Formal Analysis of Scores

Léa Fanchon

Team: RepMus MuSync

Tutor from IRCAM: Mr. Florent Jacquemard
Tutor from Centrale: Mr. Gregory Gournay

November 10, 2012

Abstract

The subject of this internship was to add a static-analysis module to the score-following system Antescofo. The aim of this static-analysis module is to forecast the temporal behavior of the system during live performances, providing a substantial help to both composers and interprets. We follow an approach similar to the Inverse Method implemented by Etienne André in the tool IMITATOR. This method permits to infer constraints on the timing bounds (parametric delays) in timed automata guarantying the same execution trace as for given reference values for the delays. In our case, the parameters should represent the tempo of the musician, and the constraints should restrict the tempo variations, indicating the degree of freedom in interpretation guarantying the expected realtime behavior of the system. For musicality reasons, we found a way to relax the notion of same execution trace to some alternatives.

Le sujet de ce stage était d'ajouter un module d'analyse statique au système de suivi de partitions Antescofo. Le but de ce module d'analyse statique est de prévoir le comportement temporel du système durant les performances (concerts), offrant ainsi une aide substantielle aux compositeurs ainsi qu'aux interprètes. Nous utilisons une approche analogue à celle développée par Etienne André dans l'outil IMITATOR. Cette méthode (Inverse Method) permet l'inférence de contraintes sur les bornes des gardes et des invariants d'un automate temporisé paramétré garantissant un même comportement que pour une instantiation donnée des paramètres. Dans notre cas, les paramètres représentent les délais de jeu du musicien, et les contraintes ont pour but de restreindre les variations de tempo, indiquant le degré de liberté autorisé dans l'interprétation du musicien pour lequel le comportement du système en temps réel reste satisfaisant. La notion de comportement acceptable doit être définie par rapport à des exigences de musicalité.

Contents

List of Figures

Chapter 1

Introduction

Embedded systems are expected to be powerful, performant, and reliable, as they tend to have an important impact of our lives. It means that they have to satisfy many quantitative constraints, in particular timing constraints (response time, propagation delays...). Moreover, they generally interact with the physical environment in which they operate (such system are called reactive systems); this interaction has to be regulated by constraints as well.

As part of the effort that has been made for the development of reliable embedded systems, several verification approaches have been developed in the past years for checking that embedded systems behave as expected. One can cite for instance the model-checking defined in the late 1970's, which permits to verify qualitative properties of a system by an exhaustive exploration of its reachable states. This technique has been applied successfully on many industrial cases.

In general-purpose software, the time it takes to perform a task is an issue of performance, not correctness (it is not incorrect to take longer to perform a task such as sorting a list of integers). For time critical software however, in particular open systems interacting with a user or an environment, the problem is quite different, as the time it takes to perform a task may be critical to the correct functioning of the system. Consider for instance the case of systems embedded in public transportation appliances (Siemens Metro Automation): in many situations, events must be triggered by the system at the right time, not too late but also not too early. In the early nineties, a great step has been done towards the development of verification techniques for time critical software, with the development of timed automata by Rajeev Alur and David Dill [14] [15]. The framework of timed automata extends classical finite-state automata with real-time constraints. The availability of transitions in the resulting timed automata depends on time, so that the structure of the automaton evolves during the execution. This provides an important and useful gain in expressiveness, while preserving decidability of several model-checking-related problems. These timed automata can be composed into networks of automata, which proves very useful for the modeling of complex systems.

In timed automata, the real-time constraints refer to fixed bounds delays (expressed in seconds). The behavior of the model is generally very sensitive to the values of these bounds, and in some cases it is rather difficult to find their correct values. For instance in the case of circuit verification, some bounds cannot be fixed precisely, like the stabilization time of some components. An alternative approach is to reason parametrically, by considering some bounds are unknown constants (parameters) [16] and try to synthesize a constraint on these parameters [19], in order to ensure a correct behavior. This is the idea behind parametric timed automata

[15].

Interactive Music Systems [18] permit the use of computers as musical tools, in live interaction with real musician. They are therefore significant examples of time critical systems, with a strong interaction with an environment. Timing constraints is a general problem in music performance.

In a music score, the duration of each note destined to be played by the musician is exactly specified. Despite this, it is well known that two human interpretations of the same music score can differ significantly, particularly regarding temporal aspects. In some cases, for instance when a *rubato* tempo is adequate, the musician is even encouraged to get off the beaten track and frankly vary from the durations specified on the score. However, despite the temporal variability, several musicians playing together usually manage to make the music sound "as expected", using various synchronization strategies, most of which instinctive. In the case of mixed instrumental/electronic music [1], where a computer has to play together with musicians in realtime, the synchronization strategies have to be formally specified. In that case, the system can be considered like a reactive system interacting with an unpredictable environment (the musician).

This internship took place within the team in charge of the score-following system Antescofo [2] [3]. The main function of Antescofo is to synchronize the automatic accompaniment and the musician during mixed instrumental/electronic music concerts. It is capable of following the performance of the musician in realtime within a given score, of decoding the current tempo of the musician, and of synchronizing this human performance with the computer realized elements specified on the score. The music scores currently used in Antescofo consist essentially in the specification of sequences of musical events destined to be played by the musician and detected by the system, and of electronic actions to be launched by the computer in response. The language used for writing Antescofo scores [4] offers various synchronization strategies between events and actions and various error handling strategies. These specifications can be seen as a special kind of timed automata where the delays between an event and an action or between two successive actions are expressed relatively to the recognized tempo, which changes at each element detected.

Antescofo intends to expand the paradigm of synchronization and score following and tends towards a tool for writing time and interaction in computer music, for both composition and performance. This mission raises many issues, among which the specification of a language for writing mixed-music scores adapted to the rapidly evolving needs of composers [4], and the management of the expression of delays and durations in different time scales.

The aim of this internship was to create a tool for analysing and forecasting the temporal behavior of written mixed-music scores before a musical performance, in order to assist interprets in the preparation of concerts, and composers in the specification of the interactions between the musician's part and the automatic accompaniment. Our method was to model a score with a parametric timed automata network, where the parameters are the delays of the musician for each note of the score, and infer a constraint on the parametric bounds of this network guarantying an acceptable behavior of the system during concerts.

In this report, we first present the system Antescofo (Chapter 2), and more precisely the language used in Antescofo for writing mixed-music scores. The synchronization strategies aimed at dealing with the different time scales of music interpretation will be exposed, along with the error-catching strategies. Then, (Chapter 3) we introduce the subject and the aim of the internship, and the method we used to reach our goals. We then (Chapter 4) give a few definitions on the theory of parametric timed automata developped in [7], and present the modeling of a mixed-music score into a network of parametric timed automata. We then expose the method of parameter synthesis chosen for our problem [7] and its adequacy to our case compared to other methods [12]. In the following chapter (Chapter 5) we expose our implementation of the static analysis module. We then describe a theoretical result and an algorithm aimed at improving the module by allowing the musician more interpretation freedom; we also introduce the implementation of a lighter version of this result. Finally, (Chapter 6) we expose a way of providing help in a Computer Assisted Composition context, when the composer is still at the earlier stage of creating the score.

Chapter 2

A score-following system called Antescofo

2.1 Presentation of Antescofo

As mentioned in the introduction, Antescofo detects the tempo of the musician and the current position on the score during live performances and adapts the automatic accompaniment accordingly. More precisely, Antescofo allows for automatic recognition of music score position and tempo from a realtime audio stream coming from performer(s), making it possible to synchronize an instrumental performance with computer realized elements in real time. The name Antescofo designates the actual synchronization software as well as the synchronous language meant for writing the input scores. This language allows flexible writing of time and interaction in computer music.

2.1.1 General structure of Antescofo

Antescofo is structured in two main parts: a recognition module aimed at aligning the audio stream with the score and calculating the current tempo [6], and a coordination module that reacts to the output of the recognition module and launches the electronic accompaniment.

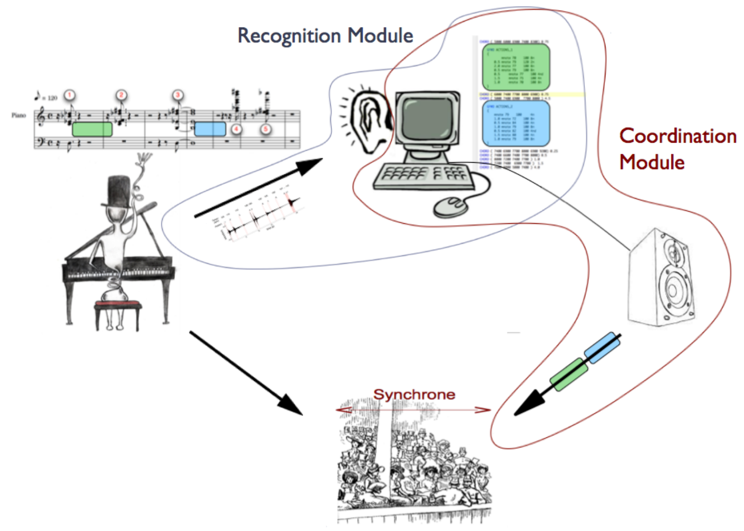


Figure 2.1: General structure of Antescofo

2.2 Quantification of time

2.2.1 Two different time scales

One of the challenges of a score-following software is the coexistence of two disctinct time scales that need to be coordinated:

- *Physical* time scale
  The physical time scale, also called absolute time scale, is the classic time scale which is mesured in seconds.

- *Relative* time scale
  The relative time scale varies in real-time with the tempo of the musician. The time unit used to quantify time in the relative scale is the tempo pulse.

Delays expressed in physical time and delays expressed in relative time may coexist in the same score. However, in the models presented in this document, we consider only delays expressed in relative time. Indeed, as described in section 4.2, we model each score with a network of parametric timed automata with clocks flowing in relative time. We expose in this report a solution for the case when the guards and invariants of this modeling network are linear constraints. As delays in physical time would translate into non-linear constraints, we didn't take them into account.

The validity of a definition of tempo is measured experimentally by its adequacy with the instinctive human perception of tempo. The definition used by Antescofo is thus very sophisticated and complex, in order to fit as closely as possible the reality of music perception by a human ear. The estimation is far from trivial. Indeed, the impact on the current tempo value of an acceleration of the musician is progressive: the Antescofo expression of the tempo pulse at a given location of the score depends from the entire history since the beginning of the piece.

2.2.2 Antescofo tempo evaluation

During a performance, the musician is not likely to stick precisely to the initial tempo. Thus, the delays of the musician's events must be considered as parameters that will be instantiated by the musician during the performance.

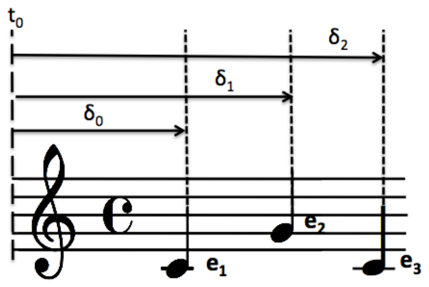


Figure 2.2: Parametric delays of events

The recognition module actualizes the current tempo value at each note played by the musician. Let's use the following notations:

- $(e_i)_{0\leq i\leq n}$: the sequence of notes to be played by the human musician.

- $\forall i \in [|0, n|]$, T_i: tempo value evaluated by the recognition module just after e_i is played.
- θ: Conversion function from durations expressed in physical time scale to durations expressed in relative time scale. θ is piecewise, and θ' is discontinuous at the dates of attack of each e_i.
- $\theta\ _i$': Slope of the affine portion of θ between e_i and e_{i+1}. As mentioned above, the tempo evaluation uses the whole history of the piece since the first note is played. Thus: θ_i' $= f(\delta_0, ..., \delta_{i-1})$.

We call *ideal performance* the performance during which the delays written on the score are exactly respected, meaning that the musician doesn't accelerate or decelerate at all. In that case, there is a linear conversion function between physical and relative durations.

In the following graph, the ideal performance is presented in green, and the actual human performance in black. Here the musician accelerates between e_1 and e_2, and decelerates a bit between e_2 and e_3.

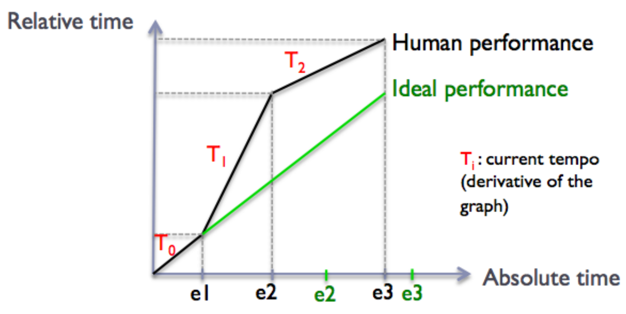


Figure 2.3: Tempo computation

2.3 Antescofo, a domain specific language

As mentioned above, the name Antescofo also designates the language used for writing the scores given in input to the software [5].

2.3.1 A hierarchical structure

The language Antescofo distinguishes the two following types of objects.

- The *events*:
  The notes that are destined to be played by the human musician.
- The *actions*:
  The atomic items that are part of the automatic accompaniment. The accompaniment actions take the form of messages sent to external real-time audio processing system such as MAX/MSP or Pure Data, for sound processing, sound spatialization, control of lightings and visuals and more... The exact nature of these messages is not relevant in the context of our study, and it is sufficient for our modeling purpose to represent the actions by abstract symbols from a finite alphabet.

Each event, action and group is specified together with a delay.

The structure of the language is strongly hierarchical: each action is nested in a group; each group is either nested in a higher group, or attached to one of the events to be played by the musician. Thus in all the document, when we mention the actions nested in a group, some of them can also be groups themselves.

The detection of the triggering event by the recognition module will trigger the launching of the group.

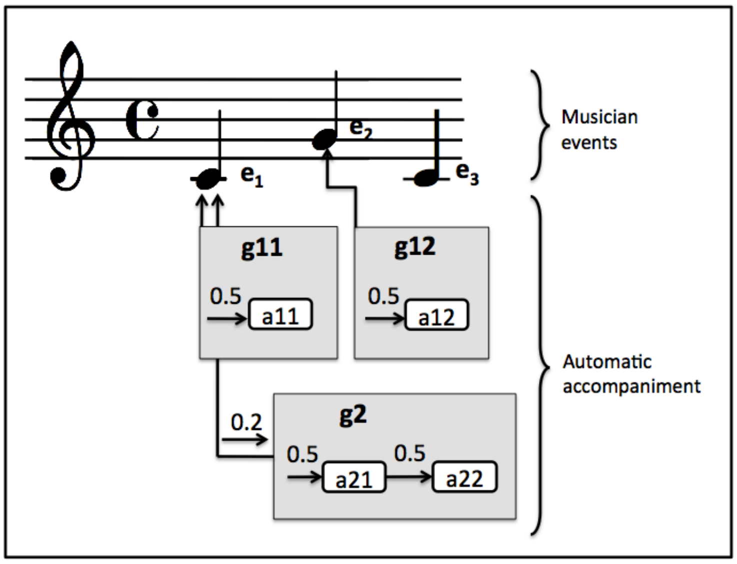
The grammar of the langage is specified by the following:

score := ϵ | event score | (d action) score
event := (e c)
group := group l synchro error (d action)+
action := a | group
synchro := loose | tight
error := local | global

The empty sequence is denoted by ϵ.
An event e denotes an instrumental event and c his duration.
An action a denotes an atomic action. Group structures allow the combination of actions.
A delay $d \in \mathbb{Q}^+$ before an action denotes a tempo relative delay. It is the duration to wait before executing the action.
Each compound action (group) has also a label l.

