Acquisition et représentation de connaissances en musique

Bernard Bel

UNIVERSITE DE DROIT, D'ECONOMIE ET DES SCIENCES
D'AIX-MARSEILLE

FACULTE DES SCIENCES ET TECHNIQUES DE SAINT-JEROME

Thèse présentée
par M. Bernard BEL

pour obtenir le grade de Docteur en Sciences (nouveau régime)
de l'Université de Droit, d'Economie et des Sciences d'AIX-MARSEILLE
(AIX-MARSEILLE III)

Spécialité: INFORMATIQUE

Acquisition et représentation de connaissances en musique

soutenue le 28 novembre 1990 devant la Commission d'Examen:

M. Jean-Paul ALLOUCHE, rapporteur
M. Jean-Claude BERTRAND
M. Eugène CHOURAQUI
M. Alain GUENOCHE, directeur
M. Otto LASKE
M. Jean-Claude RISSET, rapporteur
M. Bernard VECCHIONE, rapporteur

Remerciements

Résumé

Table des matières (Table of contents)

Remerciements

Je remercie les chercheurs qui m'ont guidé par leurs conseils dans les divers domaines scientifiques liés à cette étude: MM. Jean-Paul Allouche pour la théorie des automates, Alain Guénoche, directeur de thèse, pour l'algorithmique combinatoire, Jim Kippen pour l'anthropologie et l'ethnomusicologie, Otto Laske pour la musicologie cognitive, et Bernard Vecchione pour la musicologie et l'épistémologie.

Je remercie M. Eugène Chouraqui, directeur du Groupe Représentation et Traitement des Connaissances, grâce à qui j'ai pu mener à bien ces travaux dans les meilleures conditions de travail, en bénéficiant de la documentation du laboratoire et du soutien des chercheurs, ingénieurs, étudiants, et du personnel administratif du GRTC.

Je remercie enfin tous les membres du jury qui était composé, outre les personnes déjà citées, de MM. Jean-Claude Bertrand et Jean-Claude Risset.

Cette thèse est dédiée à la mémoire de deux personnes qui ont à la fois suscité et encouragé mon engagement dans ce projet: John Blacking, musicien et anthropologue de l'Université de Belfast, décédé le 24 janvier 1990, et Afaq Husain Khan, musicien de Lucknow, décédé le 18 février 1990. Leur générosité et leur ouverture d'esprit exceptionnelles ont été une source d'inspiration dans les moments les plus difficiles.

Les travaux menés dans la phase initiale de cette étude ont reçu le soutien de la *Ford Foundation* (USA), du *National Centre for the Performing Arts* (NCPA, Bombay), de l'*International Society for Traditional Arts Research* (ISTAR) et du *Leverhulme Trust* (Royaume Uni).

Acquisition et représentation de connaissances en musique

Définitions et conventions

La terminologie utilisée dans ce volume est, pour l'essentiel, une traduction de la terminologie anglo-saxonne en théorie des langages formels, plus particulièrement Salomaa[1], Kain[2] et Révész[3]. Certaines désignations ou définitions ne sont pas rigoureusement identiques dans diverses publications. Nous précisons ici les conventions adoptées. Les titres de chapitre et de paragraphes ont été traduits en anglais afin de restituer les termes d'origine et de faciliter la lecture de l'ouvrage.

1. Conventions diverses

Avec les quantificateurs "$\forall$" ("quel que soit") et "$\exists$" ("il existe"), nous utilisons la notation

$$x_1, \ldots, x_n \in E$$

au lieu de

$$(x_1, \ldots, x_n) \in E^n$$

sauf bien sûr pour désigner un n-uplet.

Lorsqu'un objet mathématique est désigné par une lettre minuscule de l'alphabet latin, toute mention isolée de cet objet dans une phrase est indiquée en italique. Ainsi, *a* est en italique pour être discerné du présent du verbe "avoir", par contre l'expression $a \in E$ n'introduit aucune ambiguïté. Les caractères ou chaînes de caractères sont délimités entre guillemets: "".

Nous désignons par N l'ensemble des entiers naturels {0, 1, …}, Z celui des entiers relatifs, Q celui des nombres rationnels et R celui des nombres réels. Appliquée à ces ensembles, la notation X* désigne $X \setminus \{0\}$, alors que dans tout autre contexte elle désigne l'opération monoïde.

