sequence of the form

$$\langle \ell_0, v_0 \rangle \xrightarrow{d_1,a_1} \langle \ell_1, v_1 \rangle \xrightarrow{d_2,a_2} \ldots \xrightarrow{d_k,a_k} \langle \ell_k, v_k \rangle$$

where every $\ell_i$ is a location in $L$, every $v_i : X \to \mathbb{Q}_+$ is a clock valuation, every $d_i$ is a delay in $\mathbb{Q}_+$ and every $a_i \in \Sigma$; and such that $v_0$ satisfies $Inv(\ell_0)$, and for all $1 \leq i \leq k$, for all $0 \leq d' \leq d_i$, the clock valuation $x \mapsto v_{i-1}(x) + d'$ satisfies $Inv(\ell_{i-1})$, and there exists a transition $\ell_{i-1} \xrightarrow{g_i,a_i,R_i} \ell_i$ in $T$ such that:

- $v_{i-1} + d_i$ satisfies $g_i$,
- $v_i = reset_{R_i} \circ v_{i-1}$, where $reset_{R_i}(x) = 0$ if $x \in R_i$ and $reset_{R_i}(x) = x$ otherwise,
- $v_i$ satisfies $Inv(\ell_i)$.

Below, a pair $\langle \ell_i, v_i \rangle$ as above is called a *state* of $A$.

The run is *accepting* if $\ell_0 \in I$, $\ell_k \in F$, and $v_0(x) = 0$ for all $x \in X$. In this case, we say that the timed word $\langle a_1, t_1\rangle, \ldots, \langle a_k, t_k\rangle \in (\Sigma \times \mathbb{Q}_+)^*$ is accepted, where $t_i = \sum_{j=1}^{i} d_i$ for all $1 \leq i \leq k$. The timed language of $A$ is the set of accepted timed words.

*Example 2* An example of run accepted by the automaton in Example 1 is

$$\langle \mathsf{start}, x \to 0)\rangle \xrightarrow{d_0, e_0} \langle \mathsf{prep}\, e_1, x \to 0)\rangle \xrightarrow{\gamma_0, e_1} \langle \mathsf{stop}, x \to \gamma_0\rangle$$

for some initial delay $d_0$. It is easy to see that the accepted timed language is

$$\{ \quad \langle e_0, d_0\rangle\langle e_1, d_0 + \gamma_0\rangle,\ \langle \overline{e_0}, d_0\rangle\langle e_1, d_0 + \gamma_0\rangle,$$
$$\langle e_0, d_0\rangle\rangle\overline{e_1}, d_0 + \gamma_0\rangle,\ \langle \overline{e_0}, d_0\rangle\langle \overline{e_1}, d_0 + \gamma_0\rangle \quad \}$$

for all $d_0 \leq 0$.

*Networks and synchronized product* A *network* is a set of parametric timed automata, denoted with the operator $\|$ of parallel composition. The automata of the network are supposed to have disjoint sets of clocks and locations, but they may share alphabet's symbols which are used for synchronization. It is possible to define a *synchronized product* of all the timed automata composing a network. The result of this operation is one timed automaton representing the behavior of all the components of the network, functioning and synchronizing in parallel. Let $A_1 \| \cdots \| A_m$ be a network of parametric timed automata, with $A_i = (L_i, I_i, F_i, X_i, \Sigma_i, T_i, Inv_i)$ $(1 \leq i \leq m)$ such that all the $L_i$ and $X_i$ are disjoint. The synchronized product $sync(A_1 \| \cdots \| A_m)$ is the timed automaton $A = (L, I, F, X, \Sigma, T, Inv)$ where

- $\Sigma = \bigcup_{i=1}^{m} \Sigma_i$
- $L = L_1 \times \cdots \times L_m$, $I = I_1 \times \cdots \times I_m$, $F = F_1 \times \cdots \times F_m$,
- for every $\ell = \langle \ell_1, \ldots, \ell_m\rangle \in L$, $Inv(\ell) = \bigwedge_{i=1}^{m} Inv_i(\ell_i)$,
- $X = \bigcup_{i=1}^{m} X_i$,
- $\Sigma = \bigcup_{i=1}^{m} \Sigma_i$
- $T$ is the set of all transitions $\langle \ell_1, \ldots, \ell_m\rangle \xrightarrow{g, a, R} \langle \ell'_1, \ldots, \ell'_m\rangle$ such that (where for $a \in \Sigma$, $S_a = \{i \mid 1 \leq i \leq m, a \in \Sigma_i\}$)
  - for all $1 \leq i \leq m$, if $i \notin S_a$, then $\ell'_i = \ell_i$,
    if $i \in S_a$, then there exists $g_i$ and $R_i$ such that $\ell_i \xrightarrow{g_i, a, R_i} \ell'_i \in T_i$,
  - $g = \bigwedge_{i \in S_a} g_i$,
  - $R = \bigcup_{i \in S_a} R_i$.

The $A_i$ are referred below as the *subautomata* of $A$.

*Example 3* This example illustrates the *synchronized product*.

We first introduce another automaton, simpler than those that we define in Section 7.1, which represent electronic actions generated by *Antescofo*. In the starting location "wait $e_1$", the automaton awaits for events $e_1$ or $\overline{e_1}$ before accessing respectively locations "prep $a$" or "stop $a$". When one of the two transitions is taken,

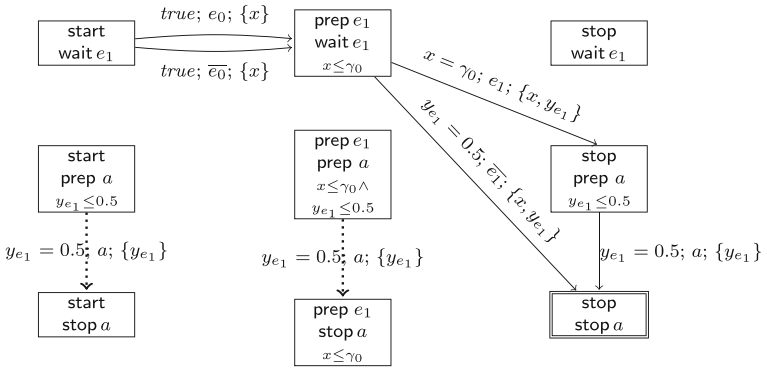
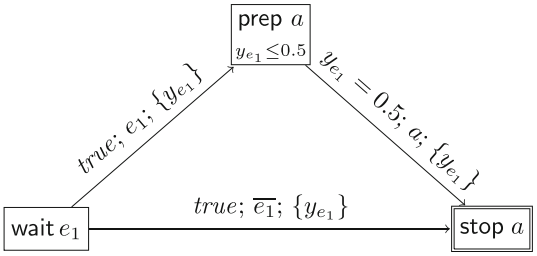



**Fig. 4** Synchronized product of Example 3

the clock $y_{e_1}$ is reset to 0. In the location "prep $a$", the transition with symbol $a$ is taken when the clock reaches 0.5.




The automaton in Fig. 4 corresponds to the synchronized product of the automaton presented in Example 1 and the above automaton. The synchronized product has nine locations. Some of them are not interesting because not reachable from the initial location (start, wait $e_1$). This is the case for instance of (start, prep $a$) and (prep $e_1$, prep $a$). The transitions from these locations are depicted with dotted arrows. Note the synchronization made in the transitions labeled with $e_1$ and $\overline{e_1}$.

## Part II: semantics for Antescofo augmented scores

In this second part we present the syntax and semantics of the core of the *Antescofo* augmented score language, specifying the management of musical phrases, tempi, relative duration, the triggering of actions and the handling of errors.

The syntax of the *Antescofo* augmented scores is presented in Section 5. Then we proceed as follows (Fig. 5): augmented scores are first normalized (Section 6). The semantics of a normalized score is then given as a network of parametric timed automata (Sections 7.1, 7.2, 7.3). The semantics of a performance (either ideal or actual) is obtained by giving actual values to the parameters of the automata (Section 7.4). In Section 7.5, we rely on the semantics just presented to characterize the determinism of *Antescofo* : a fixed instrumental performance will always induce the same accompaniment reaction of the system.

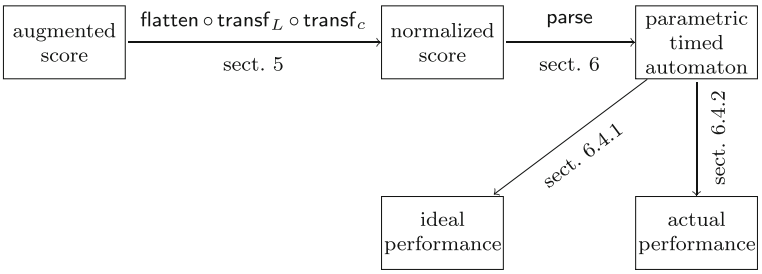



**Fig. 5** Specifying the semantics of the *Antescofo* augmented score language. The labels on the edges correspond to the functions used to compute one representation from another and are defined in the mentioned sections

## 5 Syntax

An *Antescofo* score is a specification of both an instrumental (e.g., soloist) part and the accompaniment actions. As explained above, this specification is given as an input to the system within one augmented score (see Fig. 3).

The syntax for the instrumental part allows the construction of complex events such as trills, glissandi, improvisation boxes and also continuous events. Actions are interpreted with a *message-passing* semantics: every atomic action *a* corresponds to a specific command sent to the audio environment, for instance to a sound synthesis module or a real-time audio processing system such as Max/MSP (Puckette 1991). The syntax for actions provides constructions for the *grouping* of actions in order to create polyphonic phrases, as well as loops and continuous trajectories. The timing of delays for actions can be expressed in relative time (in beats) or in physical time (in milliseconds).

In the textual format of specifications, the composers are also allowed to create nested hierarchies within electronic phrases (groups inside groups), employ macro expansions and use data flow functionals, see Cont (2011) for a complete descriptions.

In this section, we propose a simplified abstract syntax for scores, defined by a formal grammar in the next section. It differs from the concrete textual *Antescofo* syntax mentioned above by many details and additional constructs, but contains the essential features that permit us to describe formally the temporal behavior of *Antescofo*.

### 5.1 Abstract score

We define the *score* as an interleaving of instrumental events and associated actions or group of actions, with the following grammar.

$$\begin{aligned}
\mathit{score} &:= \mathit{event} \mid \mathit{event}\ \mathit{score} \mid (d\ \mathit{group})\ \mathit{score} \mid (d\ \mathit{loop})\ \mathit{score} \mid (d\ \mathit{cont})\ \mathit{score} \\
\mathit{event} &:= (e\ c) \\
\mathit{group} &:= \texttt{group}\ \ell\ \mathit{synchro}\ \mathit{error}\ (d\ \mathit{action})^+ \\
\mathit{loop} &:= \texttt{loop}\ \ell\ \mathit{synchro}\ \mathit{error}\ p\ n\ (d\ \mathit{action})^+
\end{aligned}$$

$$
\begin{aligned}
\textit{action} &:= a \mid \textit{group} \mid \textit{loop} \mid \textit{cont} \\
\textit{cont} &:= \texttt{cont}\ \ell\ \textit{step}\ \textit{error}\ (d\ \mathbf{v})^+ \\
\textit{synchro} &:= \texttt{loose} \mid \texttt{tight} \\
\textit{error} &:= \texttt{local} \mid \texttt{global}
\end{aligned}
$$

A score is a sequence of events and actions. An event $e \in \mathcal{E}$ denotes an instrumental event and $c$ his duration (see Section 5.2). An action $a \in \mathcal{A}$ denotes an atomic action, and the constructions *group*, *loop* and *cont* describe compound actions, respectively groups, loops and continuous actions (see Section 5.3). A delay $d \in \mathbb{Q}_+$ before an action denotes a tempo relative delay (number of beats, see Section 5.5). Loops specify a tempo relative period $p \in \mathbb{Q}_+$ (in beats) and an iteration number $n \in \mathbb{N}$ that represents the number of times the body of a loop will be iterated. A continuous group corresponds to the sampling at rate $\textit{step} \in \mathbb{Q}_+$ of the piecewise linear functions specified by the timed sequence $(d\ \mathbf{v})^+$ of control vectors $\mathbf{v}$ (see Sections 5.3.3 and 6.1). Each compound action has also a label $\ell \in \mathcal{L}$. The attributes *synchro* and *error* specify respectively a synchronization strategy and an error handling strategy, as explained in Section 5.6.

