The Hierarchical Structure May Improve the Resolution of Musical Problems

Myriam Desainte-Catherine

LaBRI[1]
Université Bordeaux I
351, cours de la Libération
33405, Talence Cedex
France

myriam@labri.u-bordeaux.fr

Abstract : *This paper presents some theoretical considerations concerning the link between the resolution of musical problems and their representation. In particular, by the use of formal graph grammars, we show that the hierarchical representation of some musical problems may change their property of context-freeness.*

1. Introduction

Lots of formal models have been used for musical representation, composition, automatic composition, analysis, etc. Among them, finite automatas and formal grammars are certainly the most popular [Che87, Bel90, Roa85]. Limits of formal grammars are well-known: they only provide a syntactic representation of music. Attribute grammars have been introduced for building compilers for programming languages. They correspond to formal grammars which are associated to a semantic. We have proposed [DCBM93] to use attribute grammars, which had already been applied to music for score writing purposes [Sze90], for associating a semantic to musical hierarchies. Hierarchical structures are widely used for representing music [BRBM78, Bal89, BS93]. Then, to be integrated in an environment for musical composition, the attribute grammar model applied to the musical hierarchy has been extended [BBDC94a] in order to become incremental and dynamic (theoretical works have also been done to provide incrementality to formal attribute grammars [Hud91]). The resulting model provides the composer a way to build step by step his musical piece. At each step, he precises whether the structure or the musical objects. The system verifies the consistency of the information with the semantic of the piece. Various musical problems can be solved with that model. However, by contrast with general systems for constraints resolution, we do not want to compute all the solutions. Consistency is the only aim.

Using the theory of graph grammars, we show that musical problems are not represented only by the semantic of the hierarchy. A part of them lies in the hierarchy itself, that is, in the syntax. This theoretical consideration shows that the hierarchical structure must be taken in account when possible. We illustrate this aspect with an example of a problem whose resolution is context-sensitive when it is defined on words (strings) and which becomes context-free when it is defined on graphs. Duality arises at this point: the problem becomes simpler while the data structure gets more complex (graphs instead of words). Hyperedge-replacement graph grammars [Cou90] are used to formalize the resolution of the problem on graphs. This example is an illustration of a theoretical purpose and constitutes a formal proof. The formulation of this example is not supposed to have a practical and immediate musical interest, but illustrates a simplification of a family of problems which are musically interesting.

This result should be useful to people interested in constraints resolution applied to solve musical problems [TA91, BR94, Bal94]. Those problems are only details which are solved independently of the structure of the whole piece in which they are integrated later on. Time performances of systems based on constraints resolution are so bad that they discourage their authors to take in account larger problems, specially those involving time. This result brings the hope that the

[1] Laboratoire Bordelais de Recherche en Informatique – Unité de Recherche Associée au Centre National de la Recherche Scientifique n.1304

complexity of the resolution should not increase in a so much important way if we take in account the overall structure of the piece.

2. The Hierarchical Structure

In this section we briefly recall the most important aspects of a hierarchical structure for music. A hierarchical structure is a tree or a DAG (Direct Acyclic Graph) whose vertices are labeled by labels belonging to a finite set of operators with arity Op and a finite set of musical symbols A. Application of an operator of Op of arity n to a sequence of n symbols of A is a piece of music. Thus, pieces of music may be described by the means of those kinds of expressions. The hierarchical representation of a piece is the abstract derivation tree of the well formed expression defining the piece. Repetitions in the musical piece correspond to several occurrences of the same symbol in the expression and may imply sharing nodes in the tree. In this case, the representation is no longer a tree but a graph, or more precisely a DAG which preserves the underlying tree.

The way we use this kind of hierarchy for musical composition purposes does not matter in this paper. The reader interested in this subject may refer to other works [DCBM93, BBDC94a, BBDC94b, BBDC95].

3. Formal Languages

Though most of musical domains are finite, they may induce infinite computation sets because of elements with no bounded sizes. For example, the set of musical pieces which verify certain properties is advantageously represented by a formal tool providing both analysis and generation. In this section we recall the formal tools that will be used in the proof of the following section, that is, context-free words grammars and hyperedge-replacement graph grammars.