2. Applications

Soient deux ensembles E et F. On appelle **graphe d'une correspondance** entre E et F un sous-ensemble Γ du produit cartésien E x F. La **correspondance** (*mapping*) est l'opérateur *f* qui, à tout élément *x* de l'**ensemble de départ** E pour lequel il existe au moins un couple $(x,y) \in \Gamma$, associe son **image** f(x) dans F:

---

[1] 1973

[2] 1981

[3] 1985

$$f(x) = \{y \mid (x,y) \in \Gamma\}$$

Une correspondance telle que pour tout $x \in E$ il existe au moins un couple $(x,y) \in \Gamma$ est appelée une **application multivoque**. On peut sans perte de généralité ramener toute correspondance *f* à une application multivoque[4] en adoptant la convention:

$$(x,y) \notin \Gamma \iff f(x) = \varnothing$$

où "∅" désigne l'ensemble vide. L'application est **univoque** lorsque card(f(x)) = 1 pour tout $x \in E$. [5]

Nous convenons que toute application est univoque à moins que l'épithète "multivoque" ne soit explicité.

L'application *f* est **injective** si

$$\forall x,y \in E, \quad x \neq y \implies f(x) \cap f(y) = \varnothing$$

et **surjective** si

$$\forall y \in F, \exists x \in E \mid y \in f(x) .$$

A l'inverse de certains auteurs[6], nous convenons qu'une application ne peut être bijective que si elle est univoque. Une application univoque est **bijective** si elle est à la fois injective et surjective. On retrouve ainsi une propriété classique des bijections:

> S'il existe une bijection entre E et F, alors card(E) = card(F).

Nous appelons **fonction multivaluée** la donnée d'un ensemble de départ E, d'un ensemble d'arrivée F, et d'une application multivoque *f* de E dans F. Le **domaine de définition** de définition de la fonction est:

$$D_f = \{ x \in E \mid f(x) \neq \varnothing \}$$

Si l'application *f* est telle que, pour tout $x \in E$, $card(f(x)) \leq 1$, alors la fonction est **monovaluée**. Nous convenons qu'une fonction est monovaluée à moins que la propriété "multivaluée" ne soit mentionnée explicitement.

3. Langages formels

On appelle **suite** toute application *f* d'une partie contiguë E = {1,…,n} de N dans un ensemble quelconque F. L'image f(x) d'un entier *i* est appelée le **terme de rang *i*** de la suite. Si E = N la suite est infinie.

Nous appelons **liste** la réduction d'une suite à une partie de N. Le nombre card(E) est la **longueur de la liste**.

---

[4] D'où le terme anglais *mapping* utilisé indifféremment pour les deux.

[5] "card(X)" désigne le cardinal de l'ensemble X.

[6] Kaufmann & Pichat 1977, I, p.84.

Une **chaîne** (*string*) est une liste dans un ensemble F fini appelé **alphabet**. Nous utilisons le mot "**symbole**", et non "caractère", pour désigner tout élément d'une chaîne. Un symbole peut être un graphème quelconque ou même une chaîne de symboles dans un autre alphabet. Nous avons évité les désignations "mot" et "phrase" qui peuvent renvoyer à la linguistique ou à la musique. Au chapitre VI, "**mot**" désigne un élément d'un ensemble fini de chaînes (un **vocabulaire**) qui sert à **coder** un langage formel.

Nous représentons "." l'opération de concaténation de listes ou de chaînes. Cette représentation est omise lorsqu'aucune ambiguïté n'est possible.

La **chaîne vide** est représentée par le symbole λ.

I. Introduction

Un objectif de cette étude est de traiter certains problèmes de représentation des connaissances: traitement du temps, parallélisme et synchronisation, multiplicité des fonctions sonologiques, etc., qui sont spécifiques à la musique par rapport à d'autres formes de communication.[7] Bien que dérivés, pour la plupart, de la théorie des langages formels, les formalismes décrits n'ont qu'un lien de parenté lointain avec ceux de la linguistique computationnelle.

La plupart des projets “intelligence artificielle et musique” partent d'outils existants (réseaux sémantiques, automates et grammaires formelles, systèmes experts, etc.) à l'aide lesquels il est possible de représenter certains concepts musicaux, par exemple les rudiments du solfège tonal. Peu de chercheurs s'intéressent à l'acquisition de connaissances, prenant pour argent comptant que l'univers du discours musical est entièrement décrit par les traités de “théorie de la musique”. Les problèmes des formes musicales autres que la musique occidentale sont rarement abordés. Enfin, on peut s'interroger sur l'utilisation pratique des connaissances ainsi modélisées, et, partant, sur la finalité des réalisations informatiques.

Nous pensons donc qu'il est utile de situer cette étude dans le contexte des problématiques musicologiques et technologiques qui l'ont suscitée.