*Example 4* Our running example pictured in Fig. 6 is composed of two events $e_0$ and $e_1$ and two simple groups $g_1$ and $g_2$. Group $g_1$ is composed of two atomic actions $a_{11}$ and $a_{12}$. The synchronisation attribute of $g_1$ is `tight`.

## 5.2 Instrumental part

We assume a predefined set $\mathcal{E} = \{e_0, \ldots, e_n\}$ of symbolic atomic instrumental events (note, chord, silence, trills, etc.) totally ordered by an ordering $<$ compatible with the indexes. Moreover, we assume for technical convenience that the events occurring in the score are pairwise distinct, and that the event immediately preceding $e_i$ in the score (when $i \geq 1$) is $e_{i-1}$ and the event immediately succeeding $e_i$ (when $i \leq n$) is $e_{i+1}$. Hence $e_0$ and $e_n$ are respectively the first and last event of the score. Note that

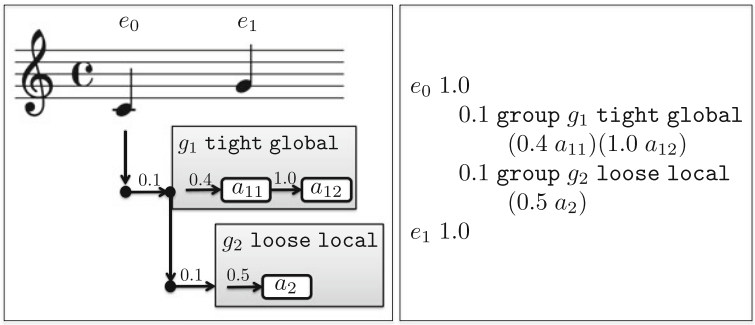



**Fig. 6** Score diagram and the *Antescofo* augmented score corresponding to the running example. The musical notes corresponds to the events performed by the musician and the rest to the electronic actions

the musical notation language offered by *Antescofo* is much richer, but this hypothese is sufficient for our modeling purpose.

In the score, the duration $c$ of an event $(e_i\ c)$ is also the interval of time between the start of $e_i$ and the start of $e_{i+1}$ (this duration fluctuates in an actual performance). From now on, the duration $c$ of an event $e$ in $(e_i, c)$ will be denoted by $c_i$.

### 5.3 Electronic part

The electronic parts of a score specify the actions triggered after an event detection or another action. Indeed, any action is bound to an event or to the preceding action. In Antescofo, an action is scheduled in real time after the detection of the event to which it is bound, as described in Section 7.

We assume a predefined finite set of atomic actions $\mathcal{A}$. For technical convenience, we assume, like for events, that all the atomic actions occurring in a score are distinct.

For compositional purposes, it seems more natural to be able to express accompaniment actions as phrases as opposed to small segments within instrumental note onsets. We describe below three primitives to compose actions and used to start a sequence of actions from a trigger event or from another action, in the instrumental score. They are referred under the generic name of *groups* and qualified as *simple*, *periodic* (loop) or *continuous*. These constructions allow the polyphonic phrases composition including a precise management of the temporality of their elements. Every group has a unique label $\ell$, in a set $\mathcal{L}$.

Simple and periodic groups can be arbitrarily nested into each other. Continuous groups occur only at the bottom of the nesting hierarchy. Note that atomic action cannot occur at the top-level of the abstract syntax. This is not a real restriction as we can define a simple group containing a single action. This way, we ensure that every action as an associated error handling strategy (see Section 5.6).

#### *5.3.1 Simple groups*

The `group` construction allows to describe logically within a same block several actions that share common properties of synchronization and errors handling strategies.

#### *5.3.2 Periodic group*

The `loop` construction is similar to the previous one but the actions composing the loop are iterated a given number of times. Each iteration takes the same amount of time, a period, which must be greater than the total duration of the actions in the loop body.

#### *5.3.3 Continuous group*

The continuous group `cont` is a primitive allowing the composer to control some process parameterized by $q$ parameters which evolves perceptively in a "continuous way". The sampling rate *step* can be as small as needed to achieve perceptual continuity. An action is emitted for each sample with the actual values of the $q$ parameters as argument.

The $q$ parameters are sampled at a given rate *step* by a linear interpolation between the reference values given by a timed sequence $(d_1\ \mathbf{v}_1) \ldots (d_k\ \mathbf{v}_k)$. The delay

$d_i$ specifies the duration of the interpolation between $\mathbf{v}_i$ and $\mathbf{v}_{i+1}$ and the dimension of the vectors $\mathbf{v}_i$ is $q$.

If the continuous group starts at time $t_0$, the $k$th parameter goes from $\mathbf{v}^k_{j-1}$ to $\mathbf{v}^k_j$ during the time interval $[t_0 + \sum_i^{j-1} d_i, t_0 + \sum_i^j d_i]$ where $1 < j$ and $\mathbf{v}^k_j$ denotes the $k$th component of the vector $\mathbf{v}_j$.

### 5.4 Parallelism

The actions in a group are played in parallel to the stream of input events. Nesting of groups is also a parallel composition: a nested group is played in parallel with the actions following it in the embedding group, not in sequence (in the same manner that every group is played in parallel to the events). This specificity of the hierarchical composition in the *Antescofo* score language has been introduced following musical expressiveness motivations (see Sections 2 and 5.6).

While the ability of authoring parallel phrases (as opposed to segmented and chopped actions) makes the act of authoring more appealing for a composer (Manoury 2007), such timing overlaps create important issues both for real-time coordination of events with the live performer and also their authoring.

### 5.5 Delays

Every atomic event $e_i$, atomic action $a$ or group is specified together with a delay. There is a different interpretation of the delays for events and actions.

- The duration $c_i$ specified after an event $e_i$ (with $0 \leq i \leq n$) is the number of beats expected to flow (i.e., the delay) between $e_i$ and the next event $e_{i+1}$.
  We assume by convention that $c_n = 0$.
- The delay $d$ before an atomic action or a group is the number of beats to wait to launch the action after the
  element which *syntactically precedes* it in the score. This element can be an event or an action.

Hence the sequence $(e_0\ c_0)(e_1\ c_1)\ldots(e_n\ 0)$ can be seen as a simplified presentation of the instrumental part represented on the top line of the graphical view of scores in Figs. 2 and 6. Every group, loop or continuous action can be seen as another line in the graphical view of score (the boxes in Fig. 2).

In Antescofo, the delays are expressed either relatively to the tempo (number of beats), like in traditional music notation, or in physical time (milliseconds). In the above abstract syntax, we have included only the former. The extension to latter would be a straightforward extension of the syntax and the semantics presented Section 7.

*Example 5* In the example of the Fig. 6, when the first note $e_0$ is detected the first `group` $g_1$ will be launched 0.1 beats after the occurrence of $e_0$. Action $a_{11}$ will be launched 0.4 beats after the start of $g_1$. Action $a_{12}$ follows $a_{11}$ after a delay of 1.0 beat. The group $g_2$ is launched 0.1 beat after the start of $g_1$ and proceed in parallel.

These delays are counted in the tempo scale inferred by the listening machine. The tempo is inferred "continuously" and change at the detection of each events.

## 5.6 Synchronization strategies

The musician's performance is subject to many variations from the score. There are several ways to adapt to this musical indeterminacy based on specific musical context. The musical context that determines the correct synchronization and error handling strategies is at the composer or arranger's discretion. To this end, we propose two explicit synchronization strategies taking into account performance variations. They are chosen by the composer and specified as an attribute for each group. Next paragraph we introduce two error handling strategies to manage the errors of the musician and the recognition algorithm.

### *5.6.1 Loose synchronization*

By default, once a group is launched, the scheduling of its sequence of relatively-timed actions follows the real-time changes of the tempo from the musician. This synchronization strategy is qualified as `loose`.

Figure 7 attempts to illustrate this within a simple example: Fig. 7a shows the *ideal performance* or how actions and instrumental score is given to the system. In this example, an accompaniment phrase is launched at the beginning of the first event from the human performer. The accompaniment in this example is a simple group consisting of four actions that are written parallel (and thus synchronous) to subsequent events of the performer in the original score, as in Fig. 7a. In a regular score following setting (i.e., correct listening module) the action group is launched synchronous to the onset of the first event. For the rest of the actions however, the synchronization strategy depends on the dynamics of the performance. This is demonstrated in Fig. 7b and c where the performer hypothetically accelerates or decelerate the consequent events in her score. In these two cases, the delays between the actions will grows or decreases until converge to the performer tempo.

The `loose` synchronization strategy ensures a fluid evolution of the actions launching but it does not guarantee a precise synchronization with the events played by the musician. Although this fluid behavior is desired in certain musical configurations, it seems natural to propose an alternative synchronization strategy where the electronic actions will be launched as close as possible to the events detection.

### *5.6.2 Tight synchronization*

If a group is `tight`, its actions will be dynamically analyzed to be triggered not only using relative timing but also relative to the nearest event in the past. Here, the nearest event is computed in the ideal timing of the score.

This feature evades the composer from segmenting the actions of a group to smaller segments with regards to synchronization points and provide a high-level vision during the compositional phase. A dynamic scheduling approach is adopted to implement the `tight` behavior. During the execution the system synchronize the next action to be launched with the corresponding event.

Note that the arbitrary nesting of groups with arbitrary synchronization strategies do not always make sense: a group `tight` nested in a group `loose` has no well defined triggering event (because the start of each action in the `loose` group are supposed to be synchronized dynamically with the tempo). All other combinations

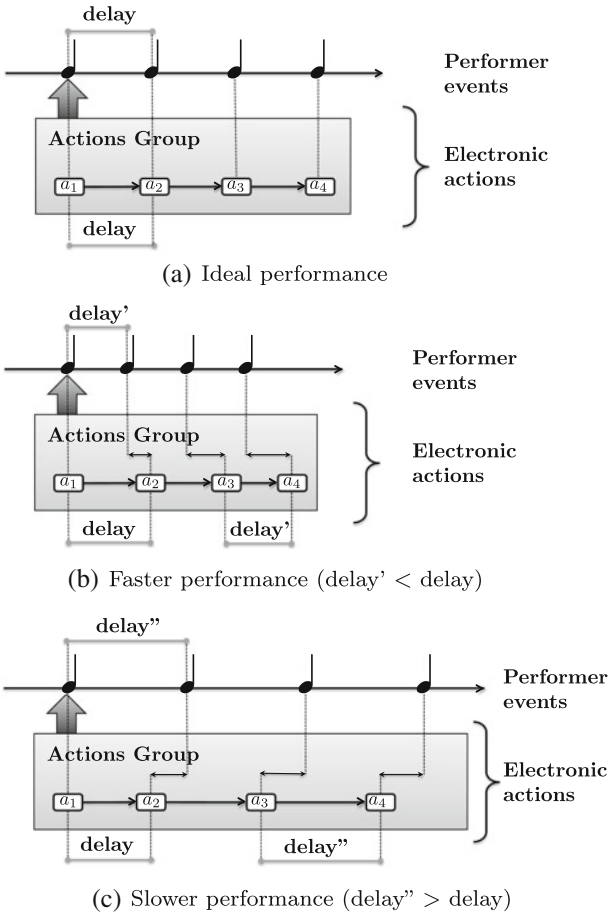



**Fig. 7** The effect of tempo-only synchronization for accompaniment phrases: illustration for different tempi. In the score, the actions are written to occur simultaneously with the notes, cf. figure **a**. Figures **b** and **c** illustrate the effect of a faster or a slower performance. In these cases, the tempo inferred by the listening machine converges towards the actual tempo of the musicians. Therefore, the delays, which are relative to the inferred tempo, vary in absolute time to converge towards the delay between the notes observed in the actual performance

are meaningful. To acknowledge that, we suppose that groups nested in a `loose` group, are `loose`. This is not enforced by the syntax but the transformation presented in Section 6.3 takes this constraint into account.