The attributes synchro and error specify respectively a synchronization strategy and an error handling strategy, as explained in 2.4.

In the following image, the numerical values are the delays of apparition of the corresponding automatic action or group. The code next to the image is the Antescofo implementation of this example.



```
BPM 60

NOTE 60 1.0

0.0 Group11{
  0.5 action11
 }

0.2 Group2{
  0.5 action21
  0.5 action22
 }

NOTE 67 1.0

0.0 Group12{
  0.5 action12
 }

NOTE 60 1.0
```

Figure 2.4: Example of Antescofo score

2.3.2 Evolutions and needs of the language

The language has to keep up with the needs and requests of the composers, and is thus in constant evolution. The short to middle term evolutions are a complexification of the structure of automatic actions and of the hierarchical organization, the introduction of conditional and iterative structures, and the diversification of timing and error-catching strategies.

It will get harder and harder to forecast the behavior of the software before the concert. Thus, a static analysis module becomes necessary.

2.4 Several synchronization and error-catching strategies

2.4.1 Two synchronization strategies

Each group has to be assigned one of the two synchronization strategies: *tight* and *loose*.

- *loose* group
  The group is launched when its triggering event is detected. The relative delays of the actions are calculated with the estimated tempo of the instrumentist.
- *tight* group
  When the group is launched, the system calculates to which event each action nested in the group is associated. These actions will be launched only once the corresponding event is detected.

Intuitively, the tight strategy is meant for the cases when the automatic accompaniment has to rythmically adjust very carefully to the musical line played by the musician. The loose strategy allows more flexibility between the line of the musician and the automatic accompaniment. For instance, if the accompaniment is a harmonic base for the melody played by the musician, then it should be *tight*. If the automatic group imitates the sound of a passing motorcycle (we remind the reader that the software is targeted at contemporary music), then it should be *loose*.

A tight group can be rewritten in a succession of loose groups. Indeed, a tight group is equivalent to several loose groups, each attached to the event occurring immediately before the actions of the loose group. Each loose group contains the actions supposed to be launched between two consecutive events. This rewriting-process is illustrated by the following figure:

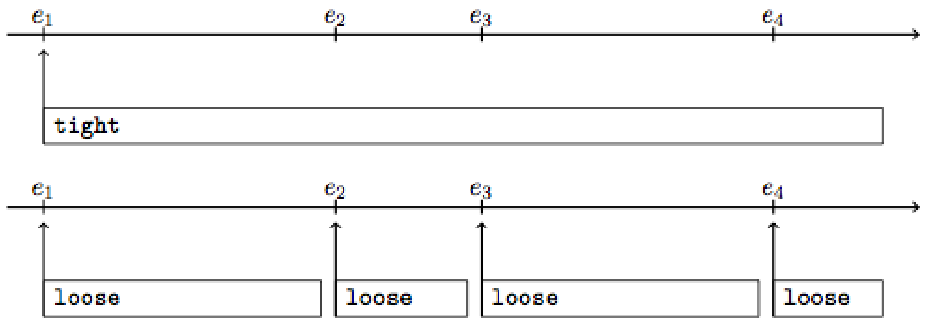


Figure 2.5: Re-writing of a tight group in a succession of loose groups

A tight group nested in a loose group makes no real musical sense. Thus the following rule has been added: tight groups nested in loose groups are turned into loose groups.

2.4.2 Two error-catching strategies

In case of non-detection of one of the events that was supposed to be played by the musician, the automatic groups that were attached to that event, or enclosed in a group attached to that event, are treated differently depending on their error-catching strategy. The two possible error-catching strategies are the following:

- *local* strategy
  If the triggering event is not detected, then the automatic group is not launched.
- *global* strategy
  If the triggering event is not detected, then the automatic group is launched nonetheless, with a launching delay set to zero.

Intuitively, the local strategy is meant for the cases when the function of the automatic accompaniment is to support the musical line played by the musician, for instance in case of an harmonic accompaniment of the melody. If the accompaniment consists in actions which importance is crucial for the good progress of the piece, for instance in case of a soundscape gradually settling, the global strategy is more adapted.

The groups directly attached to an event played by the musician are called *top-level groups*. Consider a non-top-level global group nested in a local group.

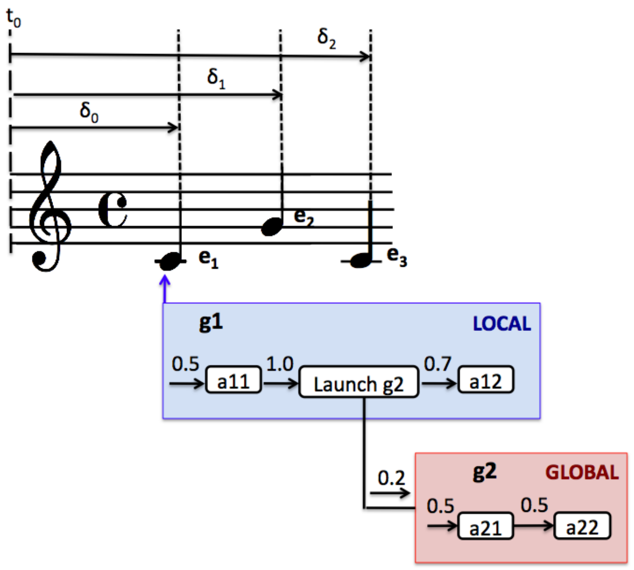


Figure 2.6: Global group nested in a local group

Intuitively, for the composer of the score, the global group is nested in a local group, so it shouldn't be launched if the triggering event e_1 is not detected. Thus the following rule has been added: global groups included in local groups are turned into local groups.

2.4.3 The four possible behaviors

Synchronization and error-catching strategies can be arbitrarily combined. Four behaviors emerge from these combinations.

- Loose-Local
  If the triggering event e_1 of the group is not detected then no action of the group is launched.
- Loose-Global
  If the triggering event e_1 of the group is not detected then then the group is launched with zero delay as soon as the next event e_2 is detected. Within the group, the delays of actions are maintained.

In both loose cases, if there is an error, the whole loose group is affected: either it is translated on the following event or it is simply ignored. At least only an error on the triggering event can affect the behavior of the group.

- Tight-Local:
  If an event associated to the group e_1 is not detected, then the actions triggered by this event are ignored; the actions of the group are launched as planned initially if their position is after the next detected event e_2.
- Tight-Global
  If an event associated to the group e_1 is not detected, then the actions triggered by this event are launched with a zero delay after the next event e_2 is detected. The actions located after the next event e_2 are launched normally.

Unlike the loose strategy, in case of an error from the instrumentist, only the part of the tight group triggered by the missing event will be affected.

Chapter 3

Aim of the internship and method of resolution

3.1 General objective of the internship

3.1.1 An example of problematic situation

Reminder: We call *ideal performance* the performance during which the delays written on the score are exactly respected, meaning that the musician doesn't accelerate or decelerate at all. The order of events, actions and groups in the ideal performance is called *ideal trace*. We call *execution trace* the order of events, actions and groups for a given execution of the piece (possibly with accelerations and decelerations).

Some interpretations of the musician can lead to an execution trace different from the ideal trace, as in the following example:

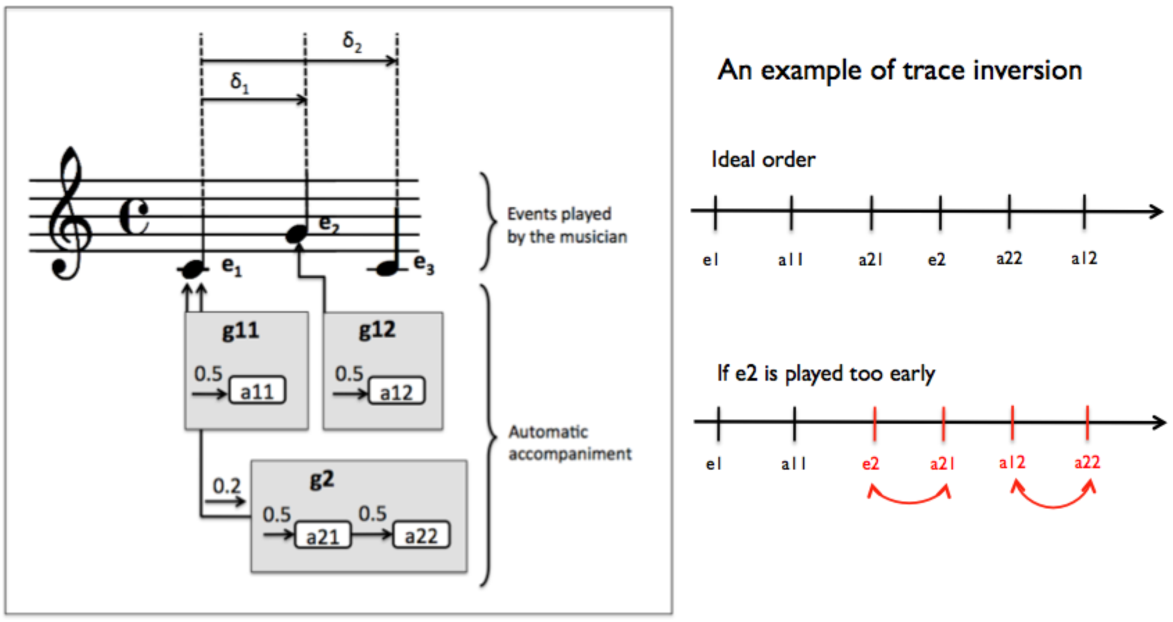


Figure 3.1: An example of problematic execution trace

These situations can be very problematic. For instance, in the image above, if the action a_{22}

consists in turning off the lights and the action a_{12} in turning them back on again, if the order "a_{22} before a_{12}" is not respected, then the lights will be turned off permanently for all the rest of the concert. Hence the necessity of preventing these "bad behaviors" from happening.

3.1.2 Aim of the internship

The aim of this internship is to give the composer and the interpret quantitative indications on the robustness of the composition to the tempo variations of the interpret during the concert. More concretely, the goal is to create a module that takes in input a mixed-music score and returns a constraint on the delays of the musician's events ensuring a correct behavior of the system.

3.1.3 Presentation of the static analysis module

As mentioned above, the function of the static analysis module is to take in input a mixed-music score and return a constraint on the delays of the musician's events ensuring a correct behavior of the system. It is thus necessary to first define "a correct behavior" for a given mixed-music score. It might be a bit too restrictive to consider the ideal trace as the only correct behavior of the system. A better solution is to let the composer define, for each score, a set $L_{acceptable}$ of acceptable execution traces, possibly but not necessarily a singleton containing only the ideal trace, and give this set as an additional input to the module.

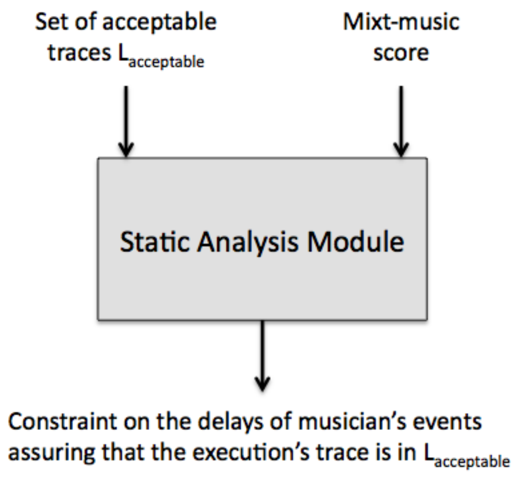

Set of acceptable traces $L_{acceptable}$
Mixt-music score
Static Analysis Module
Constraint on the delays of musician's events assuring that the execution's trace is in $L_{acceptable}$

Figure 3.2: Static analysis module

3.1.4 Workflow

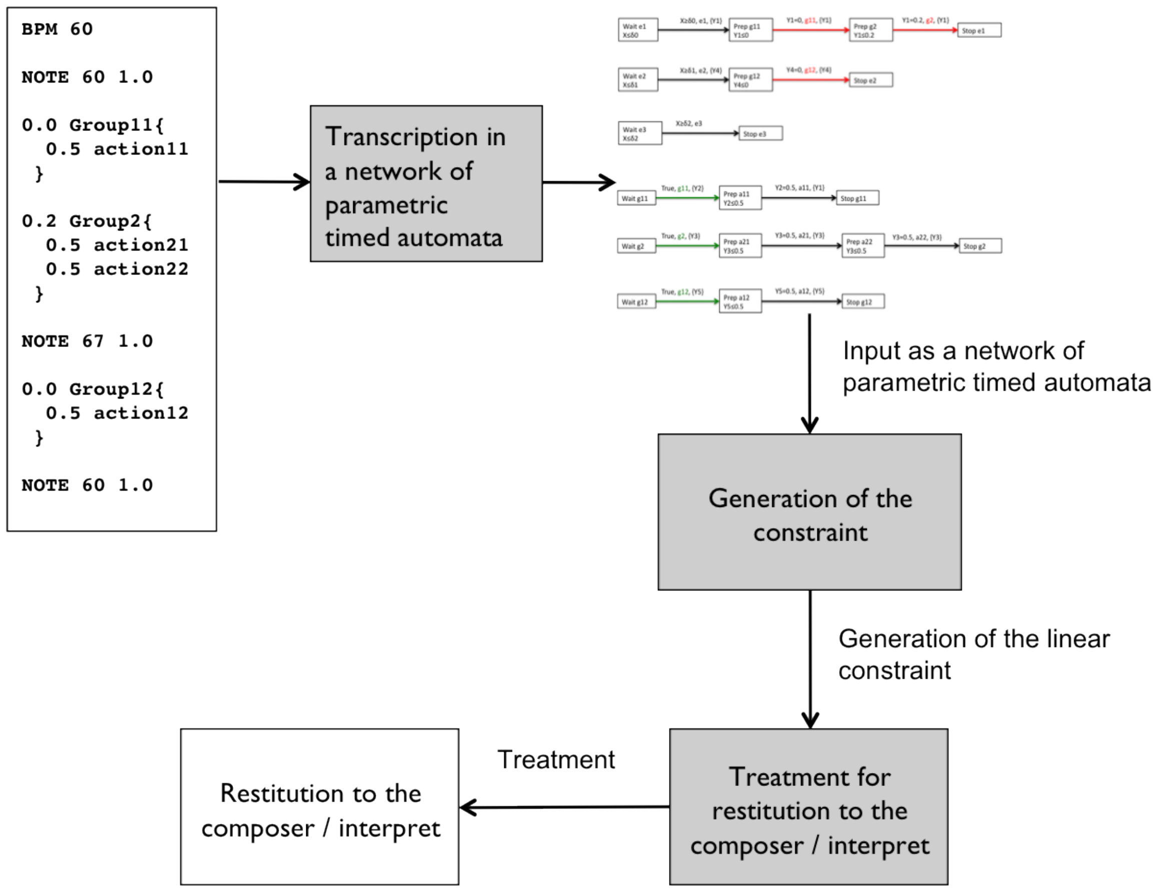
Given a mixed-music score and a set $L_{acceptable}$ of acceptable traces, we want to exhibit a constraint K on the parametric delays δ_i of the player, such that if K is validated by a valuation π of the parameters, then the sequence of events and actions induced by π will be included in $L_{acceptable}$. The generation of this constraint can be broken down to the following steps:

1. Transcription of the Antescofo score into a network of parametric timed automata which parameters are the delays δ_i of the musician's events.