3.1. Finite Automatas

Among the formal tools describing infinite sets, the finite automata [Eil76] is certainly the most simple. In spite of a power of expression relatively weak, it has great advantages: solid theory (union, intersection, superposition, concatenation, etc.), simplicity for programming, efficiency in time. Moreover, the finite automata is used with the same ease for both analysis and generation. However, it is limited concerning musical purposes because it cannot express operations such as, for example, reversing, permuting and repeating.

3.2. Context-free Grammars

Context-free words grammars [Aut] are also used for both generating and analyzing words. Nevertheless, analysis with words grammar is not so easy. Classifications of grammars and algorithms for analysis have been proposed for the construction of efficient compilers. Context-free grammars can be represented by a finite automata equipped with a stack. However, as it is pointed out in the next section, they cannot recognize repetitions. Moreover, their theoretical properties are less interesting than those of the automatas. In particular, the intersection of two context-free grammars is not generally context-free.

3.3. Graph Grammars

Graph grammars are of various types and the definition of their context-freeness depends on the authors. Our illustration uses the formalisms introduced by Bruno Courcelle [Cou90], and more precisely the HR grammars (Hyperedge-Replacement).

The basic rewriting step is the replacement of a hypergraph for a hyperedge, the replacement of a word for a letter thus extending to hypergraphs. These grammars are context-free according to basic aspects of context-freeness for graphs discussed by Bruno Courcelle [Cou90]. Among them, there is the non modification of the context by a rewriting step [2]. This implies that derivation sequences may be described by a derivation tree, different traversals of the derivation tree corresponding to different rewriting sequences and yielding the same graph.

[2] For example, a production rule of a HR grammar may not delete a vertex

Let $G(X)$ the set of finite graphs the edges of which are labelled with labels from X. In order to define formally the hyperedge replacement, graphs are associated to *sources*. Sources are distinguished vertices of the graph that will be involved as "gluing vertices" in the replacement operation. A graph involving k sources will be called a k-graph. The subset of $G(X)$ containing only k-graphs will be denoted by $G(X)_k$.

A graph grammar is a four-tuple $\Gamma = \prec B, U, P, Z \succ$ where B is a terminal ranked alphabet, U is a nonterminal ranked alphabet, P is a finite set of rules of productions, that is, a finite set of pairs of the form (u, D) where $D \in G(B \cup U)_k$ and $u \in U$ and k is the arity of u (k is 1 or 2 in the next example), and Z is the axiom ($Z \in G(B \cup U)$). The set of graphs which are defined by Γ is the language $L(\Gamma) := L(\Gamma, Z)$ where for all $K \in G(B \cup U)_n$

$$L(\Gamma, K) := \{H \in G(B)_n, K \xrightarrow{*} H\},$$

and $\longrightarrow$ is the basic rewriting step defined as follows: $K \longrightarrow H$ if it exists an edge e in K whose label is some u in U and a production rule (u, D) in P such that $H = K[D/e]$, that is, H is the result of the replacement of D for e in K. This graph may be constructed as follows:

- First construct K' by deleting e from K and keeping the vertices of e.
- Add to K' a copy $\bar{D}$ of D, disjoint from K'.
- Fuse the first vertex of e with the first source of $\bar{D}$ and then fuse the second vertex of e with the second source of $\bar{D}$.

The sequence of sources of H is that of K'.

Generation with graph grammars has been widely used and is efficient. By contrast, analyzing graphs by graph grammars is quite an open field. The definition of an automata on general graphs is not easy. Only trees and DAGs provide some interesting results.

4. Interest of the Hierarchical Representation

In this section we show that there exists problems whose resolutions are improved by the use of a hierarchical representation. The proof consists in the exhibition of an example of those problems, the one concerning the musical repetition. We first consider this problem without structure, that is, defined on words, and we point out its well-known context-sensitiveness. Then, the same problem is defined with the knowledge of the structure of the repetition. We prove the context-freeness of this problem by exhibiting a HR grammar recognizing the repetition.