*Example 6* The `tight` group $g_1$ is composed of two actions $a_{11}$ and $a_{12}$. The action $a_{11}$ is launched $0.1 + 0.4 = 0.5$ beats after the detection of $e_0$. The action $a_{12}$ is launched in the score $0.5 + 1.0 = 1.5$ beats after $e_0$ which is latter that the specified occurrence of $e_1$: the `tight` strategy implies that during the performance, the action $a_{12}$ will be launched $0.5 = 1.5 - 1.0$ beats after $e_1$ (the number 1.0 represents the duration of $e_0$).

### 5.7 Missed event errors strategies

Parts but not all of the errors during the performance can be handled directly by the listening modules (such as false-alarms and missed events by the performer). The critical safety of the accompaniment part can thus be reduced to handling of missed events (whether missed by the listening module or human performer). The natural

question to ask in this case is what the system should do in case of a missed event? Should the associated actions be performed or not? The answer to this question seems more musical than technical: In some automatic accompaniment situations, one might want to dismiss associated actions to a missed event if the scope of those actions does not bypass that of the current event at stake. On the contrary, in many live electronic situations such actions might be initializations for future actions to come.

This discussion shows that while such considerations can be addressed automatically in special cases, various error handling strategies should be available in any specification language for automatic accompaniment and interactive computer music pieces.

The problem of error handling boils down to the ability of attributing *scopes* to accompaniment phrases. A group is said to be `local` if it should be dismissed in the absence of its triggering event during live performance; and accordingly it is `global` if it should be launched in priority and immediately if the system recognizes the absence of its triggering event during live performance. Once again, the choice of a group being `local` or `global` is given to the discretion of the composer or arranger.

The combination of the synchronization attributes (`tight` or `loose`) and error handling attributes (`local` or `global`) for a group of accompaniment actions give rise to four distinct situations. Figure 8 attempts to showcase these four situations for

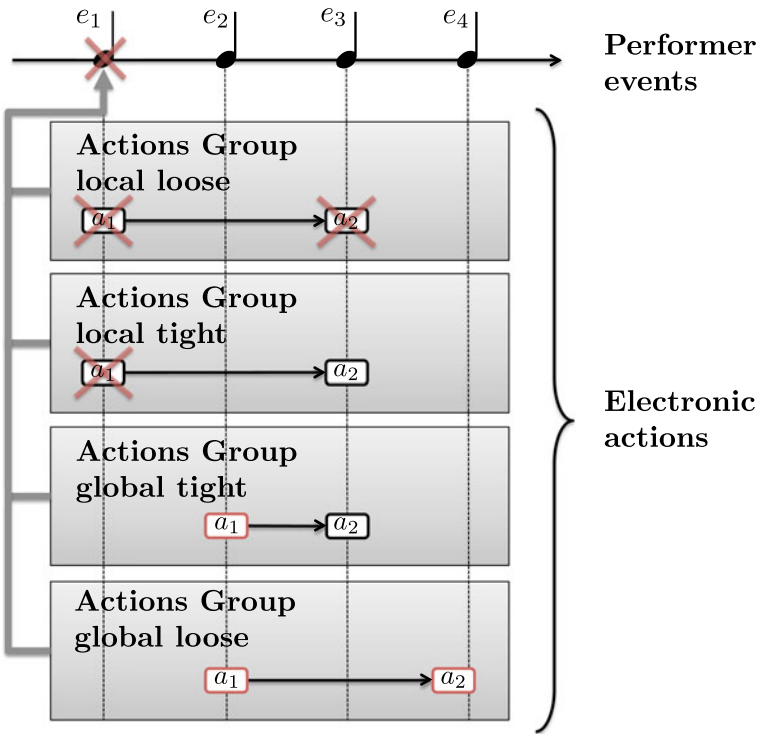



**Fig. 8** Accompaniment behavior in case of missed event for four synchronization and error handling strategies

a simple hypothetical performance setup similar to Fig. 7. In this example, the score is assumed to demand for four distinct performer events ($e_1$ to $e_4$) with grouped actions whose two actions are initially aligned on $e_1$ and $e_3$. Figure 7 illustrates the system behavior in case $e_1$ is missed but the rest detected without tempo change during live performance for the four configurations discussed above. Note that $e_1$ is detected as missed (in real-time) once of course $e_2$ is reported.

While with respect to a specific musical application some of these behaviors might look undesirable, it is worth to note that each combination corresponds to a musical situation encountered in authoring of mixed interactive pieces:

- `local` and `loose`: A block that is both local and loose correspond to a musical entity with some sense of rhythmic independence with regards to synchrony to its counterpart instrumental event, and strictly reactive to its triggering event onset (thus dismissed in the absence of its triggering event).
- `local` and `tight`: Strict synchrony of inside actions whenever there's a spatial correspondence between events and actions in the score. However actions within the strict vicinity of a missing event are dismissed. This case corresponds to an ideal concerto-like accompaniment system.
- `global` and `tight`: Strict synchrony of corresponding actions and events while no actions is to be dismissed in any circumstance. This situation corresponds to a strong musical identity that is strictly tied to the performance events.
- `global` and `loose`: An important musical entity with no strict timing in regards to synchrony. Such identity is similar to integral musical phrases that have strict starting points with *rubato* type progressions (free endings).

## 6 Normalization of Antescofo augmented scores

In order to give a semantics to the score language defined in the previous section, we start by rewriting some constructions into their equivalent program fragments using only a strict subset of the full syntax.

In the next section, we show how a continuous group can be rewritten into an expression involving only groups. In Section 6.2, we present the rewriting of loops into groups. These two transformations defines the semantics of continuous groups and loops.

To simplify the expression of the semantics of error handling of `tight` groups, we present in Section 6.3 a transformation which turn `tight` groups into *zero-delay*, *explicit*, *flat*, *singleton*, *exact* `tight` groups:

- *Zero-delay* groups are group that occurs simultaneously with the event or the action that triggers the group.
- In a *flat* group, actions are either elementary action or `loose` groups (remember loose groups contain only elementary and loose groups).
- A *singleton* group contains only one action.
- Actions in an *exact* `tight` group $g$ are all triggered by the same event which triggers $g$.

A score where there is only loose groups and zero-delay, flat, singleton exact `tight` groups is said in *normal form*. The semantics of normal form score is defined in the next section.

### 6.1 Continuous group syntax rewriting

For the sake of the simplicity, we suppose that in a continuous group the delays $d_i$ are all divisible by the step *step*. The score transformation $\mathsf{transf}_c$ rewrites continuous groups using nested groups by induction on the structure of a score:

$$\begin{aligned}
\mathsf{transf}_c(\epsilon) &= \epsilon \\
\mathsf{transf}_c(\mathit{eventscore}) &= \mathit{event}\mathsf{transf}_c(\mathit{score}) \\
\mathsf{transf}_c\big((\mathit{dgroup})\mathit{score}\big) &= \mathsf{transf}_c(\mathit{d\ group})\ \mathsf{transf}_c(\mathit{score}) \\
\mathsf{transf}_c\big((\mathit{dloop})\mathit{score}\big) &= \mathsf{transf}_c(\mathit{d\ loop})\ \mathsf{transf}_c(\mathit{score}) \\
\mathsf{transf}_c\big((\mathit{dcont})\mathit{score}\big) &= \mathsf{transf}_c(\mathit{d\ cont})\ \mathsf{transf}_c(\mathit{score}) \\
\mathsf{transf}_c(\mathit{d\ a}) &= (\mathit{d\ a})
\end{aligned}$$

$$\begin{aligned}
&\mathsf{transf}_c\big(d\ \texttt{group}\ \ell\ \mathit{synchro\ error}\ (d_1\ \mathit{action}_1)\ldots(d_n\ \mathit{action}_n)\big) \\
&= d\ \texttt{group}\ \ell\ \mathit{synchro\ error}\ \mathsf{transf}_c(d_1\ \mathit{action}_1)\ldots\mathsf{transf}_c(d_n\mathit{action}_n)
\end{aligned}$$

$$\begin{aligned}
&\mathsf{transf}_c\big(d\ \texttt{loop}\ \ell\ \mathit{synchro\ error}\ p\ n\ (d_1\ \mathit{action}_1)\ldots(d_m\ \mathit{action}_m)\big) \\
&= d\ \texttt{loop}\ \ell\ \mathit{synchro\ error}\ p\ n\ \mathsf{transf}_c(d_1\ \mathit{action}_1)\ldots\mathsf{transf}_c(d_m\ \mathit{action}_m)
\end{aligned}$$

The interesting case is the transformation of a `cont`:

$$\begin{aligned}
&\mathsf{transf}_c\big(d\ \texttt{cont}\ \ell\ \mathit{step\ error}\ (d_1\ \mathbf{v}_1)\ldots(d_n\ \mathbf{v}_n)\big) \\
&= d\ \texttt{group}\ \ell\ \mathit{synchro\ error}\ (0\ g_1)\ (d_1\ g_2)\ldots(d_{n-1}\ g_n)
\end{aligned}$$

with

$$g_i = \texttt{group}\ \ell\ \mathit{synchro\ error}\ (0\ w_{i,1})\ (\mathit{step}\ w_{i,2})\ldots(\mathit{step}\ w_{i,m_i})$$

where $m_i = \frac{d_i}{\mathit{step}}$ and for $1 \le i < n$ and $1 < j \le m_i$:

$$\mathbf{w}_{i,1} = \mathbf{v}_i, \qquad \mathbf{w}_{i,j} = \mathbf{w}_{i,j-1} + \frac{\mathit{step}}{d_i}(\mathbf{v}_{i+1} - \mathbf{v}_i)\ .$$

In the previous transformation, each group $g_i$ implements the linear interpolation between the two control vectors $\mathbf{v}_i$ and $\mathbf{v}_{i+1}$. It is illustrated in Fig. 9.