2. Generate a linear constraint on the parameters.

3. Treatment of the constraint to give a proper restitution to the composer or interpret.

In other words, we first need to convert a score written in Antescofo language into a network of PTA. Then, we generate the desired constraint. This constraint has to undergo some transformations so it can be given to the user in a more readily understandable way.



```
BPM 60

NOTE 60 1.0

0.0 Group11{
  0.5 action11
 }

0.2 Group2{
  0.5 action21
  0.5 action22
 }

NOTE 67 1.0

0.0 Group12{
  0.5 action12
 }

NOTE 60 1.0
```

Transcription in a network of parametric timed automata
Input as a network of parametric timed automata
Generation of the constraint
Generation of the linear constraint
Treatment for restitution to the composer / interpret
Treatment
Restitution to the composer / interpret

Figure 3.3: Workflow

We will focus on the two steps of the generation of the constraint: conversion of the Antescofo score into a network of parametric timed automata, and generation of a linear constraint on the parametric delays of the musician's events. The network of parametric timed automata will be given as an input to a constraint generator.

In order to generate a linear constraint on the δ_i, we need to decide which constraint generation method we want to use. We will thus in the following section 3.2 expose our choice for a constraint generation method.

3.2 Inference of linear constraints on the parametric delays

Given a network of parametric timed automata and a behavioral requirement on this network, we need to generate a constraint on the parameters ensuring that the behavioral requirement is fulfilled.

One method has to be chosen among the existing approaches of parameters synthesis. These existing approaches are divided into two main groups: bad-state oriented methods and good-state oriented methods.

3.2.1 Bad-state and good-state oriented methods

In bad-states oriented methods [13], a set of bad states is defined and a constraint is calculated by successive refinement in order to avoid the bad states. Roughly, if it appears that the system can reach a bad state even when respecting the constraint, then the constraint is strengthened in order to exclude that counter-example. When the procedure is over, the constraint assures that all the bad states are excluded.

This principle is close to the CEGAR method, developped in [12], another method that has been often employed within bad state parameter synthesis. CEGAR stands for Counter Example Guided Abstraction Refinement. The principle is the following: for efficiency reasons of the verification procedure, the studied model is over-approximated (the over-approximation contains more accessible states than the exact model). If there is a false positive, meaning a bad state acessible in the over-approximation but not in the exact model, then the approximation is refined to exclude this counter-example.

The good-state oriented approach has been suggested by Laurent Fribourg and Al to converge faster and more often (as the problem of parameters synthesis is undecidable in general) when the set of acceptable traces is small. This method seems more adapted to our problem: in many cases, the set of acceptable traces will be reduced, or almost reduced to the ideal trace. It is thus more intuitive in our case to ask the composer to define the good behaviors instead of the bad ones. The selected tool for our purpose is called Imitator.

3.2.2 Chosen good-state oriented method: IMITATOR

IMITATOR is a software developped by Etienne André, Romain Soulat and Laurent Fribourg at the LSV (Laboratoire de Spécification et Vérification) laboratory. IMITATOR stands for Inverse Method for Inferring Time AbstracT behaviOR.

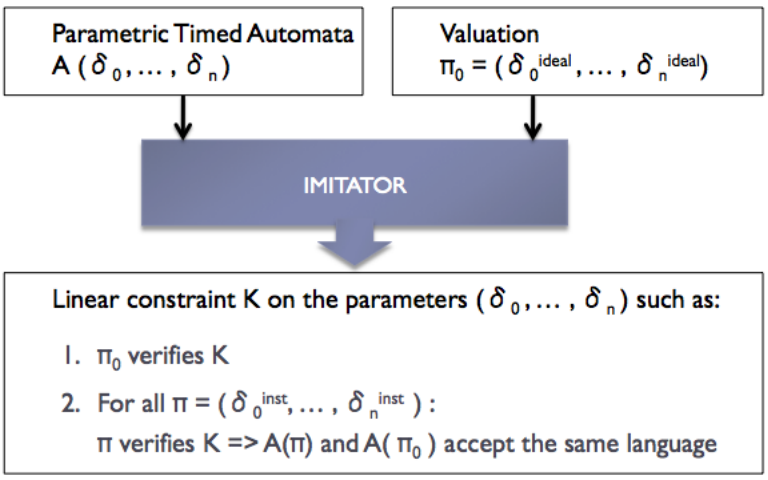

Parametric Timed Automata
A (δ 0, ... , δ n)
Valuation
π0 = (δ 0ideal, ... , δ nideal)
IMITATOR
Linear constraint K on the parameters (δ 0, ... , δ n) such as:
1. π0 verifies K
2. For all π = (δ 0inst, ... , δ ninst) :
π verifies K => A(π) and A(π0) accept the same language

Figure 3.4: Imitator

Given a parametric timed automaton $\mathcal{A}$ and a valuation π_0 of its parameters, IMITATOR returns a linear constraint on $\mathcal{A}$'s parameters such as for every valuation π of $\mathcal{A}$'s parameters, the timed automata $\mathcal{A}[\pi]$ and $\mathcal{A}[\pi_0]$ accept the same language.

Example

Consider the example of the figure 2.4, and suppose we want to limit the acceptable traces to the ideal trace induced by the score. We call respectively δ_0, δ_1 and δ_2 the parametric delays before e_1, e_2 and e_3 (see figure 2.2). Given that e_1, e_2 and e_3 are quarter notes, the ideal valuation of the parameters is $\delta_0^{ideal} = 1$, $\delta_1^{ideal} = 2$ and $\delta_2^{ideal} = 3$. Note that the starting time t_0 of the piece is arbitrary; we chose it here such that $\delta_0^{ideal} = 1$, but it could have been elsewise.

The input given to IMITATOR for this example is the network of parametric timed automata $\mathcal{N}(\delta_0, \delta_1, \delta_2)$ obtained after converting the score (see section 4.2 for the details of this conversion), and the ideal valuation of $\mathcal{N}$'s parameters $\pi_0 = \{\delta_0^{ideal} = 1, \delta_1^{ideal} = 2, \delta_2^{ideal} = 3\}$.

The output generated by the software is the following constraint:

$$K = \delta_0 + 0.7 \leq \delta_1 \wedge \delta_1 \leq \delta_0 + 1.2 \wedge \delta_1 + 0.5 \leq \delta_2$$

If the interpret respects the constraint K during the concert, meaning that the valuation π induced by his or her interpretation validates K, then the order of the events and groups will be exactly the one induced by the score: e1, a11, a21, e2, a22, a12 (...). Therefore, if π validates K, then the problematic situation presented in the figure 3.1 cannot happen.

3.2.3 Specificities in our case

The algorithm is not deterministic.

For a given score S and a given ideal valuation π_0 of parameters, the constraint returned by the algorithm Inverse Method depends on the path chosen during the execution. For the same input, all the output constraints potentially returned by IMITATOR nevertheless define the same set of acceptable traces. For example, for the score of the figure 2.4, the algorithm could return either the constraint $K = \delta_0 + 0.7 < \delta_1 \wedge \delta_1 < \delta_0 + 1.2 \wedge \delta_1 + 0.5 < \delta_2$ or the more stringent constraint $K = \delta_0 + 0.7 < \delta_1 \wedge \delta_1 < \delta_0 + 1.2 \wedge \delta_0 + 1.2 < \delta_2$. Both of these constraints assure that the only accepted trace is the ideal one.

Large number of clocks, states and parameters.

In our model (see section 4.2), the input network given to IMITATOR has a great number of automata, parameters and clocks. Indeed:

- For each group and each event of the score, a new automaton and a new clock are created, in order to define the launching dates of the groups and actions directly linked to that group or event.
- For each event, a new parameter is created.

The scores can contain hundreds of events and thousands of actions or groups. Of course, it is usually possible to subdivide a score into smaller parts in accordance with its musical structure, but the number of automata, clocks and parameters is still much higher than the usual applications of IMITATOR.

Nevertheless, the fixed form of the input is a very simple one: the automata given in input to IMITATOR have a finished size, have no loops and the acceptable behaviors are limited to a single trace. These characteristics ensure that the execution time is not too long, and that the algorithm terminates, which is not true in the general case when the input automaton has not a finished size.

Note that even when the composer wants to allow several execution traces, the PTA network input given to IMITATOR has for acceptable behaviors a single trace. Indeed, our method to allow several execution traces is to modify the model of conversion of the score into a PTA network - see last paragraph of section 5.2. IMITATOR generates a constraint K ensuring that this modified input has for only acceptable trace the ideal trace; the network is modeled so that K also ensures that the initial score has for acceptable traces the traces chosen by the composer.

The time is quantified by clocks that flow in relative time.

We have described in section 2.2 the two time scales, physical and relative. Usually, the clocks of the automata networks given in input to IMITATOR flow in physical time. In our case, with clocks flowing in physical time, the automata networks modeling the scores would have non-linear constraints for guards and invariants, which would considerably complexify the problem. We thus decided to make all clocks flow in relative time. As all clocks flow at the same rate, the algorithm Inverse Method is valid.

Chapter 4

Model

4.1 Definitions

Most of these definitions can be found in [7] and [17].

4.1.1 Constraints on the clocks and parameters

Let us consider a fixed set of clock variables $X = \{x_1, ..., x_H\}$. A clock variable is a variable with values in $\mathbb{R}_+$. All clocks evolve linearly at the same rate. A clock valuation is a function $\omega : X \to \mathbb{R}_+$, assigning a value in $\mathbb{R}_+$ to each clock variable.

$\forall d \in \mathbb{R}_+$, we use the notation $X + d$ for the set $\{x_1 + d, ..., x_H + d\}$.

Let's consider a fixed set of parameters $P = \{p_1, ..., p_M\}$. A parameter valuation π is a function $\pi : P \to \mathbb{R}_+$, assigning a value in $\mathbb{R}_+$ to each parameter.

A linear inequality of the parameters P (resp. on the clock variables X and the parameters P) is an inequality $e \prec e'$, where $\prec \in \{<, \leq\}$, and e, e' are two linear terms of the form $\sum_i \alpha_i p_i + d$, (resp. $\sum_i \alpha_i p_i + \sum_j \beta_j x_j + d$), where $1 \leq i \leq M, 1 \leq j \leq H$ and $\alpha_i, \beta_j, d \in \mathbb{N}$.

A constraint on the parameters P (resp. constraint on the clock variables X and the parameters P) is a conjunction of inequalities on P (resp. on X and P).

Given a parameter valuation π and a constraint C, $C[\pi]$ is the constraint obtained by replacing each parameter p in C with $\pi(p)$. Given a clock valuation ω, $C[\pi][\omega]$ is the expression obtained by replacing each clock x in $C[\pi]$ with $\omega(x)$. A clock valuation ω satisfies a constraint $C[\pi]$ if $C[\pi][\omega]$ evaluates to true. A parameter valuation π satisfies a constraint K if the expression obtained by replacing each parameter p in K with $\pi(p)$ evaluates to true.

Given a constraint C on the clocks and the parameters, the expression $(\exists X : C)$ denotes the constraint on the parameters obtained from C after elimination of the clocks.

4.1.2 Parametric Timed Automata (PTA)

Parametric Timed Automaton

Given a set of clocks X and a set of parameters P, a parametric timed automaton $\mathcal{A}$ is a 7-tuple of the form $\mathcal{A} = (\sum, Q, I, A, K, Inv, T)$, such as:

- $\sum$ is a finite set of actions
- Q is a finite set of locations
- $I \subset Q$ is the set of initial locations
- $A \subset$ Q is the set of accepting locations
- K is a constraint on the parameters P
- *Inv* is the invariant, assigning to every $q \in Q$ a constraint I_q on the clocks and the parameters
- T is the set of transitions of the automaton. A transition is a step relation consisting of elements of the form (q, g, a, ρ, q'), also denoted by $q \xrightarrow{g,a,\rho} q'$, where $(q, q') \in Q$, $a \in \sum$, $\rho \subset X$ is a set of clocks to be reset by the step, and g (the step guard) is a constraint on the clocks and the parameters.

A Timed Automaton (TA) is a particular case of parametric timed automaton, in which $K = \mathit{True}$ and the set of parameters P is empty.

Consider a PTA $\mathcal{A} = (\sum, Q, I, A, K, Inv, T)$.

The notation $\mathcal{A}(K)$ just emphasizes that only the constraint K will change in $\mathcal{A}$.

For every parameter valuation $\pi = (\pi_1, ..., \pi_M)$, $\mathcal{A}[\pi]$ denotes the PTA $\mathcal{A}(K)$ with $K = \bigwedge_{i=1}^{M} p_i = \pi_i$. Thus $\mathcal{A}[\pi]$ is the PTA obtained by substituting all occurrences of a parameter p_i by the constant π_i in invariants and guards. We say that p_i is instantiated with π_i. Note that as all parameters are instantiated, $\mathcal{A}[\pi]$ is a timed automaton.

In the rest of the report, the automata we will introduce have a set of initial locations reduced to a singleton $\{q_0\}$.

(Symbolic) state and step

A symbolic state s of $\mathcal{A}(K)$ is a couple (q, C) where q is a location, and C a constraint on the clocks and parameters.

The initial state of $\mathcal{A}(K)$ is a state s_0 of the form (q_0, C_0), where $C_0 = K \wedge I_{q_0} \wedge \bigwedge_{i=1}^{H-1} x_i = x_{i+1}$. The constraint $\bigwedge_{i=1}^{H-1} x_i = x_{i+1}$ assures that all clocks evolve from the same initial value.

We use the notation $C(X)$ to indicate that X is the set of clocks occuring in C. We use $X' = \rho(X)$, where X' is a renaming of X, to denote the conjunction of equalities $x'_i = 0$ for all $x_i \in \rho$, and $x'_i = x_i$ otherwise. Given a state $s = (q, C)$, a *step* of the automaton from s is defined below:

- $(q, C) \xrightarrow{a} (q', C')$ if $(q, g, a, \rho, q') \in T$ and C' is a constraint on the clocks and the parameters defined by: $C'(X') = (\exists X : (C(X) \wedge g(X) \wedge (X' = \rho(X)) \wedge I_{q'}(X')))$
- $(q, C) \xrightarrow{d} (q', C')$ where d is a new parameter with values in $\mathbb{R}_+$, C' is thus defined by: $C'(X') = (\exists X : (C(X) \wedge (X' = X + d) \wedge I_q(X')))$

- $(q,C) \overset{a}{\Rightarrow} (q',C')$ if $\exists C''$ such that $(q,C) \xrightarrow{a} (q',C'')$ and $(q',C'') \xrightarrow{d} (q',C')$. It can be shown that C' can be put under the form of a constraint on the clocks and the parameters ([1]).