The property of context-freeness is lost when the application of a production rule needs to know the context of its left hand-side member. Intuitively, the graph provides a way to bring a context to a nonterminal by the means of edges or vertices because they are multidimensional.

4.1. Repetition Problem on Words

Let P be the following problem. Let A be a finite alphabet and A^* the free monoid, and let $D = \{u.u, u \in A^*\}$, where the symbol "." denotes the concatenation of words. The problem consists in generating a word $v \in A^*$ belonging to D.

The context-sensitiveness of the language D is well-known. Effectively, in order to be able to repeat a sequence of letters that have already been generated, it is necessary to keep them in memory. However, only a stack is provided as a memory for context-free grammars. The access of the stack is not direct so that it does not permit exact repetition. By contrast, it permits repetitions of sequences in a reverse order.

4.2. Repetition Problem on Graphs

Let us now transpose the repetition problem in the graph world. The transposition of the language A^* will be denoted by $\dot{A}$.

As it has been sketched in section 2, the hierarchy is defined with two sets, the set Op of operators and the set A of musical symbols. The elements of both A and Op are of arity 1 and will be used for labeling graph vertices. With this formalism, only edges are labeled. Labels for vertices are thus represented by an edge of arity 1 attached to the vertex and carrying the desired label.

Let us introduce the operator of concatenation, denoted by the symbol "." of the set Op, and define it in terms of a graph operation. For a and b in $G(X)_1$, the graph

$$K = a.b$$

is also in $G(X)_1$. It is represented by a tree whose root is a vertex labeled by the operator "." and having two sons being fused with the sources of the graphs a and b. The source of the graph K is its root.

Let $G_1(A)$ be the set of 1-graphs which are reduced to one vertex labeled with a label in A (by the means of an edge of arity 1 attached to it) then $\dot{A}$ is the set of 1-graphs $K \in G(A \cup \{.\})$ which are defined by induction:

- $K \in G_1(A)$
- $\exists s \in G_1(A)$ and $\exists K' \in \dot{A}$ such that $K = s.K'$.

Now D is defined exactly the same way than in the previous subsection. The definition of the concatenation operator becomes: $D = \{u.u, u \in \dot{A}\}$, where the operation "." is the graph operation described just before.

Now, the problem $\mathcal{P}$ consists in generating a graph $v \in \dot{A}$ belonging to the language D. The graph grammar $\Gamma_1 = \prec B, U, P, Z \succ$ recognizing D is given below.

- $B = \{e\} \cup A \cup \{.\}$, where e is of arity 2. The label e will be given to all edges of arity 2 whose labels do not matter. In the terminal graph, only label of arity 1 are significant.
- $U = \{z, u, v\}$, where z is of arity 1 and u and v are of arity 2.
- $Z = z$.
- P is displayed on Figure 1. In this example, vertices carrying the same label are not shared so that the resulting structure is always a tree. In order to point out that it may be a DAG depending on implementation choices concerning vertices sharing, let us describe briefly two other possible production sets:
  - Let us call Γ_2 the grammar which shares vertices which are leaves carrying the same label. This grammar only differs from Γ_1 in the following way: the two vertices appearing in the right-hand side of the production rule 4 are fused.
  - Let us call Γ_3 the grammar which shares the whole repetition. This time, vertices which are joined by the edge labeled u in rule 1 are fused so that labels u and v are no more useful and rules 2, 3 and 4 are removed. Other rules are necessary for describing the structure of a single occurrence.

An example of derivation sequence is displayed on Figure 2. In this example, $A = \{C, D, E, F\}$ and the grammar Γ_1 is used for generating the graph. The same derivation using grammar Γ_2 produces a DAG instead of a tree (left leaves of the tree should all be fused two by two, the result of the fusion carrying the common label). Application of the grammar to a graph K for analyzing, should be performed by first attaching the axiom to the source of K.