### 6.2 Loop rewriting

The score transformation $\mathsf{transf}_L$ is defined by induction on the score structure and expands a loop into the corresponding successive groups.

$$\begin{aligned}
\mathsf{transf}_L(\epsilon) &= \epsilon \\
\mathsf{transf}_L(\mathit{eventscore}) &= \mathit{event}\mathsf{transf}_L(\mathit{score}) \\
\mathsf{transf}_L\big((\mathit{dgroup})\mathit{score}\big) &= \mathsf{transf}_L(\mathit{d\ group})\mathsf{transf}_L(\mathit{score}) \\
\mathsf{transf}_L\big((\mathit{dloop})\mathit{score}\big) &= \mathsf{transf}_L(\mathit{d\ loop})\mathsf{transf}_L(\mathit{score})
\end{aligned}$$

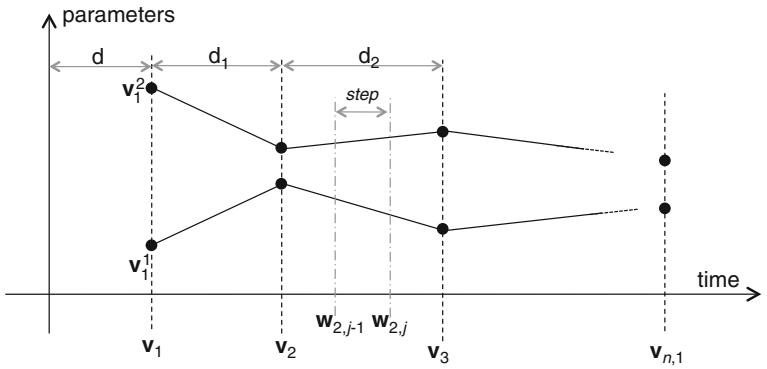



**Fig. 9** Interpolation of the control vectors in a continuous group

$$\mathsf{transf}_L(d\ a) = (d\ a)$$

$$\mathsf{transf}_L\big(d\ \texttt{group}\ \ell\ \textit{synchro error}\ (d_1\ \textit{action}_1)\ldots(d_n\ \textit{action}_n)\big)$$
$$= d\ \texttt{group}\ \ell\ \textit{synchro error}\ \mathsf{transf}_L(d_1\ \textit{action}_1)\ldots\mathsf{transf}_L(d_n\ \textit{action}_n)$$

The interesting case is the transformation of a `loop`. This is done by induction on the number of iterations of the loop body. The base case is obvious:

$$\mathsf{transf}_L\big(d\ \texttt{loop}\ \ell\ \textit{synchro error}\ p\ 0\ (d\ \textit{action})^+\big) = \epsilon$$

The inductive case simply unfolds one time the loop:

$$\mathsf{transf}_L\big(d\ \texttt{loop}\ \ell\ \textit{synchro error}\ p\ n\ (d_1\ \textit{action}_1)\ldots(d_n\ \textit{action}_n)\big)$$
$$= \mathsf{transf}_L\big(d\ \texttt{group}\ \ell\ \textit{synchro error}\ (d_1\ \textit{action}_1)\ldots(d_n\ \textit{action}_n)\big)$$
$$\mathsf{transf}_L\big(0\ \texttt{loop}\ \ell\ \textit{synchro error}\ p\ (n-1)\ (d_1\ \textit{action}_1)\ldots(d_n\ \textit{action}_n)\big)$$

## 6.3 Tight normalization

In the following, we suppose that the previous transformations have been applied: that is, there no more loops and continuous groups in the score.

The normalization of tight groups is done in two steps. The first transformation flatten makes all group `tight` with a zero delay and flat (that is, there is no nesting of tight groups).

The second phase will spread tight groups in order to associate them directly to its triggering event.

### *6.3.1 Flattening of tight groups*

The transformation flatten is defined by induction on the structure of a score.

$$\mathsf{flatten}(\epsilon) = \epsilon$$

$$\mathsf{flatten}\ (\textit{eventscore}) = \textit{event}\ \mathsf{flatten}\ (\textit{score})$$

$$\begin{aligned}&\mathsf{flatten}\left((d\ \texttt{group}\ \ell\ \ \texttt{loose}\ \mathit{error}\ (d_1\ \mathit{action}_1)\ldots(d_n\ \mathit{action}_n))\mathit{score}\right)\\&\quad=\left(d\ \texttt{group}\ \ell\ \texttt{loose}\ \mathit{error}\ \mathsf{flatten}_l(d_1\ \mathit{action}_1)\ldots\right.\\&\qquad\left.\ldots\mathsf{flatten}_l(d_n\ \mathit{action}_n)\right)\mathsf{flatten}\ (\mathit{score})\end{aligned}$$

The function $\mathsf{flatten}_l$ threads the transformation through the nesting of groups:

$$\begin{aligned}\mathsf{flatten}_l(d\ a)&=(d\ a)\\\mathsf{flatten}_l(d\ g_{\text{loose}})&=\mathsf{flatten}(d\ g_{\text{loose}})\end{aligned}$$

In the previous equation, $g_{\text{loose}}$ represent any groups with a `loose` attribute. We explicitly state that in the threading of flatten, `tight` groups nested in a `loose` group become `loose`:

$$\begin{aligned}&\mathsf{flatten}_l\left(d\ \texttt{group}\ \ell\ \ \texttt{tight}\ \mathit{error}\ (d_1\ \mathit{action}_1)\ldots(d_n\ \mathit{action}_n)\right)\\&\quad=\mathsf{flatten}\left(d\ \texttt{group}\ \ell\ \texttt{loose}\ \mathit{error}\ (d_1\ \mathit{action}_1)\ldots(d_n\ \mathit{action}_n)\right)\end{aligned}$$

It remains to define flatten on `tight` groups. The idea is to replace one tight group $(d\ g_{\text{tight}})$ by the actions it contains. However, this will delay the following actions. To avoid this delay, each action of the tight group is encapsulated in one tight group with a zero delay. Doing that, the actions following $g_{\text{tight}}$ arise too early by a duration of $d$ and must be delayed.

$$\begin{aligned}&\mathsf{flatten}\left(d\ \texttt{group}\ \ell\ \ \texttt{tight}\ \mathit{error}\ (d_1\ \mathit{action}_1)\ldots(d_n\ \mathit{action}_n)\ \mathit{score}\right)\\&\quad=\mathsf{flatten}_t(\mathit{error},d+d_1,\mathit{action}_1)\ldots\\&\qquad\ldots\mathsf{flatten}_t(\mathit{error},d+d_1+\cdots+d_n,\mathit{action}_n)\ \mathsf{flatten}\left(\mathsf{delay(d,\ score)}\right)\end{aligned}$$

This equation relies on two auxiliary functions delay and $\mathsf{flatten}_t$. The function delay is used to delay the actions in a score bound with a previous event:

$$\begin{aligned}\mathsf{delay(d},\epsilon)&=\epsilon\\\mathsf{delay(d,\ event\ score)}&=\mathit{event\ score}\\\mathsf{delay(d,\ (d_g\ group)\ score)}&=(d+d_g\ \mathit{group})\ \mathit{score}\end{aligned}$$

Indeed, to delay a score, it is enough to delay the first action of this score (because the start of an action is relative to the previous one, delaying the first action of a sequence will delay all the subsequent actions). An event is not affected and reset the temporal basis for subsequent actions (so there is no need to propagate the delay).

The function $\mathsf{flatten}_t$ threads the function flatten over the hierarchy of actions in the tight group and encapsulate elementary actions and loose groups in a tight singleton:

$$\begin{aligned}\mathsf{flatten}_t(\mathit{error},d,a)&=(0\ \texttt{group}\ \ell\ \texttt{tight}\ \mathit{error}\ (d\ a))\\\mathsf{flatten}_t(\mathit{error},d,g_{\text{loose}})&=(0\ \texttt{group}\ \ell\ \texttt{tight}\ \mathit{error}\ \mathsf{flatten}_l(d\ g_{\text{loose}}))\end{aligned}$$

The transformation must be done recursively to handle tight groups nested in tight groups:

$$\begin{aligned}&\mathsf{flatten}_t(\mathit{error}, d, \texttt{group}\ \ell\ \texttt{tight}\ \mathit{error}\ (d_1\ \mathit{action}_1)\dots(d_n\ \mathit{action}_n))\\ &\quad= \mathsf{flatten}(0\ \texttt{group}\ \ell\ \texttt{tight}\ \mathit{error}\ (d+d_1\ \mathit{action}_1)(d_2\ \mathit{action}_2)\dots(d_n\ \mathit{action}_n)\end{aligned}$$

*6.3.2 Making tight groups exact*

The second step of the transformation anchors each zero-delay singleton `tight` group with the right event in the score.

*Example 7* To understand the problem, consider the score of our running example.

$$\begin{aligned}\mathit{score} = e_1\ 1.0\\ 0.1\ \texttt{group}\ g_1\ \texttt{tight global}\ (0.4\ a_{11})(1.0\ a_{12})\\ 0.1\ \texttt{group}\ g_2\ \texttt{loose local}\ (0.5\ a_2)\\ e_2\ 1.0\end{aligned}$$

The action $a_{12}$ in the `tight` group associated to $e_0$ is supposed to occurs 1.5 beat after the start of $e_0$, that is, 0.5 beats after the occurrence of $e_1$. Therefore, it is possible to associate action $a_{12}$ to $e_1$, making the group *exact*. The exact corresponding score is thus given by:

$$\begin{aligned}\mathit{score} = e_0\ 1.0\\ 0.0\ \texttt{group}\ g_{11}\ \texttt{tight global}\ (0.5\ a_{11})\\ 0.2\ \texttt{group}\ g_2\ \texttt{loose local}\ (0.5\ a_2)\\ e_1\ 1.0\\ 0.0\ \texttt{group}\ g_{12}\ \texttt{tight global}\ (0.5\ a_{12})\end{aligned}$$

The previous transformation is achieved by the $\mathsf{transf}_t$ function defined by induction on the score structure:

$$\begin{aligned}\mathsf{transf}_t\big((e\ c)\mathit{score}\big) &= (e\ c)\mathsf{transf}_{t1}(c, \mathit{score})\\ \mathsf{transf}_{t1}(c_{\text{lag}}, \epsilon) &= \epsilon\\ \mathsf{transf}_{t1}(c_{\text{lag}}, (e\ c)\mathit{score}) &= (e\ c)\mathsf{transf}_{t1}(c, \mathit{score})\\ \mathsf{transf}_{t1}(c_{\text{lag}}, (d\ g_{\text{loose}})\mathit{score}) &= (d\ g_{\text{loose}})\mathsf{transf}_{t1}(c_{\text{lag}} - d, \mathit{score})\end{aligned}$$

The auxiliary function $\mathsf{transf}_{t1}$ takes an additional argument which represents the time remaining until the next event. The action of $\mathsf{transf}_{t1}$ on a tight group depends of this remaining time:

$$\begin{aligned}&\mathsf{transf}_{t1}\big(c_{\text{lag}}, (0\ \texttt{group}\ \ell\ \mathit{synchro}\ \texttt{tight}\ (d\ a))\mathit{scoreevent}'\mathit{score}'\big)\\ &\quad= \text{if } d < c_{\text{lag}}\\ &\qquad(0\ \texttt{group}\ \ell\ \mathit{synchro}\ \texttt{tight}\ (d\ a))\mathsf{transf}_{t1}(c_{\text{lag}}, \mathit{scoreevent}'\mathit{score}')\\ &\qquad\text{otherwise}\\ &\qquad\mathsf{transf}_{t1}\big(c_{\text{lag}}, \mathit{scoreevent}'(0\ \texttt{group}\ \ell\ \mathit{synchro}\ \texttt{tight}\ (d - c_{\text{lag}}\ a)\mathit{score}'\big)\end{aligned}$$

In the previous equation, *event'* represents the event immediately succeeding the tight group in the rest of the score; that is, there is no occurrence of events in *score*.

## 7 Semantics

We present in this section an operational semantics for the performance of the abstract *Antescofo* augmented scores defined in Section 5 and put into normal form as explained in Section 6. The semantics is based on a model of networks of parametric timed automata (as described in Section 4), whose transitions are labelled by atomic events and actions and with parameters are used to represent the delays between the events played by a musician. Thus, the parametric timed automaton associated to an augmented score represents a family of possible performances. By instantiating these parameters in the parametric automaton, we obtain one timed automaton which represents one interpretation of the score (amongst all possible interpretations): this automaton accepts the timed trace (of events and actions) recording this interpretation.

The timed automaton relies on a set of clocks which evolve at the same rate and corresponds to the passing of time *in the scale of the tempo*. So, all delays are relative to the tempo scale and refer to this clock, assumed shared by all the automata of the network.

The automata network associated to an augmented score is built using the parallel composition of smaller automata associated to the groups and to the instrumental score. The motivation for the choice of this approach is twofold.

First, every automaton of the network represents intuitively one part of the score: either the instrumental part (the automaton described in Section 7.1 will simulate the performance of a musician) or one electronic part, corresponding to a group or an atomic action (Section 7.2). The construction is completed in Section 7.3 for error (missed events) handling. Moreover, these automata are composed using the synchronized product operation, giving semantics for the whole system (Section 7.4). The synchronized product defines intrinsically all the potential interleavings of events and actions.