$\forall\ Q\ \subset Q(\mathcal{A}(K)), \forall i \in \mathbb{N}^* : Post^i_{\mathcal{A}(K)}(Q)$ is the set of states reachable from Q in exactly i steps. We use the notation $Post_{\mathcal{A}(K)}(Q) = Post^1_{\mathcal{A}(K)}(Q)$.

We note $S(\mathcal{A}(K))$ the set of steps between the states of the automata $\mathcal{A}(K)$: $S(\mathcal{A}(K)) = \{(q,C) \overset{a}{\Rightarrow} (q',C') \mid (q,q') \in Q(\mathcal{A}(K))^2\}$

$\forall\ Q\ \subset Q(\mathcal{A}(K))$, we also use the notation :
$S_Q(\mathcal{A}(K)) = \{((q,C) \overset{a}{\Rightarrow} (q',C')) \mid q \in Q\ and\ ((q,C) \overset{a}{\Rightarrow} (q',C')) \in S(\mathcal{A}(K))\}$

Symbolic run of a parametric timed automata

A symbolic run of $\mathcal{A}(K)$ of length m is a finite alternating sequence of symbolic states and actions of the form: $s_0 \overset{a_0}{\Rightarrow} s_1 \overset{a_1}{\Rightarrow} ... \overset{a_{m-1}}{\Rightarrow} s_m$ such that $\forall i \in [|0, m-1|], a_i \in \sum$ and $s_i \overset{a_i}{\Rightarrow} s_{i+1}$ is a symbolic step of $\mathcal{A}(K)$ and $s_m = (q_m, C_m)$ is such that q_m is an accepting state of $\mathcal{A}(K)$.

$(a_i)_{i\in[|0,m-1|]}$ is then called a trace of $\mathcal{A}(K)$.

The language $\mathcal{L}(\mathcal{A})$ accepted by a parametric timed automata $\mathcal{A}$ is the set of traces of $\mathcal{A}$.

Given a set L of words composed with letters of $\sum(\mathcal{A})$, L is $\mathcal{A}$-acceptable if all words in L are traces of $\mathcal{A}(K)$.

Network of parametric timed automata (NPTA)

A network of parametric timed automata is a finite set of parametric timed automata $(\mathcal{A}_i)_{\{1\leq i\leq n\}}$, with disjoint sets of parameters, locations and clocks.

Product of automata

Consider a network of n parametric timed automata $(\mathcal{A}_i)_{\{1\leq i\leq n\}}$.

$\mathcal{A} = \prod\limits_{\{1\leq i\leq n\}} \mathcal{A}_i$ is defined by the following characteristics:

- Clocks and parameters: $X = \biguplus_{\{1\leq i\leq n\}} X_i$ and $P = \biguplus_{\{1\leq i\leq n\}} P_i$
- Alphabet: $\sum(\mathcal{A}) = \bigcup\limits_{\{1\leq i\leq n\}} \sum(\mathcal{A}_i)$
- Locations: $Q(\mathcal{A}) = \prod\limits_{\{1\leq i\leq n\}} Q(\mathcal{A}_i)$
- Initial states: $I(\mathcal{A}) = \prod\limits_{\{1\leq i\leq n\}} I(\mathcal{A}_i)$
- Accepting states: $A(\mathcal{A}) = \prod\limits_{\{1\leq i\leq n\}} A(\mathcal{A}_i)$

- Constraint: $K(\mathcal{A}) = \bigwedge_{\{1 \leq i \leq n\}} K(\mathcal{A}_i)$

- Invariant: $\forall\ q = (\prod_{\{1 \leq i \leq n\}} q_i) \ \in\ Q(\mathcal{A}) : I_q = \bigwedge_{\{1 \leq i \leq n\}} I_{q_i}$

- Transitions: $T(\mathcal{A}) = \{\ \ (q_1, ..., q_n) \xrightarrow{g,a,\rho} (q'_1, ..., q'_n)$ such as:
  - $\forall i \in [|1, n|],\ if\ a \in \sum(\mathcal{A}_i)\ then\ \exists(g_i,\ \rho_i)\ such\ as\ q_i \xrightarrow{g_i,\ a,\ \rho_i} q'_i \in T(\mathcal{A}_i)$
  - $\forall i \in [|1, n|],\ if\ a \notin \sum(\mathcal{A}_i)\ then\ q_i = q'_i$
  - $\exists i \in [|1, n|],\ a \in \sum(\mathcal{A}_i)$
  - $g = \bigwedge_{\{i|\ a \in \sum(\mathcal{A}_i)\}}(g_i)$
  - $\rho = \bigcup_{\{i|\ a \in \sum(\mathcal{A}_i)\}}(\rho_i)\ \ \}$

4.2 Transcription of the Antescofo score into a network of PTA

In this section, we present the modeling process of a score into an automata network intended to be given as an input to IMITATOR.

4.2.1 The model

Each group and each event of the score is modeled by an elementary TA. The model also includes a parametric timed automaton to model the launching of the musician's events with delays depending on the performance.

Pre-treatment

Prior to being modeled by a network of PTA, the score is submitted to a small pre-treatment:

- As we explained in the section 2.4, the initial synchronization strategy of a given group can be changed depending on its hierarchical situation: for instance, tight groups included in loose groups are turned into loose groups.

- Flattening of tight groups: tight groups are directly linked to the event immediately preceding them in the ideal order.

We don't enter into the details of this pre-treatment partly aimed at reducing the size of the network.

Modeling of each event and group of the score

Let's consider an event or a group of the score. We call this event or group b_0. We call:

- $\mathcal{A}_{b_0}$ the automata that models b$_0$.

- $L = (b_0^i)_{0 \leq i \leq n}$ the list of groups and actions immediately nested in b_0 and $D = (d_i)_{0 \leq i \leq n}$ their respective launching delays from the moment b_0 is detected.

- D' the list $(d'_i)_{0 \leq i \leq n}$, with $d'_0 = d_0$ and $d'_i = d_i - d_{i-1}$ for $i > 0$

- x_{b_0} the single clock of the automaton $\mathcal{A}_{b_0}$

We suppose that the lists L and D are ordered chronogically.

$\mathcal{A}_{b_0}$ is defined by:

- $\sum = \{\text{Done } b_0\} \bigcup \cup_{0\leq i\leq n}\{\text{Done } B_0^i\}$
- $Q = \{\text{Wait } b_0\} \bigcup \cup_{0\leq i\leq n}\{\text{Wait } b_0^i\} \bigcup \{\text{End } b_0\}$
- $I = \{\text{Wait } b_0\}$
- $A = \{\text{End } b_0\}$
- Inv: For $i \in [|0, \text{n}|]$, $Inv(\text{Wait } b_0^i) = x_{b_0} \leq d'_i$
- $T = \{(\text{Wait } b_0) \xrightarrow{\text{Done } b_0,\ \{x_{b_0}=0\}} (\text{Wait } b_0^0)\}$

  $\bigcup \cup_{0\leq i\leq n-1}\{(\text{Wait } b_0^i) \xrightarrow{x_{b_0}=d'_i,\ \text{Done } b_0^i,\ \{x_{b_0}=0\}} (\text{Wait } b_0^{i+1})\}$

  $\bigcup\{(\text{Wait } b_0^n) \xrightarrow{x_{b_0}=d'_n,\ \text{Done } b_0^n,\ \{x_{b_0}=0\}} (\text{End } b_0)\}$

Modeling of the musician's performance

Let's call $L = (e_0^i)_{0\leq i\leq n}$ the list of the musician's events, and $D = (\delta_i)_{0\leq i\leq n}$ the list of their respective launching delays from the beginning of the piece, that will be instantiated by the musician. The ideal valuation of these delays is the one induced by the score like explained above. We suppose that the lists L and D are ordered is the sense of increasing ideal delays.

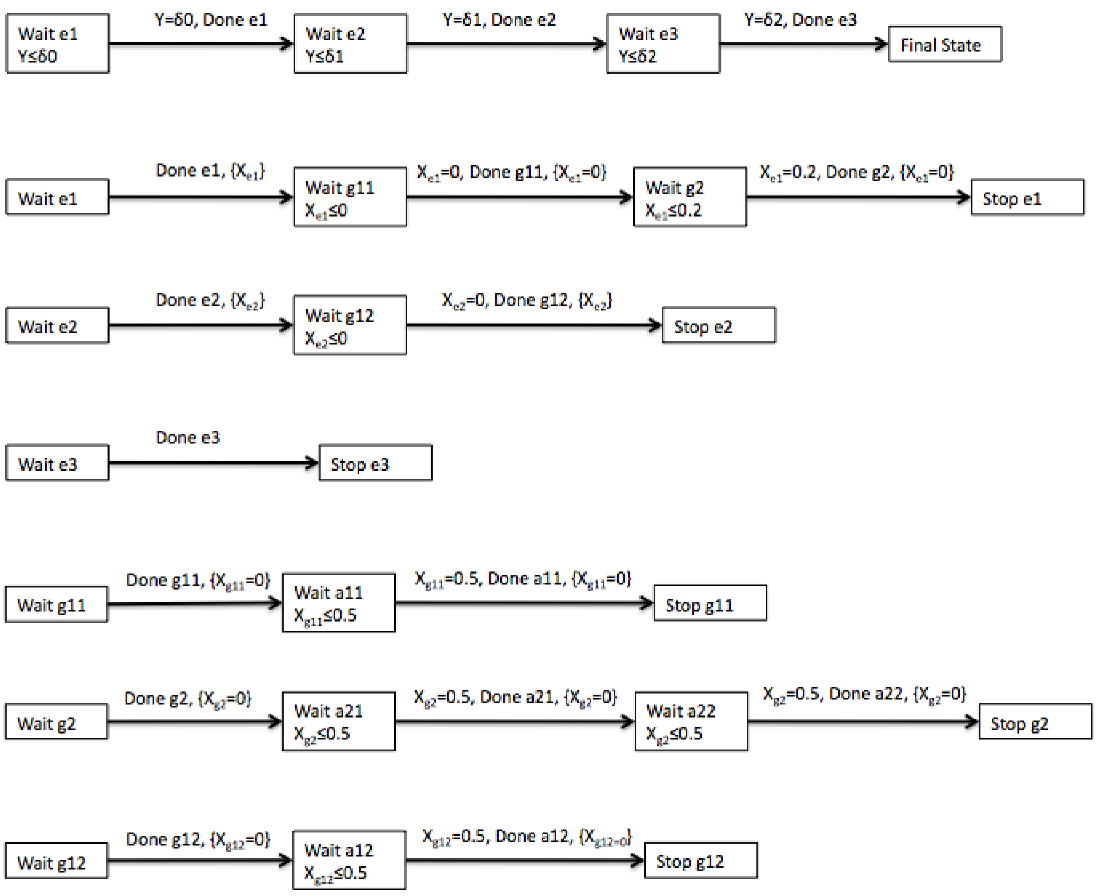
We call y the single clock of the automata $\mathcal{A}_{launcher}$ modeling the musician's performance. $\mathcal{A}_{launcher}$ is defined by:

- $\sum = \cup_{0\leq i\leq n}\{\text{Done } e_i\}$
- $Q = \cup_{0\leq i\leq n}\{\text{Wait } e_i\} \bigcup \{\text{Final state}\}$
- $I = \{\text{Wait } e_0\}$
- $A = \{\text{Final state}\}$
- *Inv*: For $i \in [|0, n|]$, $Inv(\text{Wait } e_i) = y \leq \delta_i$
- $T = \cup_{0\leq i\leq n-1}\{(\text{Wait } e_i) \xrightarrow{Y=\delta_i,\ \text{Done } e_i} (\text{Wait } e_{i+1})\}$

  $\bigcup\{(\text{Wait } e_n) \xrightarrow{y=\delta_n,\ \text{Done } e_n,\ \{y=0\}} (\text{Final state})\}$

4.2.2 Example

The score given as an example in the figure 2.4 will be modeled by the network shown in figure 4.1. In that figure, the first elementary automaton is the events launcher, followed by the three automata that model the three events, and by the three automata that model the three groups of the score. Clocks flow in relative time (tempo pulse unit). For clocks flowing in absolute time, the generated constraints are non-linear.



Wait e1 $Y \leq \delta 0$
$Y=\delta 0$, Done e1
Wait e2 $Y \leq \delta 1$
$Y=\delta 1$, Done e2
Wait e3 $Y \leq \delta 2$
$Y=\delta 2$, Done e3
Final State
Wait e1
Done e1, $\{X_{e1}\}$
Wait g11 $X_{e1} \leq 0$
$X_{e1}=0$, Done g11, $\{X_{e1}=0\}$
Wait g2 $X_{e1} \leq 0.2$
$X_{e1}=0.2$, Done g2, $\{X_{e1}=0\}$
Stop e1
Wait e2
Done e2, $\{X_{e2}\}$
Wait g12 $X_{e2} \leq 0$
$X_{e2}=0$, Done g12, $\{X_{e2}\}$
Stop e2
Wait e3
Done e3
Stop e3
Wait g11
Done g11, $\{X_{g11}=0\}$
Wait a11 $X_{g11} \leq 0.5$
$X_{g11}=0.5$, Done a11, $\{X_{g11}=0\}$
Stop g11
Wait g2
Done g2, $\{X_{g2}=0\}$
Wait a21 $X_{g2} \leq 0.5$
$X_{g2}=0.5$, Done a21, $\{X_{g2}=0\}$
Wait a22 $X_{g2} \leq 0.5$
$X_{g2}=0.5$, Done a22, $\{X_{g2}=0\}$
Stop g2
Wait g12
Done g12, $\{X_{g12}=0\}$
Wait a12 $X_{g12} \leq 0.5$
$X_{g12}=0.5$, Done a12, $\{X_{g12=0}\}$
Stop g12

Figure 4.1: Automata network

The obtained network is printed in HyTech language (see [10] and [11]). This printout can be given as is to IMITATOR, that will compute the network's cartesian product and generate on the fly the desired constraint.

Chapter 5

Implementation and results

5.1 Case of a unique acceptable trace

We consider in this section that the only acceptable trace is the ideal one induced by the score, meaning that the composer doesn't want to allow any inversion in the order of events and actions.

The aim is to obtain a constraint K on the musician's parameters such as, if K is verified, then the only possible execution trace is the ideal trace induced by the score.

5.1.1 Method

As in the previous chapters, $\forall i \in [|0,\ n|]$, we call δ_i the parametric date of launching of the event e_i, depending on the interpretation of the musician, and by δ_i^{ideal} the ideal date induced by the score. Here are the inputs given to IMITATOR's Inverse Method in order to obtain the constraint K allowing only the ideal trace:

- The parametric timed automata network $\mathcal{N}(\delta_0,\ ...,\ \delta_n)$ generated from the score as explained in section 4.2.
- The valuation $\pi_0 = (\delta_0^{ideal},\ ...,\ \delta_n^{ideal})$ of $\mathcal{N}$'s parameters. $\pi_0 = (\delta_0^{ideal},\ ...,\ \delta_n^{ideal})$ is such that the automaton called $\mathcal{A}[\pi_0]$, obtained by computing the synchronized product of the network's automata, accepts only the ideal trace induced from the score.