5. Conclusion

We have shown that hierarchical structure may improve the power of expression of musical problems without losing context-freeness. This property should have some implications on the complexity of

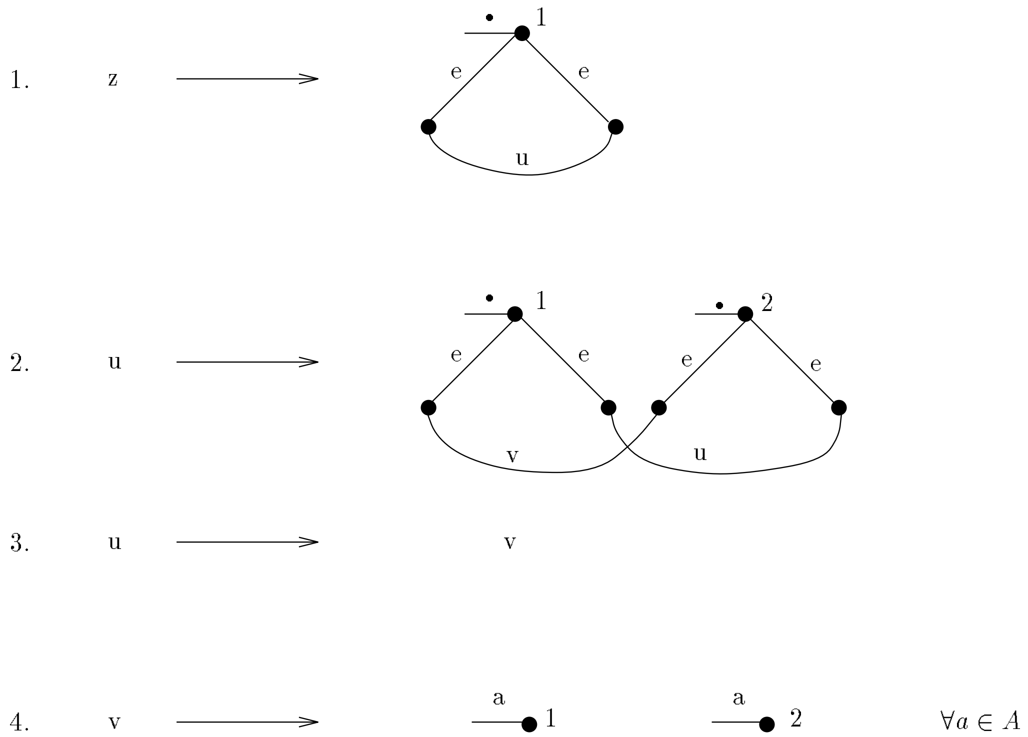

Figure 1: Production Rules of a Graph Grammar Solving P. Sequences of sources of the right-hand side graphs are indicated by a numerotation. For example, in the first rule, the left-hand side edge z is of arity 1 so that only one source is pointed out on the right-hand side. Left-hand side edges of the other rules are all of arity 2 so that two sources are pointed out on the right-hand side graphs. The rule number 3 is multiple. Its multiplicity is equal to the cardinality of A. With this formalism, labels are only carried by edges. Thus, small labeled edges of arity 1 represent vertices labels, that is, operators and musical symbols.