Second, with the parameter automata model, different instantiations of the parameters will allow us to account for the various performances. Amongst all possible performances, the *idealized performance* corresponds to the instrumental part played by a machine, without errors and at the right tempi and in strict accordance with the specification of the score (Section 7.4.1). Only one execution is possible in the case of an ideal performance. An *actual performance* imply an elastic tempo and the handling of the possible errors of the musician or the listening machine (Section 7.4.2). This is accounted by an instantiation of the parameters representing the actual tempi observed during the performance. Despite the variations of the tempo, *Antescofo* exhibits a form of determinism. This is reflected by the fact that only one timed word, modulo permutations of simultaneous letters, will be accepted by the synchronized product automaton, after the corresponding instantiation of the parameters (Section 7.5).

### 7.1 Instrumental score model and actions triggering

We present now the construction of a network of timed automata associated to an augmented score given in the syntax presented in Section 5. In a first step, we

built one automaton corresponding to the instrumental part of the score, and some automata in charge of triggering the actions and group attached to the (instrumental) events in the score. The automaton corresponding to the events abstracts in some sense the listening machine: the timed words accepted by this automaton describe the sequences of recognized and missed events produced by the listening module. Automata corresponding to the groups (possibly nested) in the score are built in the next sections.

We assume that the score has been normalized using the functions of Section 6 and contains only loose groups and exact flat singleton tight groups, and no periodic nor continuous groups.

### *7.1.1 Locations*

The locations of the automaton associated to the instrumental part are represented as *circles*, and the locations of the automata associated to the groups launching electronic actions are represented as *rectangles*. In the following, a location labeled $\mathsf{prep}\, x$ corresponds to counting the delay that precedes the onset of an action or an event $x$. A location labeled $\mathsf{wait}\, x$ waits to synchronize with the emission of an event or a label $x$ by another part of the automaton.

### *7.1.2 Alphabet*

The alphabets of the constructed automata contain symbols in the following set $\Sigma = \mathcal{E} \cup \overline{\mathcal{E}} \cup \mathcal{A} \cup \mathcal{L} \cup \overline{\mathcal{L}}$ where $\mathcal{E}$ is the set of instrumental events in the score, $\overline{\mathcal{E}} = \{\overline{e} \mid e \in \mathcal{E}\}$ is a copy of $\mathcal{E}$ containing new events, $\overline{e}$ representing an error message telling that the expected event $e$ is missing, $\mathcal{A}$ is the set of atomic actions, $\mathcal{L}$ contains the set of group labels in the score, which will be used as new events for synchronization purposes and $\overline{\mathcal{L}} = \{\overline{\ell} \mid \ell \in \mathcal{L}\}$ is a copy of $\mathcal{L}$ containing new labels $\overline{\ell}$ for the case where the event triggering a tight group of label $\ell$ is missing.

### *7.1.3 Parameters*

We associate one parameter $\gamma_i$ to every event $e_i \in \mathcal{E}$. These parameters represent the delays between the events, and will be instantiated in Section 7.4 in order to define performances.

We have one clock $x$ for the automaton associated to the instrumental part, and some other clocks $y_e$ with $e \in \mathcal{E}$ and $y_\ell$ for $\ell \in \mathcal{L}$ for the other automata. We recall that all the clocks evolve at the same rate, as detailed in Section 7.4.2. However, the clocks can be reset independently.

### *7.1.4 Transitions*

The construction of the automata network associated to the *score* is realized by the function $\mathsf{parse}(score)$. It calls the recursive function $\mathsf{parse}_2(e, score)$, to which is passed the name $e$ of the last previously parsed event. The automaton corresponding to the instrumental part is obtained by parsing all the instrumental events in *score*, as defined in Fig. 10. Recall that we assume (Section 5) that the score always starts with the event $(e_0\ c_0)$.

For each event $e_i$ in the score, there are two alternatives: either $e_i$ is really sent (transition labeled by $e_i$), or it is missed (transition labeled by $\overline{e_i}$). When it

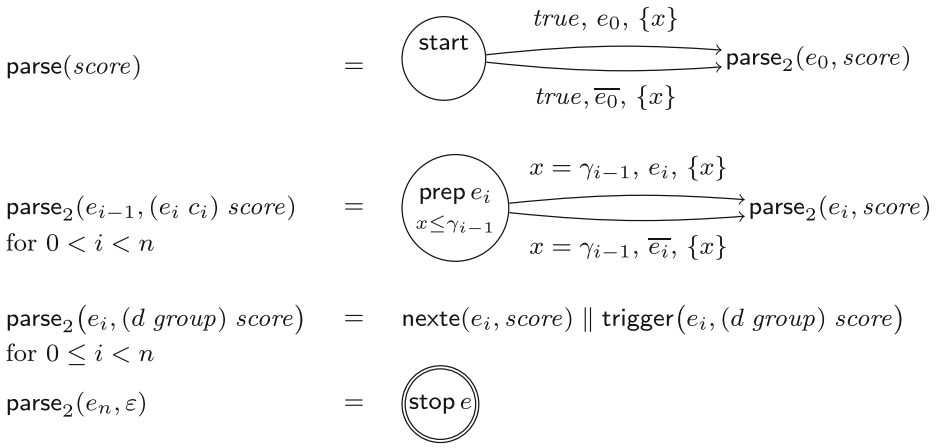



**Fig. 10** Automata for the instrumental part of the score

encounters a group, the function $\mathsf{parse}_2$ spawns an automata network, in parallel to the instrumental automaton. This automata network corresponds to the list of groups between the events $e_i$ and $e_{i+1}$ in the score (i.e., the list of groups triggered by $e_i$). It is built by the function trigger described below. The auxiliary function nexte will simply traverse the groups until the next event is found in the score.

$$\begin{array}{lcl} \mathsf{nexte}(e_i, (d\ group)\ score) & = & \mathsf{nexte}(e_i, score) \\ \mathsf{nexte}(e_i, (e_{i+1}\ c_{i+1})\ score) & = & \mathsf{parse}_2(e_i, (e_{i+1}\ c_{i+1})\ score) \end{array}$$

We call a *trigger* an automaton that will schedule a list of actions and groups that are triggered by a given event or label $\tau \in \mathcal{E} \cup \mathcal{L}$. The construction of this automaton, as well as the automaton corresponding to the hierarchy of groups triggered, is performed by a call to $\mathsf{trigger}(\tau, list)$, defined in Fig. 11, where *list* is the fragment of score starting with the triggered elements.

The transitions labelled by $\tau$ or $e_i$ in Fig. 11 enforce the synchronization of the start of the list *list* with the triggering event $\tau$ or $e_i$. The list of elements triggered is then parsed by the function $\mathsf{trigger}_2$ which is defined in Fig. 12.

The last two cases of the definition of $\mathsf{trigger}_2$ in Fig. 12 correspond to the end of the parsing of the list of top-level actions triggered by $e_{i-1}$, which are specified

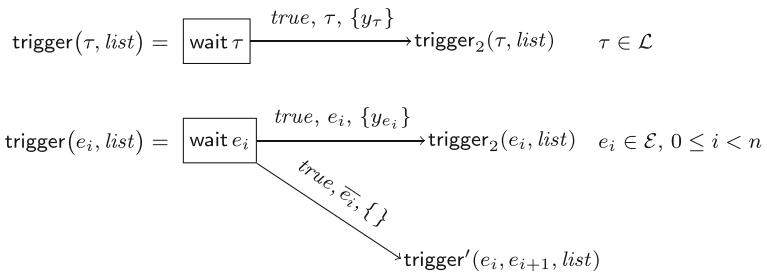



**Fig. 11** Construction of a trigger automaton, first part (continued in Figs. 12 and 13)

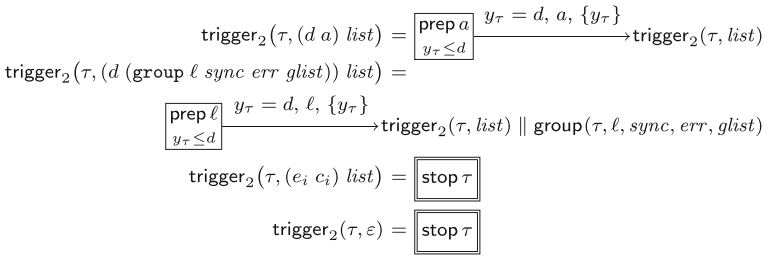



**Fig. 12** Construction of a trigger automaton, second part (following Fig. 11)

between $(e_{i-1}\ c_{i-1})$ and either $(e_i\ c_i)$ or the end of the score. The parsing of an atomic action $a$ is straightforward: the resulting automaton awaits the end of the delay $d$ using the clock $y_\tau$. Then this clock is reseted by crossing the transition towards the handling of the next action. The parsing of a group, in the second equation of Fig. 12, spawns an automaton corresponding to this group, constructed by the function group (defined in Section 7.2) to which are passed the attributes *sync*, defining the synchronization strategy, and *err*, defining the error handling strategy of the group, together with the content *glist* of the group. The group's label $\ell \in \mathcal{L}$ is used as a fresh event symbol for the synchronization between this new automaton and the trigger list. It does not correspond to an event of the score but is used to trigger the start of the nested group.

The function trigger$'(e_i, e_j, list)$, with $0 \le i < j < n$, corresponds to the case where the events $e_i$ to $e_{j-1}$ are missing, and $e_j$ is the next expected event. It is defined in Fig. 13. The list of the triggered elements is constructed by the function trigger$'_2$ in case of one or several missed events. This function differs from trigger$_2$ by the treatment of delays (see Section 7.3, Fig. 18). Notice that unlike trigger, the function trigger$'$, is never called with a group label.

Note that following the hypotheses that every event, action and group identifier occurs only once in the score (see Section 5), the locations of the above automata are unique.

## 7.2 Group models

There are two subcases for the definition of the function group, according to the synchronization strategy.

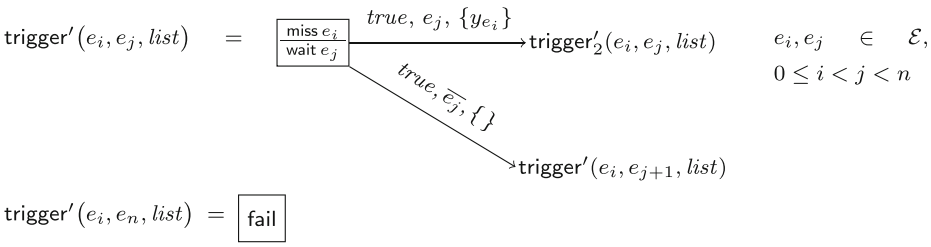



**Fig. 13** Construction of a trigger automaton, case of missed events (continued in Fig. 18)

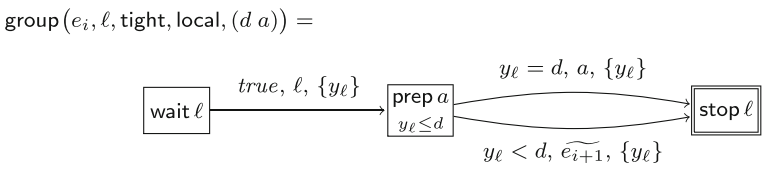



**Fig. 14** Automaton for a tight, local group containing a single action

### *7.2.1 Loose groups*

The construction of an automaton corresponding to a group with synchronization strategy loose is defined below by reusing the function trigger. This function will parse the elements of the group and spawn new automata for nested groups if necessary. Note that the error handling attribute *err* is simply ignored since we are in the case where the triggering event is not missing.

$$\mathsf{group}(\tau, \ell, \mathsf{loose}, \mathit{err}, \mathit{glist}) = \mathsf{trigger}(\ell, \mathit{glist})$$

### *7.2.2 Tight groups*

The case of the groups with a synchronization attribute set to tight is more complicated. Intuitively, whereas loose group flow asynchronously, independently of the events, once they are launched, a tight group is resynchronized with every event detected after the group has been launched. Roughly, this behavior is like the group has been split into several loose groups dispatched on successive events (see Section 5.6). Also, the semantics of tight groups depends on the error handling attribute, even when no event is missing, as explained in Section 7.3.2 below.