The output returned by the algorithm will be a linear constraint K on the parameters $(\delta_0,\ ...,\ \delta_n)$ such that:

- π_0 satisfies K
- For each valuation $\pi = (\delta_0^{inst},\ ...,\ \delta_n^{inst})$ of parameters satisfying K: the automaton $\mathcal{A}[\pi]$ accepts only the ideal trace induced from the score.

5.1.2 General Workflow

I needed to code a parser that would create an IMITATOR input out of every Antescofo score. The team had already implemented a code that parsed every Antescofo score into an Abstract Syntax Tree (AST). An AST is a tree representation of the abstract syntactic structure of a source code written in a programming language. Each node of the tree denotes a construct occurring in the source code. A good solution was thus to structure the conversion in three steps:

- Use the existing code to convert the score into an AST
- Read through the AST and create the automata network modeling of the score
- Print this automata network in the form of an IMITATOR input.

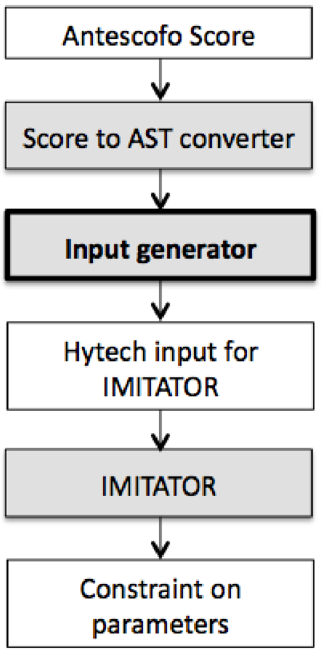


Figure 5.1: General workflow

5.1.3 Functioning of the IMITATOR input generator

The IMITATOR input generator (see workflow image in previous paragraph) reads through the AST and creates the automata network $\mathcal{N}(\delta_0,\ ...,\ \delta_n)$. Then, it prints a text file containing the Hytech version of this network. The language used is C++.

Creation of the automata network

The IMITATOR input generator goes through the AST and creates each automata of the network $\mathcal{N}(\delta_0,\ ...,\ \delta_n)$. We used a visitor design pattern to read through the AST; we thus wrote a visit function for each type of visited node: events, groups, actions, etc. With the visitor pattern, the visit of two nodes of the AST representing the same type of object (for instance two groups) will trigger the execution of the same function, no matter their hierarchical situation (directly linked to an event or nested in another group), their synchronization strategy (tight or loose), and their error-catching strategy (local or global). That function will thus have to check in which hierarchical case the currently visited node is, and what are its synchronization and error-catching strategies. As we explained in the paragraph 2.4.3, the initial synchronization strategy of a given group can be changed depending on its hierarchical situation: for instance, tight groups included in loose groups are turned into loose groups. This also has to be taken into account by the function of visit.

The synchronization strategy has to be taken into account by the code because the generation of the network requires a pre-treatment including the flattening of tight groups (see figure 2.5). The error-catching strategy is also taken into account in prevision of the adaptation of the code to the more general case of several acceptable traces, including those in which some events of the musician are not detected.

The necessity of taking into account all the possible cases has made the code quite complicated and heavy. We thus won't give here the full detail of that code.

Generation of IMITATOR's input file in Hytech language

We need to print in Hytech language the network that has been generated when reading through the AST. As exposed above, this network is constituted of one automaton for each group and event of the score, and of one *Event-Launcher* automaton.

IMITATOR's input consists in:

- The Hytech versions of each automaton of the network printed one after the other in the same text file.
- Another file containing the ideal values of each parameter.

The following example shows the Hytech version of a simple automaton.

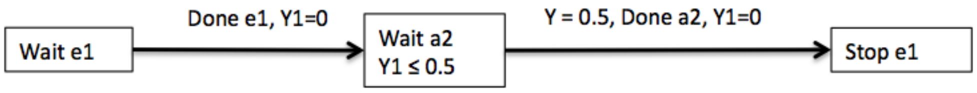
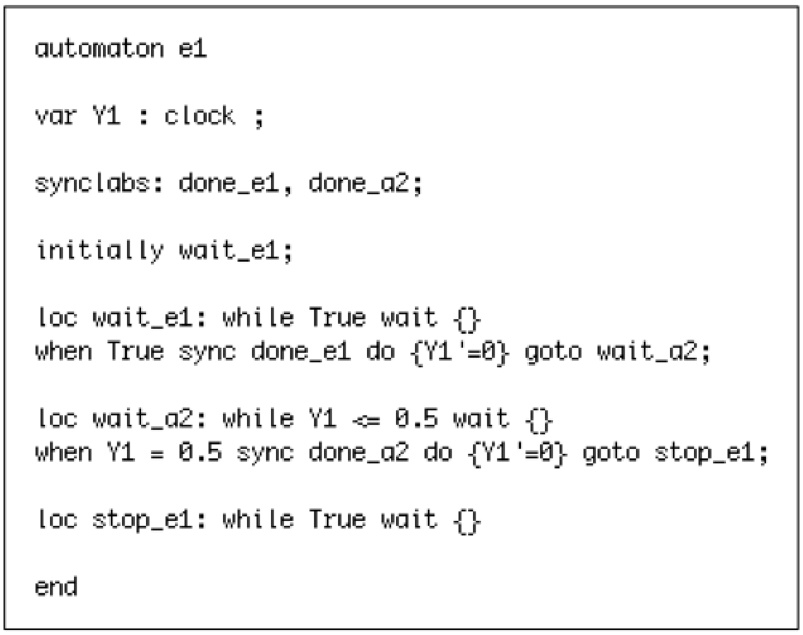


Figure 5.2: Automaton



```
automaton e1

var Y1 : clock ;

synclabs: done_e1, done_a2;

initially wait_e1;

loc wait_e1: while True wait {}
when True sync done_e1 do {Y1'=0} goto wait_a2;

loc wait_a2: while Y1 <= 0.5 wait {}
when Y1 = 0.5 sync done_a2 do {Y1'=0} goto stop_e1;

loc stop_e1: while True wait {}

end
```

Figure 5.3: Print-out in Hytech language

5.1.4 IMITATOR's algorithm

To compute the constraint K, IMITATOR runs the algorithm Inverse Method.

The procedure consists in generating runs starting from the initial state (q_0, C_0), and removing states that are not accessible in the automaton $\mathcal{A}[\pi_0]$ (called π_0-incompatible states) by appropriately refining the current constraint K_0 on the parameters (see [7]).

```
Input:
$\mathcal{A}$ : PTA
$\pi_0$: Reference valuation of $P$.

Output:
$K$: Constraint on parameters.

Variables:
$i$: Current iteration
$S$: Current set of reachable states
$K$: Current constraint on the parameters

$i := 0$; $K :=$ True; $S = \{s_0\}$

DO

    DO UNTIL $S$ is $\pi_0$-compatible

        Select a $\pi_0$-incompatible state $(q, C) \in S$
        Select a $\pi_0$-incompatible inequality $J$ in $(\exists X : C)$ such as $\pi_0$ doesn't
        validate $J$
        $K := K \wedge \neg J$
        $S := \cup_{j=0}^{i} Post_{\mathcal{A}[K]}^{j}(\{s_0\})$

    OD
    IF $Post_{\mathcal{A}[K]}(S) = \emptyset$ THEN RETURN $K := \cap_{(q,C) \in S}(\exists X : C))$
    $i := i + 1$;
    $S := S \cup Post_{\mathcal{A}[K]}$;

OD
```

5.1.5 Example

We give here an example to illustrate the workflow exposed in figure 5.1. Let us consider the following score:

```
bpm 60

NOTE 66 1.0 e1

0.5  Max_a 88 local a1
10.7  Max_b 22 global a3

NOTE 110 2.0 e2

GFWD 1.2 g1 global loose
{ 2.3  Max_b 11 global a11
  2.2  Max_b 11 local a41}
GFWD 2.0 g5 global tight
{ 7.3  Max_b 11 global a51}

NOTE 110 2.0 e3

GFWD 2.8 g6 local tight
{ 2.3  Max_b 11 global a61}
```

Figure 5.4: Input score

This score is given in input to the AST Converter and to the Imitator Input Generator. The output will be the PTA network modeling the score and the ideal valuation of parameters. From the network, we give in the following figures only the event-launcher automaton (figure 5.5) and the automaton modeling the event e_3 (figure 5.6).

```
automaton EventLauncher

var Y : clock ;
  delta0, delta1, delta2: parameter;

synclabs: done_e1, done_e2, done_e3;

initially wait_e1;

loc wait_e1: while Y <= delta0 wait {}
when Y = delta0 sync done_e1 do {} goto wait_e2;

loc wait_e2: while Y <= delta1 wait {}
when Y = delta1 sync done_e2 do {} goto wait_e3;

loc wait_e3: while Y <= delta2 wait {}
when Y = delta2 sync done_e3 do {} goto final_state;

loc final_state: while True wait {}

end
```

Figure 5.5: Event-launcher automaton

```
automaton e3

var Y3 : clock ;
synclabs: done_e3, done_a61, done_a51, done_a3;

initially wait_e3;

loc wait_e3: while True wait {}
when True sync done_e3 do {Y3'=0} goto prep_a61;

loc prep_a61: while Y3 <= 5.1 wait {}
when Y3 = 5.1 sync done_a61 do {Y3'=0} goto prep_a51;

loc prep_a51: while Y3 <= 2.2 wait {}
when Y3 = 2.2 sync done_a51 do {Y3'=0} goto prep_a3;

loc prep_a3: while Y3 <= 0.4 wait {}
when Y3 = 0.4 sync done_a3 do {Y3'=0} goto stop_e3;

loc stop_e3: while True wait {}

end
```

Figure 5.6: Automaton modeling of the event e_3

```
delta0 = 0 &
delta1 = 1 &
delta2 = 3
```

Figure 5.7: Ideal valuation

These two text files (the HyTech version of the PTA network and the best valuation) are given in input to IMITATOR. The output generated is a constraint on parameters and the ideal trace in the form of a graph.

$$\delta_0 \geq 0 \wedge \delta_1 + 3.4 > \delta_2 \wedge \delta_2 + 1.6 > \delta_1 \wedge \delta_1 > \delta_0 + 0.5$$

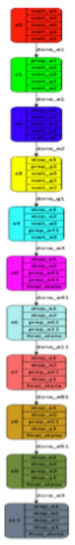


Figure 5.8: Imitator's output

5.2 Case of multiple acceptable traces

5.2.1 Broadening the concept of Acceptable Trace

As Antescofo is meant to be used with a human player, allowing only the ideal execution induced by the trace might be too restrictive. Indeed, if the restriction to the ideal trace presupposes that the musician barely interprets the score and very carefully sticks to the initial tempo, then the software fails its main mission. In order to release this restriction, we need to find a way of generating a constraint authorizing multiple traces, and not only the ideal one.

5.2.2 A regular language as an input

A regular language L is a language accepted by a finite automata.

The theoretical aim would be to allow the composer to give as a set of acceptable traces just any regular language. Indeed, the composer would then be able to allow all the execution traces he/she finds acceptable, and not only the ideal one. Bearing that objective in mind, we will establish the following result:

Theorem 5.2.2.1. *Consider a PTA $\mathcal{A}$ with a single initial location q_0, and a regular language L. We assume moreover that L is closed by prefix and that all the words of L start with the same* *letter. Then, there is a constraint K on the parameters of $\mathcal{A}$ such as: for every valuation π of $\mathcal{A}$'s parameters, if π validates K, then the language accepted by $\mathcal{A}[\pi]$ is included in L.*

The condition *all the words in L start with the same letter* is adapted to our problem, given that all the possible execution traces for a given Antescofo score start with the score's first musician's event.

5.2.3 Some notations

Consider a Parametric Timed Automaton $\mathcal{A}$ and a regular language L.

$\forall\ m\ \in L:$

- The length of the word m is noted $l(m)$.
- $\forall i \in [|1, l(m)|]$, the i^{th} letter of m is denoted by $m(i)$.

We consider another type of automata product called $\tilde{\times}$, with the same definition as the one introduced in section 4.1 except for the transitions. Transitions for $\tilde{\times}$:

Consider $\mathcal{A} = \tilde{\times}_{i=1}^{n} \mathcal{A}_i$.

T($\mathcal{A}$)={ $(q_1, ..., q_n) \xrightarrow{g,l,\rho} (q'_1, ..., q'_n)$ such that:

- $\forall i \in [|1, n|],\ l \in \sum(\mathcal{A}_i)$ *and* $\exists (g_i,\ \rho_i)$ *such as* $q_i \xrightarrow{g_i,\ l,\ \rho_i} q'_i \in T(\mathcal{A}_i)$
- $g = \bigwedge_{\{i|\ l \in \sum(\mathcal{A}_i)\}} (g_i)$
- $\rho = \bigcup_{\{i|\ l \in \sum(\mathcal{A}_i)\}} (\rho_i)$ }

5.2.4 Proof of the result

Consider a finite state automaton $\mathcal{A}_L$ such $\mathcal{L}(\mathcal{A}_L) = L$, considered as a special kind of PTA with an empty set of clocks. As all the words in L start with the same letter, it is possible to chose $\mathcal{A}_L$ so that $I(\mathcal{A}_L)$ is reduced to a singleton $\{q_L^0\}$. The language accepted by $\mathcal{A}_0$ is $\mathcal{L}(\mathcal{A}_0) = \mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{A}_L) = \mathcal{L}(\mathcal{A}) \cap L$

Considering a step $((q, q_L), C) \overset{a}{\Rightarrow} ((q', q'_L), C') \in S(\mathcal{A}_0(K))$, according to the definition of the product $\tilde{\times}$, there is one and only one step $(q, D) \overset{a}{\Rightarrow} (q', D') \in S(\mathcal{A}(K))$.

As $\mathcal{A}_L$ has an empty set of clocks, and according to the definition we gave of a step (section 4.1), if $C = D$, we also have the equality $C' = D'$.

Thus, as the initial states $((q_0, q_L^0), C)$ and (q_0, C) share the same constraint $C = K \wedge I_{q_0} \wedge \bigwedge_{i=1}^{H-1} x_i = x_{i+1}$ where the x_i are the clocks of the automata $\mathcal{A}$, we can conclude that for each step $((q, q_L), C) \overset{a}{\Rightarrow} ((q', q'_L), C') \in S(\mathcal{A}_0(K))$, the corresponding step $(q, D) \overset{a}{\Rightarrow} (q', D') \in S(\mathcal{A}(K))$ is such that $C = D$ and $C' = D'$.