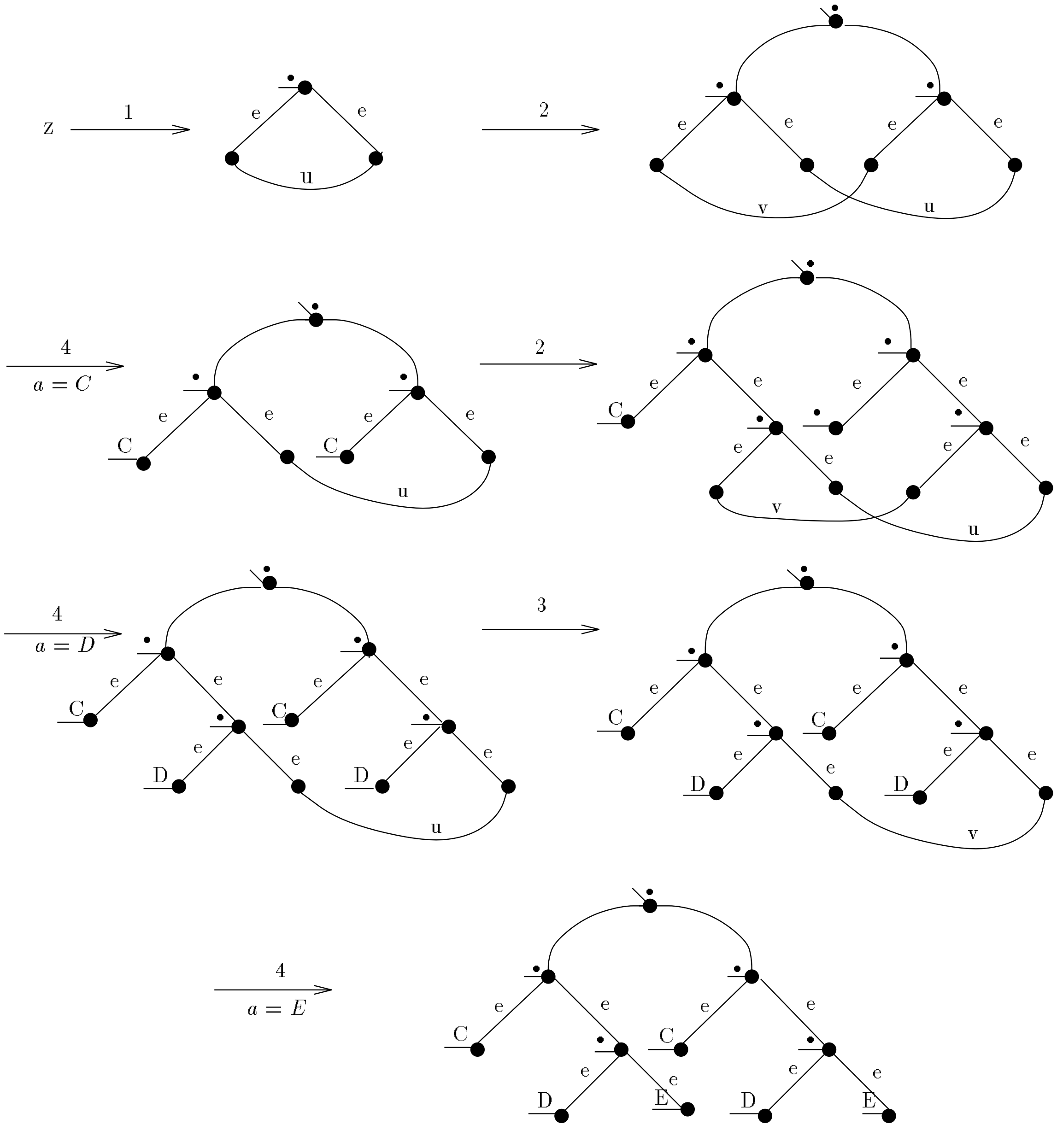

Figure 2: Example of Derivation Sequence. Each derivation step is labeled by the number of the rule which is applied. Applications of rule 4 are equipped with the instanciation of a which is used.

the resolution. In this paper, we used graph grammars as a support for a theoretical purpose which is independent of the implementation of the resolution of problems expressed in the hierarchy. We shown that the structure brings some "simplification" of musical problems. Now, the resolution on graphs structures is certainly more complex than on words. Thus, in order to go further in this study, we should compare resolution performances more precisely by providing algorithms for both generation and analyze as well as their complexities or their time execution. Other kinds of problems should be studied the same way.

This result should be useful to people interested in constraints resolution applied to solve musical problems. Those problems are for now reduced to some details of a musical piece. We may hope that, if the global structure of the piece is as simple as a tree or a DAG, it should help in improving the resolution of the whole piece.