Following the simplifications of Section 6.3 we can assume that every tight group contains either exactly one action or exactly one loose subgroup, it is not nested in a tight or loose group, and has a group delay set to 0. The latter hypothesis implies that every tight group is at the top-level of the score, and hence that the trigger event of such a group (the first argument $\tau$ of the function $\mathsf{trigger}_2$) is necessarily an event $e_i \in \mathcal{E}$, for $(0 \le i < n)$. Moreover, the *glist* of the group (fifth argument of the function group) contains either an atomic action $(d\ a)$ or a nested group $G' = (d\ (\texttt{group}\ \ell'\ \mathsf{loose}\ \mathit{err}'\ \mathit{glist}'))$. The different cases are specified in Figs. 14, 15,

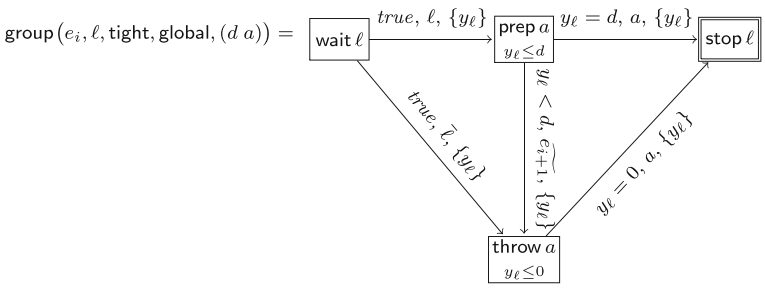



**Fig. 15** Automaton for a tight, global group containing a single action

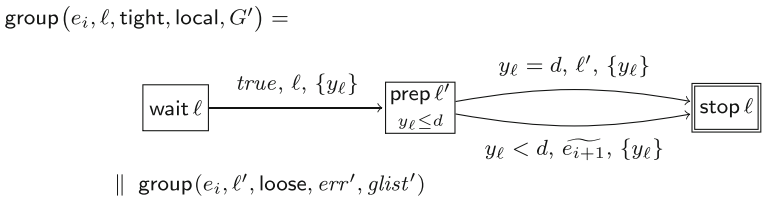



**Fig. 16** Automata for a tight, local group containing a single nested group

16 and 17, where $\widetilde{e_{i+1}}$ stands for $e_{i+1}$ or $\overline{e_{i+1}}$. In these Figures, the transitions labeled by $\overline{\ell}$ are used for launching groups whose triggering event is missing, see Section 7.3.2.

The expected behavior is that after the launching of the group, signaled by the event $\ell$ sent by a trigger, a delay of $d$ expires and then the action $a$ is sent before the detection of the event $e_{i+1}$ (otherwise, the group would have been tighten to the $e_{i+1}$ by definition of the normalizing functions of Section 6.3). If $e_{i+1}$ arrives (or $\overline{e_{i+1}}$ arrives in case $e_{i+1}$ is detected missing) earlier than expected, i.e. before the expiration of the delay $d$, then: if the group has attribute local, then $a$ (resp. $G'$) is not launched, and if the group has attribute global, then $a$ (resp. $G'$) is thrown immediately.

## 7.3 Handling missing events errors

It remains to define the functions for the case of missing triggering events. The function $\text{trigger}'_2(e_i, e_j, list)$ constructs automata for groups whose triggering event $e_i$ is missing as well as all events until $e_j$ excluded. Its definition, given in Fig. 18, is essentially the same as $\text{trigger}_2$, except that the delays before atomic actions and groups are set to 0. Note that the definition can assume that all elements of *list* are groups or events because it is called at the top-level of the score where there is no atomic actions according to the grammar.

Let us now explain the different definition cases of $\text{trigger}'_2$.

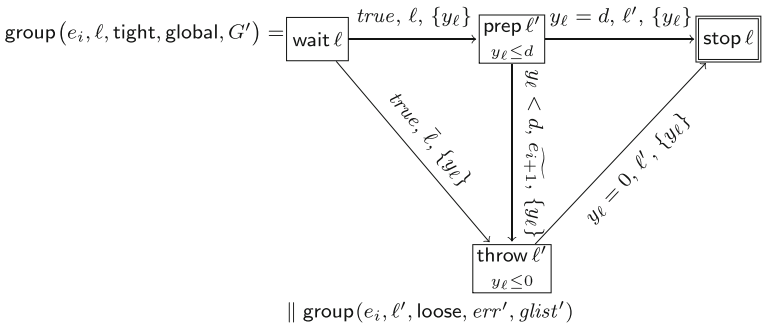



**Fig. 17** Automata for a tight, global group containing a single nested group

$$\mathsf{trigger}'_2(e_i, e_j, (d\ (\mathtt{group}\ \ell\ \mathit{sync}\ \mathsf{local}\ \mathit{glist}))\ \mathit{list}) = \mathsf{trigger}'_2(e_i, e_j, \mathit{list})$$

$$\mathsf{trigger}'_2(e_i, e_j, (d\ (\mathtt{group}\ \ell\ \mathsf{loose}\ \mathsf{global}\ \mathit{glist}))\ \mathit{list}) = \boxed{\begin{array}{c}\mathsf{prep}\ \ell \\ y_{e_i} \leq 0\end{array}} \xrightarrow{y_{e_i} = 0,\ \ell,\ \{y_{e_i}\}} \mathsf{trigger}'_2(e_i, e_j, \mathit{list})$$

$$\mathsf{trigger}'_2(e_i, e_j, (d\ (\mathtt{group}\ \ell\ \mathsf{tight}\ \mathsf{global}\ \mathit{glist}))\ \mathit{list}) = \boxed{\begin{array}{c}\mathsf{prep}\ \overline{\ell} \\ y_{e_i} \leq 0\end{array}} \xrightarrow{y_{e_i} = 0,\ \overline{\ell},\ \{y_{e_i}\}} \mathsf{trigger}'_2(e_i, e_j, \mathit{list})$$

$$\mathsf{trigger}'_2(e_i, e_j, (e_k\ c_k)\ \mathit{list}) = \mathsf{trigger}'_2(e_i, e_j, \mathit{list}) \quad \text{if } i < k < j$$

$$\mathsf{trigger}'_2(e_i, e_j, (e_j\ c_j)\ \mathit{list}) = \boxed{\boxed{\mathsf{stop}\ e_i}}$$

$$\mathsf{trigger}'_2(e_i, e_j, \varepsilon) = \boxed{\boxed{\mathsf{stop}\ e_i}}$$

**Fig. 18** Construction of a trigger automaton, case of missed events, second part (following Fig. 13)

### *7.3.1 Loose groups*

A group with synchronization attribute loose and error handling attribute local is skipped when the triggering event is missing. When its triggering event $e_i$ is missing, a group with synchronization attribute loose and error handling attribute global is launched when the first non-missing event $e_j$ is detected. This behavior is defined by $\mathsf{trigger}'$ in Fig. 13. Note that spawning a parallel automaton $\mathsf{group}(e_i, \ell, \mathsf{loose}, \mathsf{global}, \mathit{glist})$ is not needed in the definition of $\mathsf{trigger}'_2$ because this automaton has already been constructed by the function $\mathsf{trigger}_2$ (see Fig. 12). Moreover it is launched with a 0-delay, as defined by $\mathsf{trigger}'_2$ in Fig. 18.

### *7.3.2 Tight groups*

Similarly to the previous case, a group with synchronization attribute tight and error handling attribute local is skipped when the triggering event is missing.

Let us now consider the case of a group with synchronization attribute tight and error handling attribute global, and assume that the event $e_i, \ldots, e_{j-1}$ are missing, where $e_i$ is the triggering event of $G$ and $j > i$. As described in Section 5.7, there are two possible behaviors for the elements of $G$:

1. The elements located (from a temporal point of view) between $e_i$ and $e_j$ in the score are either not launched (case of error handling strategy local) or launched with a delay 0 (case of error handling strategy global).
2. The elements located after $e_j$ in the score are played normally (with specified delays).

Following the simplifications of Section 6.3, we consider only the case where $G$ contains only one action $a$ or one loose nested subgroup $G'$ and $a$ or $G'$ is located between $e_i$ and $e_{i+1}$, i.e. we only need to consider the case 1. In this case, the content of the group is launched with a delay set to 0: this is achieved by synchronization with the automata of Figs. 15 and 17, using a transition labeled $\overline{\ell}$ from location $\mathsf{wait}\ \ell$ to location throw.

*Example 8* The instrumental automaton of the running example has been given at Section 4. The construction of the rest of the automaton is launched by the function parse, cf. Fig. 19. The trigger automaton associated to $e_0$ launch $g_{11}$ and $g_2$ if $e_0$ is detected, else $\overline{g_{11}}$, see Fig. 20. The trigger associated to $e_1$ is described at Fig. 23. The automata associated to the functioning of the groups are pictured in Figs. 21, 22 and 24.

## 7.4 Product and parameter instanciations for performances

The model for the whole system is obtained by doing the synchronized product of the network of parametric timed automata returned by function parse(*score*). It

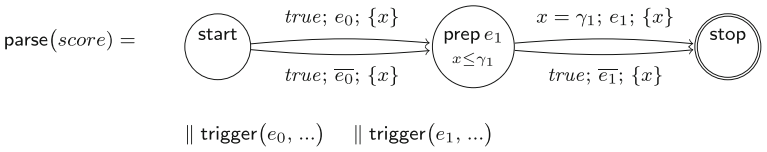



**Fig. 19** Example: subautomaton corresponding to the events in the instrumental part. The remaining automata are described in Figs. 20, 21, 22, 23 and 24

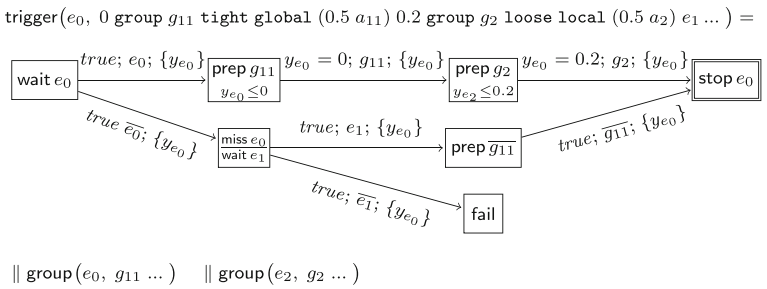



**Fig. 20** Example: trigger automaton corresponding to the event $e_0$

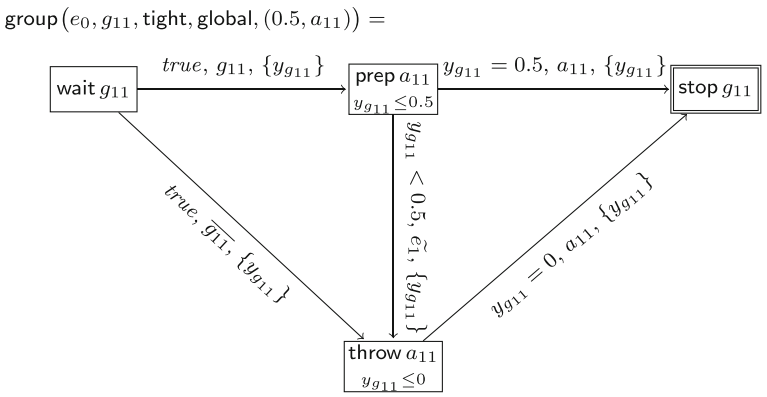



**Fig. 21** Example: subautomaton corresponding to the behavior of the $g_{11}$ group

$\mathsf{group}\,(e_0, g_2, \mathsf{loose}, \mathsf{local}, (0.5, a_2)) =$

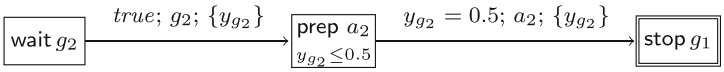



**Fig. 22** Example: subautomaton corresponding to the behavior of the $g_2$ group

$\mathsf{trigger}\,(e_1\ 0.0\ \mathtt{group}\ g_{12}\ \mathtt{tight}\ \mathtt{global}\ (0.5\ a_{12})\ ) =$

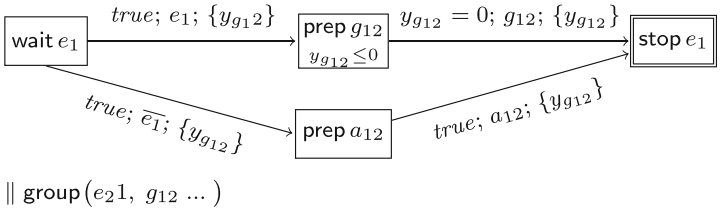



**Fig. 23** Example: trigger automaton corresponding to the event $e_1$

$\mathsf{group}\,(e_1, g_{12}, \mathsf{tight}, \mathsf{global}, (0.5, a_{12})) =$

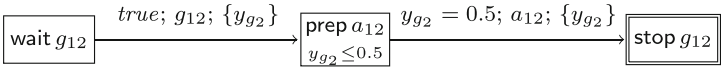



**Fig. 24** Example: subautomaton corresponding to the behavior of the $g_{12}$ group

representes all possible interleavings of events and actions that can occur during a performance with *Antescofo*.