Thus, the following notation makes sense: We note $\tilde{S}(\mathcal{A}_0(K))$ the following set:

$$\tilde{S}(\mathcal{A}_0(K)) = \{(q, C) \overset{a}{\Rightarrow} (q', C')\ \mid\ (q, C) \overset{a}{\Rightarrow} (q', C') \in S(\mathcal{A}(K))\ \text{ and }$$
$$\exists (q_L, q'_L) \in Q(\mathcal{A}_L)^2\ \textit{such as}:\ (((q, q_L), C) \overset{a}{\Rightarrow} ((q', q'_L), C') \in S(\mathcal{A}_0(K))\}$$

We would like to find a constraint K on $\mathcal{A}$'s parameters such as K validates the following condition called $\mathcal{C}$:

For every valuation π of $\mathcal{A}$'s parameters, if π satisfies K, then: $\mathcal{L}(\mathcal{A}[\pi]) \subset \mathcal{L}(\mathcal{A}_0[\pi])$

In the following algorithm, inspired by Inverse Method presented in 5.1, we generate runs starting from the initial state, and we remove the states that are not part of a run which trace is a word included in $\mathcal{L}(\mathcal{A}_0[\pi])$, by appropriately refining the constraint on the parameters. The generation procedure is then restarted until a new bad state is produced, and so on, iteratively until no bad state is generated. The constraint K assures, at the end of the algorithm, that all the runs of $\mathcal{A}(K)$ are included in $\mathcal{L}(\mathcal{A}_0(K))$.

```
$i := 0$; $K :=$ True; $S = \emptyset$; $Q = \{q_0\}$

DO UNTIL $S_Q(\mathcal{A}(K)) \subset \tilde{S}(\mathcal{A}_0(K))$

    Select a state $(q', C') \in Post(Q)$ such that: $\exists a,\ (q, C) \overset{a}{\Rightarrow} (q', C') \notin \tilde{S}(\mathcal{A}_0(K))$
    Select an inequality $J$ of $(\exists X : C')$
    $K := K \wedge \neg J$
    $S := \cup_{j=0}^{i} Post^{j}_{\mathcal{A}(K)}(\{q_0\})$
    $Q := \{q \mid \exists C :\ (q, C) \in S\}$

OD

IF $Post_{\mathcal{A}(K)}(Q) = \emptyset$, THEN RETURN $K$
FI

$i := i + 1$
$S := S \cup Post_{\mathcal{A}(K)}(Q)$
$Q := \{q \mid \exists C :\ (q, C) \in S\}$
```

For all valuation π validating K and $\forall n \in \mathbb{N}^*$, we define the following proposition called $\mathcal{H}_\pi(n)$:

- $\forall m \in \mathcal{L}(\mathcal{A}[\pi])$ such as $l(m) \leq n$,
- $\forall (q_i, C_i)_{0 \leq i \leq l(m)}$ list of states of $\mathcal{A}[\pi]$ such that (q_0, C_0) is the initial state of $\mathcal{A}[\pi]$, and: $\forall i \in [|0, l(m) - 1|],\ (q_i, C_i) \overset{m(i+1)}{\Rightarrow} (q_{i+1}, C_{i+1}) \in S(\mathcal{A}[\pi])$,

Then we have:
$\forall i \in [|0, l(m) - 1|],\ (q_i, C_i) \overset{m(i+1)}{\Rightarrow} (q_{i+1}, C_{i+1}) \in \tilde{S}(\mathcal{A}_0[\pi])$.

Consider a valuation π validating K. In order to show that the constraint K returned by the algorithm above validates the condition $\mathcal{C}$, we prove by recurrence the proposition $\mathcal{H}_\pi(n)$ for all $n \in \mathbb{N}^*$.

- $n = 1$

  $\mathcal{H}_\pi(1)$ translates into: $\forall (q, C) \in Post(\{q_0\}),\ \forall m \in \mathcal{L}(\mathcal{A}[\pi])$ such as l(m)=1:

$S_{\{q_0\}}(\mathcal{A}[\pi]) \subset \tilde{S}_{\{q_0\}}(\mathcal{A}_0[\pi])$

$\mathcal{H}_\pi(1)$ is a direct consequence of the algorithm.

- *Consider $n \in \mathbb{N}^*$. Let us assume $\mathcal{H}_\pi(n)$ is true.*

  Consider :

  - $m \in \mathcal{L}(\mathcal{A}(\pi))$: $l(m) \leq n+1$
  - $(q_i, C_i)_{0 \leq i \leq l(m)}$ list of states of $\mathcal{A}[\pi]$ such that (q_0, C_0) is the initial state of $\mathcal{A}[\pi]$, and $\forall i \in [|0, l(m)-1|]$, $(q_i, C_i) \overset{m(i+1)}{\Rightarrow} (q_{i+1}, C_{i+1}) \in S(\mathcal{A}[\pi])$

  Consider the prefix m' of m containing $l(m)-1$ letters. Then, by $\mathcal{H}_\pi(n)$, we have: $\forall i \in [|0, l(m')-1|], (q_i, C_i) \overset{m'(i+1)}{\Rightarrow} (q_{i+1}, C_{i+1}) \in \tilde{S}(\mathcal{A}_0[\pi])$

  The only thing left to prove is: $((q_{l(m)-1}, C_{l(m)-1}) \overset{m(l(m))}{\Rightarrow} (q_{l(m)}, C_{l(m)})) \in \tilde{S}(\mathcal{A}_0[\pi])$, which is a direct consequence of the algorithm.

Thus, for all π satisfying K, the language accepted by $\mathcal{A}[\pi]$ is included in L.

5.2.5 Implementation in a specific case

The algorithm described above presents a general solution to extend the notion of acceptable trace. However, it can diverge very fast and is too naïve and general to be put into practice within our scope. In this subsection, we offer an intermediary solution between a single ideal trace and an arbitrary regular language L. This solution is a good answer to our problem, and has for advantage that it can be experimented with IMITATOR.

The aim is to allow the composer to select a small zone of the ideal trace which order doesn't matter that much. The set $L_{acceptable}$ is then the set of traces obtained from the ideal trace by changing the order of the actions, groups, and events in the selected zone.

In the following score, the ideal trace is $(e_1,\ e_2,\ a_2,\ a_1,\ a_3,\ e_3,\ e_4)$. If the user selects the zone indicated in red on the image, then the constraint generated by the module will allow all the possible permutations between e_2, a_2, a_1 and a_3.

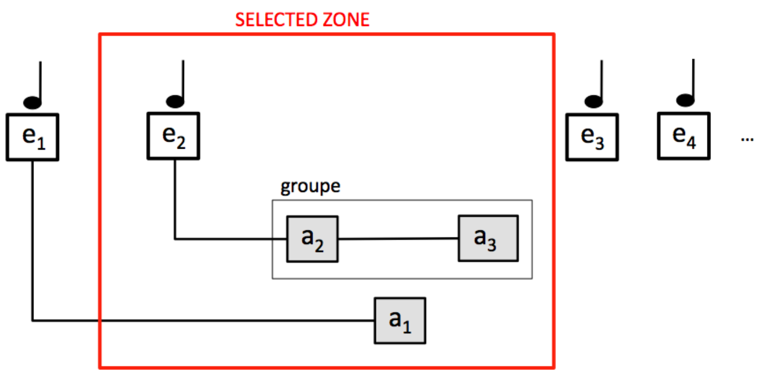


Figure 5.9: Selection of a zone

We made the hypothesis that the events played by the musician are all detected, and in the right order. Thus we restricted the possible choices of zone for the user to the zones containing at most one event (but with an unlimited number of actions and groups). Indeed: if the zone selected by the composer contains two events e_1 and e_2, then the set of acceptable traces will contain traces with e_1 before e_2 and traces with e_2 before e_1, which would mean that an inversion in the order of events is possible and acceptable, and contradicts our hypothesis.

The method we chose was to adapt the form of the PTA network given in input to IMITATOR, so that the constraint generated by IMITATOR allows all the possible permutations of trace in the selected zone. We are not exposing the full details of this adaptation, but the main idea is to simply remove the actions and groups that are inside the zone in the generated PTA network, and directly add boundary constraints of the zone to the initial constraint on parameters that can be optionally given as an input to IMITATOR.

For instance, in the previous example, instead of giving in input to IMITATOR the PTA network corresponding to the score, we give to IMITATOR the PTA network corresponding to the score in which the actions and groups inside the zone have been removed:

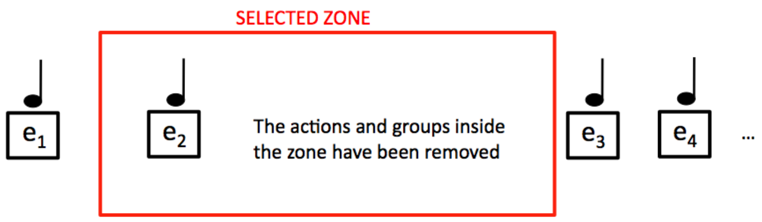


Figure 5.10: Modification of the model

In addition to that, we give to IMITATOR an initial constraint K_{ini} expressing the "boundary constraints" of the zone: here, $K_{ini} = \delta_{e_2} + \Delta_{a_3} < \delta_{e_3} \wedge \delta_{e_1} + \Delta_{a_1} < \delta_{e_3}$, where δ_i is the parametric delay of the event e_i since the beginning of the piece, and Δ_{a_i} is the delay of the action a_i since its triggering event is launched.

Chapter 6

Computer Assisted Composition

The static analysis module presented in Chapter 5 can be used to give information to the musician before the performance, for instance an indication on the allowed accelerations and decelerations at each location of the score. Another way of using the module is to provide help to the composer during the composition process. The composer can be given the same indication as the musician: the allowed accelerations and decelerations at each location of the score; this information would allow him to see, after writing a score, which areas of the score are the most difficult to play for the musician in terms of sticking to the ideal trace, and if the score allows enough space for interpretation. This use of the module takes place after the score composing process.

Another possibility would be to play a role during the score composing process, in order to assist the composer in finding the most appropriate values for some delays in its score, i.e. the values that allow the most freedom of interpretation for the musician.

6.1 A commonly used composition workflow

Each composer usually has his own composition process, his own method, for writing a mixed-music score. According to the accounts of the RIM (Réalisateurs en Informatique Musicale), very often, the composition process consists in first writing the line of events intended for the musician, and fix their delays, before writing the automatic accompaniment. Afterwards is the automatic accompaniment created, first with parametric delays; at that stage of the composition process, the composer has settled on a desired ideal trace for the piece, but has not instantiated the actions' and groups' delays accordingly yet. Then, the third step consists in the valuation of the automatic actions' and groups' delays so that the ideal trace is in the desired order. This third step is not easy for the composer; we offer here a solution to automatize it.

Please note that this composition process is not exclusive; we have heard from the RIMs at IRCAM however that it is the most frequently used process. The RIMs (Réalisateurs en Informatique Musicale) are technical experts in charge of helping the composers with the use of IRCAM technologies in a creative context and are thus quite aware of the composition workflow of each composer they assist.

In all this chapter, for a given score S containing the events $(e_j)_{0\leq j\leq n}$ and the actions $(a_i)_{0\leq i\leq m}$, we note $t_{e_j}^{ideal}$ the ideal delay of the event e_j since the beginning of the piece, t_{e_j} its parametric delay to be instantiated during a performance, and Δ_{a_i} the delay of the action a_i since the launching of its triggering event. We also introduce, for each action a_i, the quantity

$\delta_{a_i} = t^{ideal}_{e_{a_i}} + \Delta_{a_i}$, where e_{a_i} is the triggering event of a_i.

The following example illustrates the three stages of the composition process:

- *First step:* Write the line of the human musician with numeric delays.

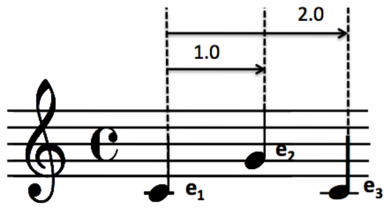


Figure 6.1: First step of the composition process

- *Second step:* Write the automatic accompaniment with parametric delays and decide the ideal order of automatic actions and groups. Note that this ideal order can be expressed in the form of a constraint on actions', groups' and events' delays.

  NOTA BENE: In the following image, the delays of actions since the beginning of the score are noted δ_{a_i}. $\delta_{a_i} = t_{e_{a_i}} + \Delta_{a_i}$, where e_{a_i} is the triggering event of a_i. At the step two of the composition process, the delays of events are instantiated, so $t_{e_{a_i}} = t^{ideal}_{e_{a_i}}$. The parametric part of δ_{a_i} at that stage is thus Δ_{a_i}.

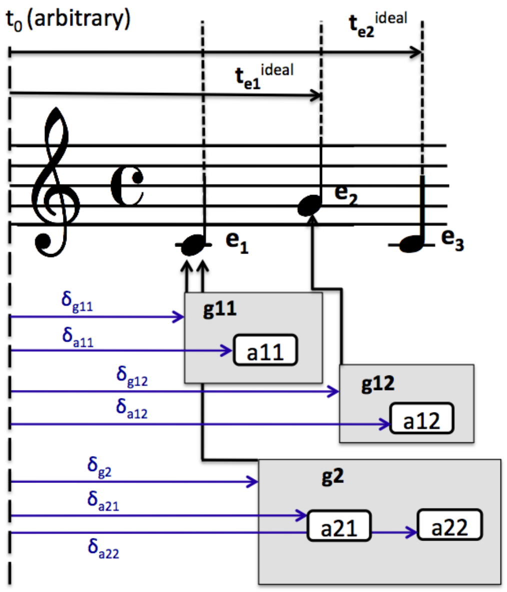


Figure 6.2: Second step of the composition process

- *Third step:* Instantiate the delays of automatic actions and groups.

  Usually, several valuations respecting the ideal trace settled during the second step are possible. The beginning of the piece is called t_0 and can be located arbitrarily anytime before the first event of the piece. We suppose here and in all this section that t_0 coincides with the launching of the first event e_1. Here, one possibility is:

$$\delta_{g_{11}} = 0;\ \delta_{g_2} = 0.2;\ \delta_{a_{11}} = 0.5;\ \delta_{a_{21}} = 0.7;\ \delta_{g_{12}} = 1.0;\ \delta_{a_{22}} = 1.2;\ \delta_{a_{12}} = 1.5$$

Our purpose is to automate the third step. Consider a music score S at the second step of the composition process exposed above: the delays of the events are fixed numerical values, and the delays of the groups/actions are parametric. We want to create an algorithm that takes in input the score S and the constraint K_{ideal} on delays expressing the desired ideal trace, and gives in output a valuation of the actions' parameters respecting this constraint.