We can expect that the complexity of the piece should not exceed too much the modeling strategies of the composer so that he can keep on mastering it. Thus, the complexity should be *modular* in some way. This property of modularity arises in software engineering, in particular, which is a discipline dealing with complexity of computer programs. In the same way, the programmer must be able to manage his program even when it becomes very large. Rules for reducing a problem to a set modules which are interconnected all together are based on psychological experiments. Modules must be small, have a clear and simple semantics and be as independent as possible of the context. Modules must be interconnected in a simple way (in fact as less connections as possible). Probably, this kind of observation should be applied to the composer. The results of musical pieces analysis often leads to very highly structured formalization. Even if the compositional process does not take in account all the structures that are to be found by an analysis, it is surely highly modular in the software engineering sense. If it is not the case, the composer may be in danger of losing the mastery of his work. Thus, we can expect that most of musical pieces have, during the compositional process, structures that are DAGs whose vertices are musical modules, that is, graphs even more complex than DAGs but relatively small.

References

[Aut] J-M. Autebert. *Théorie des Langages et des automates.* Masson.

[Bal89] M. Balaban. Music structures: A temporal-hierarchical representation for music. *Musicometrika*, 2:1–50, 1989.

[Bal94] P. Ballesta. Contraintes et objets, clefs de voûte d'un outil d'aide à la composition. In *Proc. of the JIM'94 (Journées d'informatique Musicale), Bordeaux*, pages 39–52, March 1994.

[BBDC94a] Kablan Barbar, Anthony Beurivé, and Myriam Desainte-Catherine. Incremental evaluation in a musical hierarchy. In *Proceedings of the Brazilian Synposium on Computer Music*, August 1994.

[BBDC94b] Kablan Barbar, Anthony Beurivé, and Myriam Desainte-Catherine. On the resolution of musical conflicts. In *Proc. of the Conference on New Music Research, Gent (Belgium)*, October 1994.

[BBDC95] Kablan Barbar, Anthony Beurivé, and Myriam Desainte-Catherine. Structure hiérarchique et relations temporelles d'une pièce de musique. In *Proc. of the JIM-95 (Journées d'Informatique Musicale), Paris*, April 1995.

[Bel90] B. Bel. Langages formels, formes et musique. Technical Report PR-323, M.I.M., 1990.

[BR94] A. Bonnet and C. Rueda. Situation: un logiciel de programmation par contraintes pour l'aide à la composition musicale. In *Proc. of the JIM'94 (Journées d'informatique Musicale), Bordeaux*, pages 1–4, March 1994.

[BRBM78] W. Buxton, W. Reeves, R. Baecker, and L. Mezei. The use of hierarchy and instance in a data structure for computer music. *Computer Music Journal*, 2(4):10–20, 1978.

[BS93] M. Balaban and C. Samoun. Hierarchy, time and inheritance in music modeling. *Languages of design*, 1(3), 1993.

[Che87] M. Chemillier. Monoide libre et musique. *RAIRO, Informatique Théorique et applications*, 21(3):341–417, 1987.

[Cou90] B. Courcelle. *Hanbbook of Theoretical Computer Science*, chapter Graph Rewriting: An Algebraic and Logic Approach. Elsevier Science Publisher, 1990.

[DCBM93] Myriam Desainte-Catherine, Kablan Barbar, and Alain Miniussi. The semantics of musical hierarchies. *Computer Music Journal*, 17(4):30–37, Dec 1993.

[Eil76] S. Eilenberg. *Automata, Languages and Machines.* Academic Press, 1976.

[Hud91] S. E. Hudson. Incremental attribute evaluation: A flexible algorithm for lazy update. *ACM Transactions on Programming Languages and Systems*, 13(3):315–341, jul 1991.

[Roa85] C. Roads. *Foundations of Computer Music.* M.I.T. Press, 1985.

[Sze90] S. Szeider. The semantics of musical scores. *Interface*, 19:175–184, 1990.

[TA91] C.P. Tsang and M. Aitken. Harmonizing music as a discipline in constraint logic programming. In *Proc. of the ICMC'91, Montréal*, pages 61–64, October 1991.