The automata of the network still contain the parameters $\gamma_i$ representing the delays between events taken by a real performer. A performance will then be defined by an instantiation of these parameters with values expressing a time relating to the tempo (number of beats). We consider two cases below: an ideal performance and an actual performance.

### *7.4.1 Ideal performance*

An ideal performance is an execution where the instrumental part is played at the exact tempo specified in the score (such as it is played by a machine), and no events are missing. We obtain this behavior with the automaton obtained by instantiation, for every event $e_i, 0 \leq i \leq n$, of the parameter $\gamma_i$ by the value $c_i$ of the delay, in relative time, expressed in the given score.

### *7.4.2 Actual performance*

An actual performance by a performer is described by the list of events that have been missed and by the actual durations of the events. More precisely, we define an actual performance by a list $\varepsilon_0, \ldots, \varepsilon_n$, where for all $0 \leq i \leq n$, $\varepsilon_i$ is either $e_i$ or $\overline{e_i}$ (we make no distinction below between the detection of the event $e_i$ or the detection that this event is missing) and a list $s_0, \ldots, s_{n-1}$, where $s_i$ is the actual duration of

event $\varepsilon_i$, i.e. the physical time, expressed in seconds, elapsed between event $\varepsilon_i$ and event $\varepsilon_{i+1}$ (the event $\varepsilon_n$ marks the end of performance, hence its duration does not matter). The onsets of the events $\varepsilon_0, \ldots, \varepsilon_n$ in the performance are defined by $t_0 = 0$ and $t_i = \sum_{j=0}^{i-1} s_i$ for all $0 < i \leq n$. The duration $s_i$ might differ from the expected one, as specified in the score by a duration $c_i$ *relative* to the tempo (number of beats). This reflects a change of tempo by the player during his performance. At the detection of the event $e_i$, for $0 < i \leq n$, or detection that it is missing, The *Antescofo* listening machine infers from the subsequences $\varepsilon_0, \ldots, \varepsilon_i$ and $s_0, \ldots, s_{i-1}$ an estimated value $T_i$ of the tempo (number of beats by seconds) currently followed by the performer (see Section 3). The inferred tempo is immediately broadcasted to the whole system and used by the groups to compute the delays in physical time from the delays specified in relative time in the score. The difficulty in the conversion of the delays of actions from relative into physical time is that when a new event arrives before the expiration of a delay, then the remaining delay has to be readjusted to the newly inferred tempo.

In our timed automaton model, we can overcome this difficulty by considering only delays in relative time, and assuming that all the clocks flow at the same speed, which is defined by the estimated tempo values. More precisely, we define a global clock $\theta$, which is a piecewise linear function from the physical time in seconds into the relative time in beats, such that $\theta(0) = 0$ and with a slope $T_i$ between time $t_i$ and $t_{i+1}$ for all $0 \leq i < n$. Note that this definition assumes that the tempo remains stable between any two consecutive events $e_i$ and $e_{i+1}$, which is an implementation choice of *Antescofo* (the tempo is changed only when something new is detected).

Given the two above sequences of actual durations in seconds and tempi, we can define the relative duration of event $e_i$ in the actual performance as $c'_i = s_i \times T_i$.

The behavior for the actual performance is defined by the semantics of the synchronized product automaton, executed following the global clock $\theta$, and with the instantiation of every parameter $\gamma_i$ by $c'_i$.

*Example 9* The parallel composition of these automata is represented in the Fig. 25. The timing of one of the possible performance is illustrated in the Fig. 26. It corresponds to the bold path of the Fig. 25. The scenario is as follows. The performer plays the first event; the $e_0$ transition is taken and trigger $g_{11}$ to prepare $g_2$ and $a_{11}$; 0.2 beat latter, the clock associated to $g_2$ is expired and $g_2$ is launched to prepare $a_2$. The second event $e_1$ occurs "before expected". The transition $e_1$ is taken, triggers the transition $g_{12}$ and forces the transition $a_{11}$. The transitions corresponding to the actions $a_2$ and $a_{12}$ are taken when their respective clocks expire.

### 7.5 Uniqueness of traces

The timed automaton takes as inputs the performance of musicians (notes and delay) and produces deterministically output groups of electronic actions as specified in the augmented score. As a matter of fact, we can observe that once we have instantiated all the parameters of the synchronized product automaton, either with the relative delays of the score as in Section 7.4.1, or with the relative delays corresponding to an actual performance in Section 7.4.2, we obtain a timed automaton with the expected behavior: there exists a unique accepted timed word, modulo permutations of simultaneous letters in the timed word. This means that the system modeled has

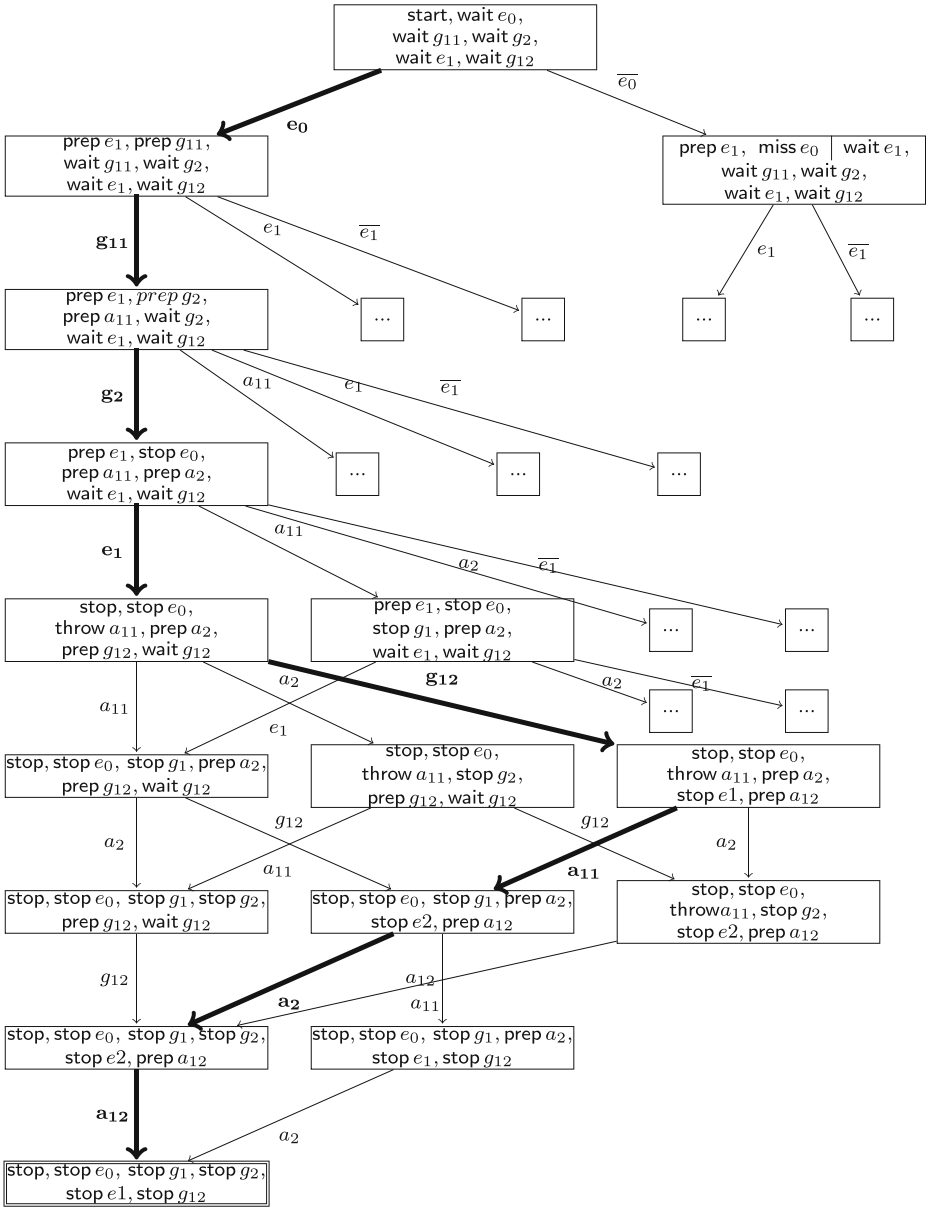



**Fig. 25** Example: synchronized product of the automata corresponding to the running example

a deterministic behavior: a fixed performance by the musician will always induce the same reaction of the system.

Indeed, all the transitions of the automata of the network parse(*score*) have one of the following form:

(i) a transition without guard (the guard equals *true*) and from a location without invariant.

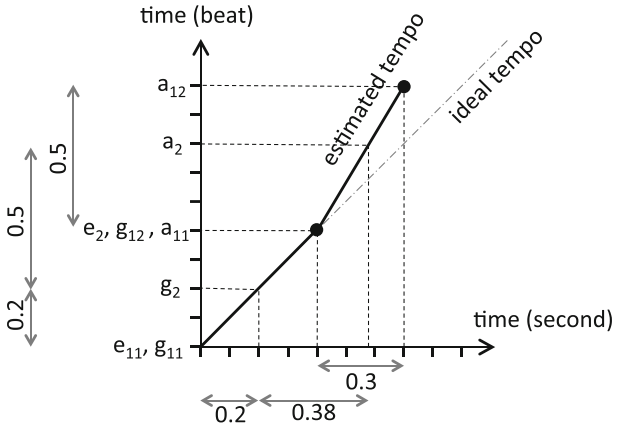



**Fig. 26** Example: this graph represents the matching between the physical and the relative time for an execution of the bold path of the synchronized product in Fig. 25

(ii) a transition with a guard of the form $x = d$ or $y = d$ and from a location with an invariant $x \leq d$ or $y \leq d$. Moreover, the clock $x$ or $y$ is reset by the transition.
(iii) a transition with a guard of the form $y < d$ and from a location with an invariant $y \leq d$, and which resets the clock $y$ (for the tight groups in Section 7.2.2, Figs. 14, 15, 16, 17).

According to the definition of the synchronized product in Section 4, every transition of type (i) or (iii) will be synchronized with a transition of type (ii). It follows that in the synchronized product, every transition will have a guard of the form $x = d$ or $y = d$, hence, in a run, each transition will be fired at a (relative) time which depend only on the automaton. More precisely, let us consider the following run of the automaton,

$$\rho = \langle \ell_0, v_0 \rangle \xrightarrow{d_1,s_1} \langle \ell_1, v_1 \rangle \xrightarrow{d_2,s_2} \dots \xrightarrow{d_{k-1},s_{k-1}} \langle \ell_{k-1}, v_{k-1} \rangle \xrightarrow{d_k,s_k} \langle \ell_k, v_k \rangle$$

where $v_0$ is the null valuation, if there are $p$ transitions outgoing of the location $\ell_{k-1}$, with respective symbols $s_1,\dots, s_j$ and guards $x = d_0$, $y_1 = d_1,\dots, y_p = d_p$, then obviously, $v_k = v_{k-1} + d_i$ for some $1 \leq i \leq p$.