For instance, in the example above, the composer would give as an input the result of the step 2, meaning:

- The parametric score written in Antescofo language, at the second step of the composition process - with numerical events' delays and parametric actions' and groups' delays
- A constraint on the actions' and groups' (parametric) delays and on the events' (numerical) delays that reflects the desired order for automatic actions, here:
  $K_{ideal} = \delta_{g_{11}} \leq \delta_{g_2} \leq \delta_{a_{11}} \leq \delta_{a_{21}} \leq t_{e_2} = 1.0 \leq \delta_{g_{12}} \leq \delta_{a_{22}} \leq \delta_{a_{12}} \leq t_{e_3} = 2.0$

The algorithm would return a valuation of the parametric delays of actions and groups (here the quantities noted δ) that respects K_{ideal}. This process is illustrated in the following picture:

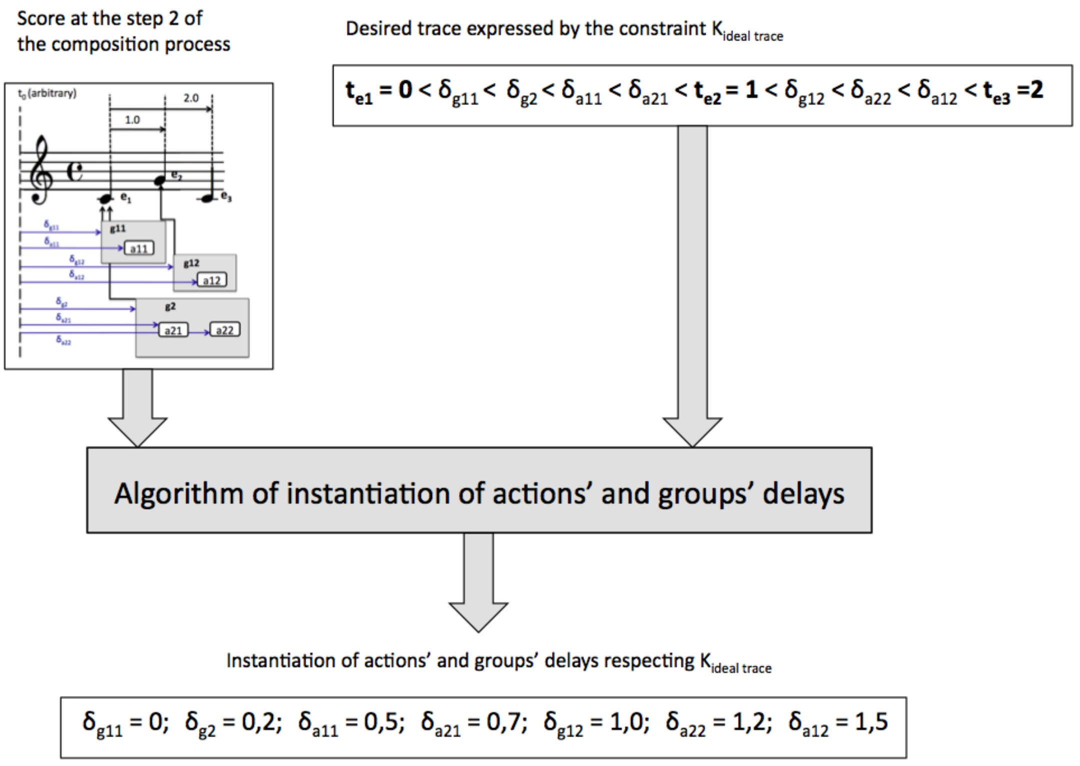

Score at the step 2 of the composition process
t0 (arbitrary)
2.0
1.0
e1
e2
e3
g11
a11
g12
a12
g2
a21
a22
Desired trace expressed by the constraint K ideal trace
$t_{e1} = 0 < \delta_{g11} < \delta_{g2} < \delta_{a11} < \delta_{a21} < t_{e2} = 1 < \delta_{g12} < \delta_{a22} < \delta_{a12} < t_{e3} = 2$
Algorithm of instantiation of actions' and groups' delays
Instantiation of actions' and groups' delays respecting K ideal trace
$\delta_{g11} = 0;\ \delta_{g2} = 0{,}2;\ \delta_{a11} = 0{,}5;\ \delta_{a21} = 0{,}7;\ \delta_{g12} = 1{,}0;\ \delta_{a22} = 1{,}2;\ \delta_{a12} = 1{,}5$

Figure 6.3: Automation of the third step

6.2 Define the optimal valuation in terms of robustness of the score

For a given score at the step 2 of the composition process, and a constraint K_{ideal} specifying the ideal order of actions, groups, and events within that score, it is relatively easy to return one valuation of the parametric actions' and groups' delays satisfying the constraint K_{ideal}. A more interesting idea would be to return the valuation that maximizes the robustness of the score to the tempo variations of the interpret.

Let us first define how we mesure the robustness of the score for a given valuation of actions' parameters.

6.2.1 Preliminary remark

Given the commonly used composition process described above, we have changed, for this chapter, our idea of an input score. Indeed, in the previous chapters of this report, we have supposed that the composer didn't know which ideal trace he wants: we supposed he only knew which numerical instantiations he wants for actions', groups' and events' delays to have in the ideal case. The static analysis module takes in input this entirely specificated score (corresponding to π_0 in the previous chapters), and computes the algorithm Inverse Method that infers the ideal trace from this ideal instantiation.

In the light of the composition process exposed above, we made the new assumption that the composer actually knows already at the second stage what ideal trace he wants. In this context, the static analysis module presented above is not necessary anymore, because the constraint it would generate has a generic form, explicited in the following paragraph.

We call K the constraint expressing the necessity of having the execution trace equal to the ideal trace. K is in the generic form of an ordering of the delays of actions, groups and events counted from the beginning of the score. With the notations defined in section 6.1, these delays can be expressed by the quantities $t_{e_{a_i}} + \Delta_{a_i}$ (for actions) and t_{e_j} (for events). By "ordering", we mean that the form of the constraint will be:

$$v_1 < v_2 < ... < v_N$$

with $\forall k \in [|1, N|]$, $v_k \in \{t_{e_{a_i}} + \Delta_{a_i}\}_{0 \leq i \leq m} \bigcup \{t_{e_j}\}_{0 \leq j \leq n}$, meaning that each v_k is equal to the delay of an action or an event calculated since the beginning of the piece. Let's consider two events or actions b and b' consecutive in the ideal trace. Then $\exists k \in [|1, N-1|]$ such that v_k and v_{k+1} are respectively the delays of b and b'. $v_k < v_{k+1}$ is then the expression of the necessity of having b before b'.

Please note that in this chapter, K denotes the constraint described above with only parametric delays, for actions and groups as well as for events. The constraint that would have been generated by Inverse Method would be the one obtained by substituting in K each action's and group's parametric delay Δ_{a_i} by its numerical value in the score, and thus having the events' delays for only parameters.

6.2.2 Robustness definition

Consider a score S containing n events $(e_j)_{j \in [|0,n|]}$, at the step 2 of the composition process, meaning that events' delays are numerical whereas actions' and groups' delays are parametric.

For each event e_j, we call $t_{e_j}^{ideal}$ the numerical delay of e_j since the beginning of the score. We call K_{ideal} the constraint on the actions' and groups' parametric delays specifying the ideal order of actions, groups, and events during a performance. K_{ideal} is the constraint obtained by substituting the $t_{e_{a_i}}$ by $t_{e_{a_i}}^{ideal}$ in the constraint K presented in the previous subsection.

Let us consider a possible valuation π of the actions' and groups' parametric delays in S, such that π verifies K_{ideal}. We note $S[\pi]$ the score obtained by substituting each action's and group's delay by their value in π. $S[\pi]$ is totally specified, meaning that the composition process has been terminated: all actions', groups' and events' delays are instantiated and have fixed numerical values.

The score is then ready to be interpreted by a human musician during a live performance. As we have seen in the first chapters of this report, the human musician is not likely to respect exactly the numerical values of the delays $t_{e_j}^{ideal}$ specified in the score. Hence the necessity of getting a constraint on tempo variations that makes sure the execution trace will be the ideal one. This constraint, called $K[\pi]$, is the one obtained by substituting, in the constraint K presented in the previous subsection, each action's delay Δ_{a_i} by its numerical value given by π. $K[\pi]$ is thus a constraint on events parameters t_{e_j}.

Here is how we define the robustness of the score $S[\pi]$.

We first define the robustness at the level of one event e_j of the score. Let's call $K[\pi]_j$ the constraint obtained by substituting in the constraint $K[\pi]$ the parametric delays of all the other events with their ideal numerical value:

$\forall k \neq j,\ t_{e_k} := t_{e_k}^{ideal}$

The only parameter occurring in $K[\pi]_j$ is thus t_{e_j}. We note $inf[\pi]_j$ and $sup[\pi]_j$ the lower and upper bounds of the parameter t_{e_j} in the constraint $K[\pi]_j$. The robustness of the event e_j is defined by the numerical quantity:

$robustness(S, \pi, e_j) = min(|t_{e_j}^{ideal} - inf[\pi]_j|, |t_{e_j}^{ideal} - sup[\pi]_j|)$.

The robustness of the score S is defined by the quantity:

$robustness(S, \pi) = min_{j \in [|0,n|]}(robustness_{S[\pi]}(e_j))$.

Intuitively, the smaller the robustness is, the closer to the score the musician must play.

Our aim is to find an algorithm that would provide, at the step 2 of the composition process, a valuation π_0 of actions' and groups' parameters that maximizes the robustness of the score, meaning such that: $robustness(S, \pi_0) = max_\pi(robustness(S, \pi)$. We call optimal such a valuation.

In order to find a good valuation for a given score, a very expensive possibility would be to generate many random valuations of the actions' and groups' parameters, and evaluate the robustness of each of them. The selected solution would be the most robust one among those tried. This procedure is nevertheless quite expensive, given that it requires the generation of many valuations of actions' and groups' parameters, and doesn't guarantee that the selected solution is really the best one.

It is thus worthy to investigate and see if there is a generic algorithm to find a valuation of actions' and groups' delays that universally maximizes the robustness of the score.

6.3 Build the optimal valuation

Consider a performance containing n events $(e_j)_{j \in [|0,n|]}$.

$\forall$ j in $[|0, n|]$, we call $(a_j^i)_{0 \leq i \leq N_j}$ the actions that are supposed to be launched between e_j and e_{j+1} in the ideal trace. The constraint K defined above will thus have the following expression, with only parametric delays:

$K = \bigwedge_{j \in [|0,n-1|]} t_{e_j} < \delta_{a_j^0} < \delta_{a_j^1} < ... < \delta_{a_j^{N_j}} < t_{e_{j+1}}$.

Please note that with the notations introduced in this chapter, the constraint K can also be written:

$K = \bigwedge_{j \in [|0,n-1|]} t_{e_j} < t_{e_{a_j^0}} + \Delta_{a_j^0} < t_{e_{a_j^1}} + \Delta_{a_j^1} < ... < t_{e_{a_j^{N_j}}} + \Delta_{a_j^{N_j}} < t_{e_{j+1}}$

In this section, for more lisibility, for each action a_j^i located between e_j and e_{j+1} in the ideal trace, we will note e_j^i instead of $e_{a_j^i}$ the triggering event of the action a_j^i. Thus K can be re-written:

$K = \bigwedge_{j \in [|0,n-1|]} t_{e_j} < t_{e_j^0} + \Delta_{a_j^0} < t_{e_j^1} + \Delta_{a_j^1} < ... < t_{e_j^{N_j}} + \Delta_{a_j^{N_j}} < t_{e_{j+1}}$

The ideal order expressed by the constraint K is illustrated in the following image:

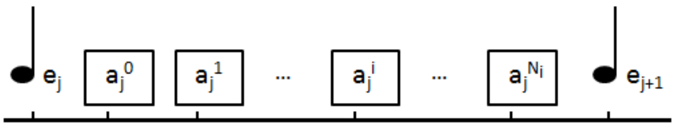


Figure 6.4: Ideal order

The aim of this section is to evaluate for which valuation π of the actions' parameters $(\Delta_{a_j^i})_{j \in [|0,n|],\ i \in [|0,N_j|]}$ the constraint $K[\pi]$ ensures the maximal robustness to the score.

6.3.1 Notations summary

$\forall i \in [|0, \mathrm{N}_j|]$, we use the following notations:

- e_j^i is the triggering event of the action a_j^i
- The parametric delay of each event e since the beginning of the piece is noted t_e.
- The ideal delay of each event e since the beginning of the piece is noted t_e^{ideal}.
- $\Delta_{a_j^i}$ is the expression $\delta_{a_j^i} - t_{e_j^i}^{ideal}$.
- $\theta_{a_j^i}$ is the expression $\delta_{a_j^i} - t_{e_j}^{ideal}$.

- $\alpha_{e_j^i} = t_{e_j^i} - t_{e_j^i}^{ideal}$
- $w_j = t_{e_{j+1}}^{ideal} - t_{e_j}^{ideal}$

For a given valuation π of actions' and groups' parametric delays, we note $\Delta_{a_j^i}[\pi]$ the value given to $\Delta_{a_j^i}$ by π.

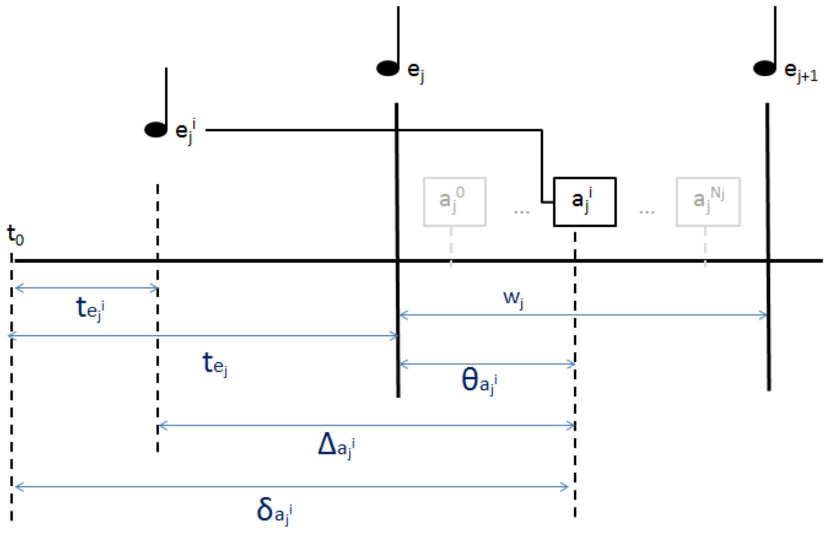


Figure 6.5: Notations

6.3.2 Expression of robustness for a given valuation

In this subsection, we re-write K_{ideal} using other variables more appropriate for the estimation of the robustness.

We consider that the score is at the step 2 of the composition process. Let's fix a valuation π of the actions' parametric delays that verifies K.

Let's fix one event e_j, $0 \leq j \leq n-1$ and one action a_j^i, $i \geq 1$.

We have seen above that the order of launching specified by the ideal trace, a_j^{i-1} before a_j^i, will translate into the constraint $t_{e_j^{i-1}} + \Delta_{a_j^{i-1}}[\pi] \leq t_{e_j^i} + \Delta_{a_j^i}[\pi]$.

Case of consecutive actions triggered by the same event

If $e_j^{i-1} = e_j^i$, meaning that the two consecutive actions a_j^{i-1} and a_j^i are triggered by the same event, then the order of launching specified by the ideal trace, a_j^{i-1} before a_j^i, will be respected in all the possible interpretations of the musician, as long as $\Delta_{a_j^{i-1}}[\pi] \leq \Delta_{a_j^i}[\pi]$. Indeed, in that case the constraint $t_{e_j^{i-1}} + \Delta_{a_j^{i-1}}[\pi] \leq t_{e_j^i} + \Delta_{a_j^i}[\pi]$ is equivalent to $\Delta_{a_j^{i-1}}[\pi] \leq \Delta_{a_j^i}[\pi]$. The events' parametric delays are absent from this constraint. (NB: we know that this constraint evaluates to True because we chose a valuation π validating the constraint K.)

Thus the expression of $K[\pi]$ is the conjunction of atomic constraints in one of the following forms, with $j \in [|0, n|]$ and $i \in [|0, N_j|]$:

- $t_{e_j^i} + \Delta_{a_j^i}[\pi] < t_{e_j^{i+1}} + \Delta_{a_j^{i+1}}[\pi]$, with $e_j^i \neq e_j^{i+1}$ (Form 1)
- $t_{e_j} < t_{e_j^0} + \Delta_{a_j^0}[\pi]$, with $e_j^0 \neq e_j$ (Form 2)
- $t_{e_j^{N_j}} + \Delta_{a_j^{N_j}}[\pi] < t_{e_{j+1}}$ (Form 3)
- $t_{e_j} < t_{e_{j+1}}$ if there are no actions supposed to be launched between e_j and e_{j+1}. (Form 4)

Expression of atomic constraints

In this paragraph, we express the 3 first atomic constraints listed above with the parameters $\theta_{a_j^i}[\pi]$ and $\theta_{a_j^{i+1}}[\pi]$, and $\alpha_{e_j^i}$ and $\alpha_{e_j^{i+1}}$.

- *Form 1:* $t_{e_j^i} + \Delta_{a_j^i}[\pi] < t_{e_j^{i+1}} + \Delta_{a_j^{i+1}}[\pi]$, *with* $e_j^i \neq e_j^{i+1}$

  We have $\Delta_{a_j^i}[\pi] = t_{e_j^{i+1}}^{ideal} - t_{e_j^i}^{ideal} + \Delta_{a_j^{i+1}}[\pi] - (\theta_{a_j^{i+1}}[\pi] - \theta_{a_j^i}[\pi])$ (see the following image).

  Thus $t_{e_j^i} \leq t_{e_j^{i+1}} + \Delta_{a_j^{i+1}}[\pi] - \Delta_{a_j^i}[\pi]$
  $\Leftrightarrow t_{e_j^i} \leq t_{e_j^{i+1}} + \Delta_{a_j^{i+1}}[\pi] - t_{e_j^{i+1}}^{ideal} + t_{e_j^i}^{ideal} - \Delta_{a_j^{i+1}}[\pi] + \theta_{a_j^{i+1}}[\pi] - \theta_{a_j^i}[\pi]$
  $\Leftrightarrow \alpha_{e_j^i} - \alpha_{e_j^{i+1}} \leq \theta_{a_j^{i+1}}[\pi] - \theta_{a_j^i}[\pi]$

  Thus the atomic constraint $t_{e_j^i} + \Delta_{a_j^i}[\pi] \leq t_{e_j^{i+1}} + \Delta_{a_j^{i+1}}[\pi]$ can be expressed by: $\alpha_{e_j^i} - \alpha_{e_j^{i+1}} \leq \theta_{a_j^{i+1}}[\pi] - \theta_{a_j^i}[\pi]$

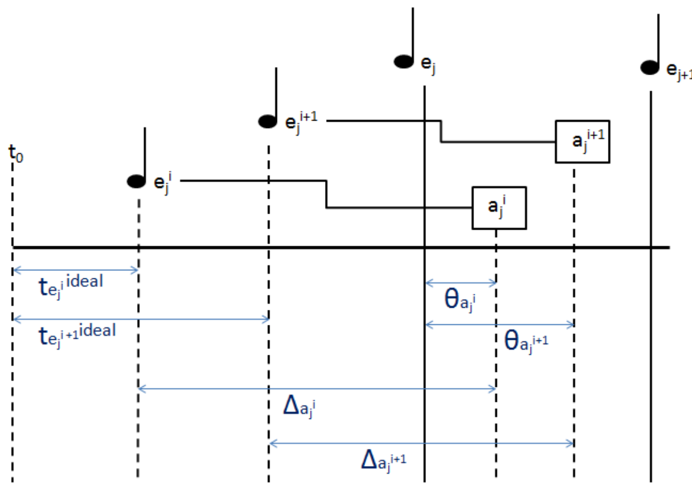


Figure 6.6: Express the Form 1 of atomic constraints

- *Form 2:* $t_{e_j} < t_{e_j^0} + \Delta_{a_j^0}[\pi]$, *with* $e_j^0 \neq e_j$

  By a similar calculation, this constraint can be expressed by : $\alpha_{e_j} - \alpha_{e_j^0} \leq \theta_{a_j^0}$

- *Form 3:* $t_{e_j^{N_j}} + \Delta_{a_j^{N_j}}[\pi] < t_{e_{j+1}}$

  By a similar calculation, this constraint can be expressed by : $\alpha_{e_j^{N_j}} \leq w_j - \theta_{a_j^{N_j}}$

6.3.3 Robustness of the score $S[\pi]$

We have shown that the expression of $K[\pi]$ is the conjunction of atomic constraints, each in one of the following forms, with $j \in [|0, n|]$ and $i \in [|0, N_j|]$:

- $\alpha_{e_j^i} - \alpha_{e_j^{i+1}} \leq \theta_{a_j^{i+1}}[\pi] - \theta_{a_j^i}[\pi]$, with $e_j^i \neq e_j^{i+1}$
- $\alpha_{e_j} - \alpha_{e_j^0} \leq \theta_{a_j^0}$, with $e_j^0 \neq e_j$
- $\alpha_{e_j^{N_j}} \leq w_j - \theta_{a_j^{N_j}}$
- $t_{e_j} < t_{e_{j+1}}$ if there are no actions supposed to be launched between e_j and e_{j+1}.

We consider an event e_k of the score. We consider the constraint $K[\pi]_k$ obtained by substituting in $K[\pi]$ all the delays of the other events of the score by their ideal value: $\forall j \neq k,\ t_{e_j} := t_{e_j}^{ideal}$

Then: $\forall j \neq k,\ \alpha_{e_j} = 0$

The constraint $K[\pi]_k$ is thus the conjunction of atomic constraints, each in one of the following forms:

- $\alpha_{e_k} \leq \theta_{a_j^{i+1}}[\pi] - \theta_{a_j^i}[\pi]$, with $e_j^i \neq e_j^{i+1}$ and $e_j^i = e_k$
- $\alpha_{e_k} \leq \theta_{a_j^i}[\pi] - \theta_{a_j^{i+1}}[\pi]$, with $e_j^i \neq e_j^{i+1}$ and $e_j^{i+1} = e_k$
- $\alpha_{e_k} \geq -\theta_{a_j^0}[\pi]$, with $e_j^0 \neq e_j$ and $e_j^0 = e_k$
- $\alpha_{e_k} \leq \theta_{a_j^0}[\pi]$, with $e_j^0 \neq e_j$ and $e_j = e_k$
- $\alpha_{e_k} \leq w_j - \theta_{a_j^{N_j}}[\pi]$ with $e_j^{N_j} = e_k$
- $t_{e_{k-1}} < t_{e_{k+1}}$ if there are no actions supposed to be launched between e_k and e_{k+1}
- $t_{e_k} < t_{e_{k+1}}$ if there are no actions supposed to be launched between e_k and e_{k+1}.

From these expressions, we can deduce that the quantity to maximize in order to get the best robustness for $S[\pi]$ is:

$$min_{j\in[|0,n|],\ i\in[|0,N_j|]}\{\ \theta_{a_j^0},\quad min_{i\in[|0,N_j-1|]}(|\theta_{a_j^{i+1}}[\pi] - \theta_{a_j^i}[\pi]|),\quad w_j - \theta_{a_j^{N_j}}[\pi]\ \}.$$

6.3.4 The optimal valuation

According to the preceding paragraphs, in order to find the best valuation in terms of robustness of the score, the quantity to maximize is the minimum of the distances between two consecutive actions triggered by different events.

We call ϵ the smallest positive duration that can be affected to a delay in an Antescofo score.

We consider $j \in [|0, n-1|]$.

We note N_ϵ the number of couples of consecutive actions triggered by the same event, and $N_{clusters}$ the number of groups of consecutive actions all triggered by the same event. These numbers can be obtained by the following computation:

```
$N_\epsilon = 0$, $N_{clusters} = 0$, i=1
If $a_j^0$ is triggered by $e_j$, then $N_\epsilon = N_\epsilon + 1$
Else, $N_{clusters} = N_{clusters} + 1$
While i < $N_j$:
    If $a_j^{i-1}$ and $a_j^i$ are triggered by the same event, then $N_\epsilon = N_\epsilon + 1$
    Else, $N_{clusters} = N_{clusters} + 1$
    i = i+1
```

The best valuation is the one verifying, for every two consecutive actions a_j^i and a_j^{i+1}:

- If a_j^i and a_j^{i+1} are triggered by the same event, then $\theta_{a_j^{i+1}} - \theta_{a_j^i} = \epsilon$.
- If a_j^i and a_j^{i+1} are not triggered by the same event, then $\theta_{a_j^{i+1}} - \theta_{a_j^i} = \frac{w_j - N_\epsilon \times \epsilon}{N_j + 1 - N_{clusters}}$

In the following example:

- $N_j = 5$
- a_j^0, a_j^1 and a_j^2 are triggered by the same event e_j^0 different from e_j
- a_j^3 and a_j^4 are triggered by the same event e_j^3 different from e_j^0
- a_j^5 is triggered by an event different from e_j^3

Thus, $N_{clusters} = 3$ and $N_\epsilon = 5$. The distance between two consecutive action triggered by different events is:

$$D = \frac{w_j - N_\epsilon \times \epsilon}{N_j + 1 - N_{clusters}} = \frac{w_j - 5\epsilon}{3}.$$

The following picture illustrates the best valuation of action's delays.

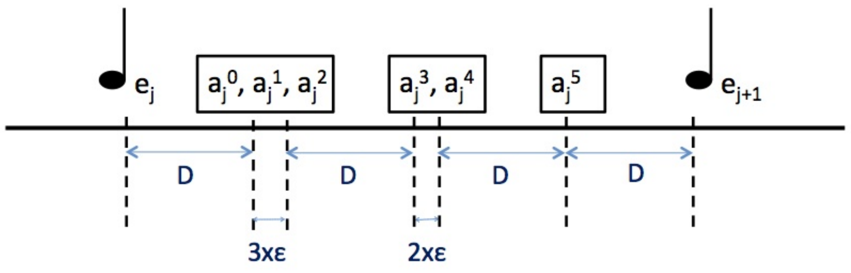


Figure 6.7: Example of best valuation of action's delays

Chapter 7

Conclusion

The initial objective of this internship was to find a way of helping the composer and the interpret forecast the temporal behavior of a mixed-music score during a concert. My contribution was to implement a module that generates a constraint on the parametric delays of the events destined to be played by the musician assuring that the temporal behavior of the system during the concert is acceptable. I also offered a solution of Computer Assisted Composition aimed at helping the composer create the most robust score to the tempo variations of the interpret.

However, before adding the static analysis module to Antescofo, the possibility of physical delays on the score has yet to be included in the model. Indeed, we restricted ourselves in this internship to the case in which all the actions', groups' and events' delays are expressed on the score in relative time. This is the generic case of classical scores, where the duration of notes is expressed in number of tempo pulses. However, the domain specific language Antescofo allows the coexistence in the same score of delays expressed in physical time and delays expressed in relative time.

Thanks

I would like to thank all the people who have given me the opportunity of doing this internship and contributed in some way to this report. Many thanks to Mr. Florent Jacquemard who supervised and mentored my work, for his attention, and for all the time he awarded to me. Many thanks also to Mr. Arshia Cont, the project manager, who welcomed me within his team. Thanks also to Mr. Jean-Louis Giavitto for the time he has dedicated to help me. I thank all the members of the MuSync team for their patience and the time they spent to explain their work to me.

My thanks also go to Mr. Gregory Gournay, who was my tutor from Centrale and gave me all his attention.

Finally, I would also like to thanks Mr. Philippe Cuvillier and Mrs. Florence Quilliard for making our common office a very pleasant place to work.

Bibliography

[1] Philippe Manoury (2007): *Considérations (toujours actuelles) sur l'état de la musique en temps réel.* Etincelle, le journal de la création à l'Ircam, November 2007. http://etincelle.ircam.fr/733.html

[2] Arshia Cont (2011): *Interaction musicale en temps réel entre musiciens et ordinateur*. Interstices.

[3] Antescofo - Anticipatory Score Follower: http://repmus.ircam.fr/antescofo

[4] José Echeveste, Arshia Cont, Jean-Louis Giavitto, Florent Jacquemard: *Antescofo: a Domain Specific Language for real time musician-computer interaction.* Discrete Event Dynamic Systems, 2012. Submitted.

[5] Arshia Cont, José Echeveste, Florent Jacquemard and Jean-Louis Giavitto. *Correct Automatic Accompaniment Despite Machine Listening or Human Errors in Antescofo.* In International Computer Music Conference (ICMC), Ljubljana, Slovenia, 2012.

[6] Arshia Cont *A coupled duration-focused architecture for realtime music to score alignment.* IEEE Transaction on Pattern Analysis and Machine Intelligence. Vol. 32(6), Pp.

[7] Étienne André, Thomas Chatain, Laurent Fribourg, Emmanuelle Encrenaz (2009): *An Inverse Method for Parametric Timed Automata*, International Journal of Foundations of Computer Science, 20(5) Pages 819-836, 2009.

[8] Étienne André and Laurent Fribourg (2010): *Behavioral Cartography of Timed Automata.* In Antonín Kučera and Igor Potapov (eds.), *RP'10, LNCS 6227*, Springer, pages 76–90.

[9] IMITATOR - Parameter synthesis for timed automata: http://www.lsv.ens-cachan.fr/Software/imitator/index.php

[10] HyTech - The HYbrid TECHnology tool: http://embedded.eecs.berkeley.edu/research/hytech/

[11] T. A. Henzinger, P. Ho, and H. Wong-Toi. *A user guide to HyTech.* In TACAS, pages 41–71, 1995.

[12] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. *Counterexample-guided abstraction refinement.* In CAV '00, pages 154–169. Springer-Verlag, 2000.

[13] G. Frehse, S.K. Jha, and B.H. Krogh. *A counterexample-guided approach to parameter synthesis for linear hybrid automata.* In HSCC '08, volume 4981 of LNCS, pages 187–200. Springer, 2008.

[14] Rajeev Alur and David L. Dill. *Automata for modeling real-time systems.* In Proc. 17th International Colloquium on Automata, Languages and Programming (ICALP'90), volume 443 of Lecture Notes in Computer Science, pages 322–335. Springer, 1990.

[15] Rajeev Alur and David L. Dill. *A theory of timed automata.* Theoretical Computer Science, 126(2):183–235, 1994.

[16] R. Alur, T.A. Henzinger, and M.Y. Vardi. *Parametric real-time reasoning.* In STOC'93, pages 592–601, New York, USA, 1993. ACM.

[17] T.S.Hune, J.M.T.Romijn, M.I.A.Stoelinga, andF.W. Vaandrager. *Linear parametric model checking of timed automata.* Journal of Logic and Algebraic Programming, 2002.

[18] Rowe, Robert. Interactive Music Systems Cambridge, MA: The MIT Press 1993

[19] T.A. Henzinger and H. Wong-Toi. *Using HyTech to synthesize control parameters for a steam boiler.* In Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control, LNCS 1165, pages 265–282, London, UK, 1996. Springer-Verlag.

[20] Patricia Bouyer: *An Introduction to Timed Automata*, MPRI, Academic year 2011-2012. Notes de cours

[21] Frank Cassez (2010): *Langages réguliers - Automates finis.* Notes de cours