It may occur that several of the $p$ above transitions, say the transitions labeled by $s_{i_1},\dots, s_{i_j}$, can be fired at the same time in the state $\langle \ell_{k-1}, v_{k-1} \rangle$, because the delays $d_{i_1},\dots,d_{i_j}$ specified by the corresponding guards expire in the same time. In this case, it is easy to see that all the permutation of the simultaneous times letters $\langle s_{i_1}, t_k \rangle,\dots,$ $\langle s_{i_j}, t_k \rangle$ $(t_k = \sum_{i=1}^{k} d_i)$ is possible at position $k$ in a timed word accepted by a run starting with the above $\rho$.

## 8 Conclusion

In this paper we presented a novel formal model of *Antescofo*[3] an automatic accompaniment system. The architecture of the system couples two modules: a listening machine which recognizes in real time the events described by a score with

[3] http://repmus.ircam.fr/antescofo.

a reactive system which coordinates and schedules in real time the accompaniment actions. The musical requirements on the coordination of the instrumental voices with the electronic voices are described in Sections 2 and 3. The coordination dedicated language allows a composer to specify the musician–computer interactions in an augmented score, indicating the events to recognize and the actions to launch. A unique characteristic of the *Antescofo* augmented score is the ability to synchronize actions in physical time or relatively to the tempo. Furthermore tight synchronizations and error handling strategies give the composer a fine control of the temporal organization.

The design choice of *Antescofo* have been validated through numerous applications. Since the first developments in 2007, *Antescofo* has gained a wide audience attracting composers in contemporary music such as Pierre Boulez, Philippe Manoury, Marco Stroppa or Emmanuel Nunes which have used *Antescofo* for the creation of new musical mixed pieces and for their performances.[4]

The modeling of the time relationships described in *Antescofo* relies on *Parametric Timed Automata*. We have proposed in Echeveste et al. (2011) a first denotational semantics of *Antescofo* augmented scores, based on timed traces. This approach makes explicit the computation of the flow of relative time from the tempo estimations at each event. However this computation is cumbersome and obscure the semantics of coordination. In contrast, the approach developed in this paper pushes aside this problem, abstracting it by a clock evolving in the tempo scale. This abstraction should be possible even when mixing relative and physical time, by considering timed automata with independently evolving clocks, as in Akshay et al. (2008). However the estimated tempo cannot be considered as a clock that exhibits a bounded divergence with the physical clock and the investigation of this problem is future work.

The semantics relying on parametric timed automata enjoys an operational style, yet the systematic use of parallel composition and the handling of errors generates automaton with a high number of states and transitions. Nevertheless, this architecture preserves some of the high-level structures of the augmented score. There is also a well defined object associated to the family of possible interpretations as well as a given interpretation.

We chose to define the semantics of continuous group in terms of discretization. A more direct approach will rely on hybrid systems. Such systems have been considered in the framework of synchronous systems (Benveniste et al. 2010) and offer alternatives we want to explore.

A problem which is not explicit in our approach is the "ordering" of simultaneous actions. It may happens that some actions occurs at the same time but at the level of the implementation, they are performed sequentially and the composer may expect some defined ordering (for instance, a light can be turned off only if it has been previously turned on). The idea is that a 0-delay does not preclude some ordering

[4] *Antescofo* is used to recreate mixed pieces—for example, several landmark pieces of Pierre Boulez have been recreated using *Antescofo* such as "…Explosante-Fixe…" or "Repons"—, as well as directly used to write new creations—as "Einspielung" by Emmanuel Nunnes, "La Nuit Gutemberg" an opera by Philippe Manoury or "Re Orso", an opera by Marco Stroppa—and for performances by the Los Angeles Philharmonic Orchestra, the Berliner Philharmoniker, the Orchestre de Radio France, etc.

of the actions. To handle these constraints, the framework presented here can be enriched with a controller used to enforce these constraints, in parallel with the automaton.

The automata based semantics is appropriate for the formal verification of system qualitative and quantitative properties, through model-checking techniques. More interestingly, some approaches, e.g. Alur et al. (1993), permit the inference of constraints on the values for parameters $\gamma_i$ ensuring a requested behaviour (e.g. one particular interleaving). This could be useful to composers in order to check the robustness of their scores to the variations in the performance. A recurring question is the characterization of the range of tempo variation allowed to preserve some ordering of actions. We have started to investigate techniques (André et al. 2009a, b) developed to infer parameters of circuits in this context. Based on the semantics presented here and these techniques, we recently achieved the computation of linear constraints on the parametric delays of events that ensure the same order of events and actions as in the ideal performance (Fanchon 2012). Some static analysis would also be relevant in a musical context, for example, in a non-deterministic musical piece (the augmented score extends to a graph with choice points), one can be interested in the minimal and the maximal duration of the performance (Wilhelm et al. 2008).

Some features of *Antescofo* have not been mentioned on this paper: the possibility to specify a local tempo for a group (lined or not to the estimated tempo), the possibility to specify messages, delays, periods, tempo through functions, expressions and variables. We are also working on extending the *Antescofo* coordination framework in several directions: guarded choices, iteration structures beyond the loop construct, dynamic groups, multiple inputs (e.g., for example to follow simultaneously music and gesture), more expressive patterns to triggers actions, etc. The development and the formalization of these features is underway.

## References

Akshay S, Bollig B, Gastin P, Mukund M, Narayan Kumar K (2008) Distributed timed automata with independently evolving clocks. In: van Breugel F, Chechik M (eds) Proceedings of the 19th international conference on concurrency theory (CONCUR'08). Lecture notes in computer science, vol 5201. Springer, Toronto, pp 82–97. doi:10.1007/978-3-540-85361-9_10

Allen J (1983) Maintaining knowledge about temporal intervals. Commun ACM 26(11):832–843

Alur R, Dill DL (1994) A theory of timed automata. Theor Comput Sci 126:183–235. doi:10.1016/0304-3975(94)90010-8

Alur R, Henzinger TA, Vardi MY (1993) Parametric real-time reasoning. In: Proceedings of the 25th annual ACM symposium on theory of computing, STOC '93. ACM, New York, pp 592–601. doi:10.1145/167088.167242

André É, Chatain Th, De Smet O, Fribourg L, Ruel S (2009a) Synthèse de contraintes temporisées pour une architecture d'automatisation en réseau. In: Lime D, Roux OH (eds) Actes du 7ème

colloque sur la modélisation des systèmes réactifs (MSR'09). Journal Européen des systèmes automatisés, vol 43. Hermès, Nantes, France, pp 1049–1064
André É, Chatain Th, Encrenaz E, Fribourg L (2009b) An inverse method for parametric timed automata. Int J Found Comput Sci 20(5):819–836. doi:10.1142/S0129054109006905
Benveniste A, Caillaud B, Pouzet M (2010) The fundamentals of hybrid systems modelers. In: 49th IEEE international conference on decision and control (CDC). Atlanta, Georgia, USA
Berry G, Gonthier G (1992) The esterel synchronous programming language: design, semantics, implementation. Sci Comput Program 19(2):87–152
Cont A (2008) Antescofo: Anticipatory synchronization and control of interactive parameters in computer music. In: Proceedings of international computer music conference (ICMC). Belfast
Cont A (2010) A coupled duration-focused architecture for realtime music to score alignment. IEEE Trans Pattern Anal Mach Intell 32(6):974–987
Cont A (2011) On the creative use of score following and its impact on research. In: Sound and music computing. Padova, Italy
Dannenberg R, Raphael C (2006) Music score alignment and computer accompaniment. Commun ACM 49(8):38–43
Dannenberg RB (1984) An on-line algorithm for real-time accompaniment. In: Proceedings of the international computer music conference (ICMC), pp 193–198
Desainte-Catherine M, Allombert A (2005) Interactive scores: a model for specifying temporal relations between interactive and static events. JNMR 34(4):361–374
Echeveste J, Cont A, Giavitto J-L, Jacquemard F (2011) Formalisation des relations temporelles entre une partition et une performance musicale dans un contexte d'accompagnement automatique Actes du 8eme Colloque sur la Modélisation des Systèmes Réactifs (MSR'11) Craye, Étienne and Gamatié, Abdoulaye Editors, vol. 45. Hermè, Journal Européen des Systèes Automatisé, pp. 109–124
Fanchon L (2012) Formal analysis of score. Master's thesis, Ecole Centrale, Paris, France. Available at http://articles.ircam.fr Accessed 19 April 2013
Halbwachs N, Caspi P, Raymond P, Pilaud D (1991) The synchronous data flow programming language lustre. Proc IEEE 79(9):1305–1320
Large E, Jones M (1999) The dynamics of attending: how people track time-varying events. Psychol Rev 106(1):119
LeGuernic P, Gautier T, Le Borgne M, Le Maire C (1991) Programming real-time applications with signal. Proc IEEE 79(9):1321–1336
Mandel L, Plateau F (2008) Interactive programming of reactive systems. In: Proceedings of model-driven high-level programming of embedded systems (SLA++P'08). Electronic notes in computer science. Elsevier, Budapest, pp 44–59
Manoury P (2007) Considérations (toujours actuelles) sur l'état de la musique en temps réel. Etincelle, le journal de la création à l'Ircam. http://etincelle.ircam.fr/prospectives.html Accessed 19 April 2013
Pouzet M (2006) Lucid synchrone, version 3. Tutorial and reference manual Université Paris-Sud, LRI
Puckette M (1991) Combining event and signal processing in the max graphical programming environment. Comput Music J 15:68–77
Raphael C (2011) The informatics philharmonic. Commun ACM 54:87–93. doi:10.1145/1897852.1897875
Vercoe B (1984) The synthetic performer in the context of live performance. In: Proceedings of the ICMC, pp 199–200
Wilhelm R, Engblom J, Ermedahl A, Holsti N, Thesing S, Whalley D, Bernat G, Ferdinand C, Heckmann R, Mitra T, Mueller F, Puaut I, Puschner P, Staschulat J, Stenström P (2008) The worst-case execution-time problem—overview of methods and survey of tools. ACM Trans Embed Comput Syst 7:36:1–36:53. doi:10.1145/1347375.1347389

**José Echeveste** is a PhD student since September 2011 at the University of Paris 6 and IRCAM, working on the *Antescofo* coordination language. Prior to this he completed his masters on "Synchronous Strategies for Real-Time Music Accompaniment Systems" within ATIAM Masters at IRCAM. His research interests include interactive computer music, computer-aided composition, domain specific language for music, synchronous languages and timed automata.

**Arshia Cont** is a researcher at IRCAM and the head of the MuSync scientific team and project, a project common to IRCAM, the CNRS, and INRIA for the study and development of synchronous processes in real-time for computer music. Cont is the creator of the *Antescofo* software program. His PhD thesis jointly awarded by University of California San Diego (UCSD) and the University of Paris VI (UPMC) in 2008 won the French Academy of Science's SPECIF/Gilles Kahn award in 2009. In 2012, he was named director of the Research/Creation Interfaces department.

**Jean-Louis Giavitto** director of research at the CNRS, works on non-conventional programming models, the use of temporal and spatial notions in programming languages, and the simulation of dynamic systems with dynamic structures. He joined the IRCAM team in 2011 where he works on writing for time and interaction in computer music, after applying his research to massively parallel computation and to synthetic and systems biology.

**Florent Jacquemard** receives his PhD (1996) on rewriting and tree automata. After a post doc at SRI and at the Max Planck Institute in Sarbrück, he joined INRIA Loraine. He spend two years at Trusted Logics, working on verification on embedded systems. From 2002, he worked at LSV on the symbolic verification of security protocols and on the typing of XML. Florent Jacquemard joined the MuSync/Antescofo project in 2012.