1. La notion de “connaissance” en musique (*the concept of “knowledge” in music*)

1.1 Sonétique vs. sonologie

Une partie de l'informatique musicale s'intéresse aux modèles d'analyse et de synthèse du son (l'indispensable *lutherie* électronique/numérique), à l'étude des fonctions perceptives mises en jeu dans l'écoute musicale, et à celle des caractéristiques des gestes producteurs de sons. Ce domaine est désigné par Vecchione[8] sous le terme de “**sonétique**”:

[7] Pour les sémioticiens, postuler une spécificité du musical revient à rompre avec le structuralisme, pour qui le langage verbal est ce qui fonde toute communication. Pour Jakobson, par exemple, “le langage, c'est réellement les fondations même de la culture. Par rapport au langage, tous les autres systèmes de symboles sont accessoires et dérivés.” (Cité par Vecchione 1990a, p.93)

[8] 1990a, pp.73-4.

> Ce type de recherches a donné lieu à l'émergence de nouveaux secteurs d'investigation, comme, par exemple la psychoacoustique et la métrologie subjective. La psychoacoustique étudie les types d'illusions auditives associées à des signaux acoustiques donnés, et la constance ou la variabilité de ces associations.[9] Quant à la métrologie subjective, elle vise l'élaboration de métriques sur des espaces de variables perçues, comme, par exemple, l'élaboration d'une métrique de proximité sur l'espace des timbres,[10] etc. De son côté, l'étude des caractéristiques du "geste" producteur d'événements acoustiques a pour objet de relier les caractéristiques du signal acoustique à celles du geste qui le produit.[11]
>
> L'ensemble tripartite de ces études — établissement de liaisons précises entre signaux acoustiques et caractéristiques numériques du geste producteur de signaux, analyse numérique des signaux acoustiques, repérage d'associations régulières entre certains types de signaux acoustiques et certains contenus de conscience réceptifs — devrait former l'ossature de cette discipline — la "sonétique", pour ne pas dès l'abord la confondre avec la "sonologie" — qui a ouvert des champs de recherche nombreux et féconds en musicologie.

Par analogie avec la distinction entre phonétique et phonologie, la **sonologie** peut être définie comme

> science de la classification fonctionnelle des faits "etics", acoustiques ou humainement facturés, signés par le geste ou par la réception […] [12]

sachant que des unités qui sont acoustiquement différentes (unités **etic**) peuvent être perçues comme équivalentes (unités **emic**) au niveau du "sens" ou de la "fonction". En introduisant cette fonction classificatoire, on s'intéresse, non plus à la musique dans sa seule réalité acoustique, psychoacoustique et gestuelle, mais aussi aux activités cognitives de **production** et de **réception**.

Se référant explicitement à Simon[13], Laske[14] présente la sonologie — par opposition à la sonétique — comme une **science de l'artificiel**, tout en spécifiant clairement l'articulation entre la sonologie et ce que nous avons désigné par "sonétique":

> For purposes of grammatical investigation as well as for an enquiry into performance, a sonological theory defined out of relation with syntax and semantics is useless; equally useless is a syntactical component whose definition fails to account for knowledge concerning the sound properties of musical structures.[15]

1.2 Performance vs. compétence

Les fondements de la **musicologie cognitive** sont à rechercher dans des travaux anciens, mais qui ont très peu attiré l'attention des chercheurs à l'époque de leur publication. On peut lire par exemple, dans le projet de sonologie présenté par Laske:[16]

---

9 Cf. les travaux de Risset et autres.

10 Cf. les travaux de Wessel et autres.

11 Voir les études menées par l'équipe de l'ACROE à Grenoble.

12 Vecchione 1990a, p.75.

13 1969

14 Cf. le chapitre "Is Sonology a Science of the Artificial?" dans Laske (1975).

15 Laske 1972a, p.29

16 1972b, pp.354-5.

> […] the notion of *sonology* was defined as the name of a discipline dealing with the design of sound artifacts and leading, in three stages, […] to the formulation of performance models for music:
>
> I. Sound Engineering and Sonic Representation[17]
>
> II. Pattern Recognition and Sonological Representation
>
> III. Musical Representation and Definition of Intelligent Musical Systems.

Plus on s'éloigne de la sonétique (en se rapprochant du niveau III) et plus les phénomènes décrits par la sonologie sont dépendants des conventions (explicites et implicites) qui sont à la base de tout système musical.

A la limite, on serait tenté d'identifier tout système musical avec cet ensemble de conventions, autrement dit, comme Farnsworth, de définir la musique comme “un ensemble de structures sonores acceptées socialement”.[18] De ce point de vue, toutefois, aucune distinction ne serait plus possible entre le niveau “grammatical” (celui de la **compétence**) et le niveau “stratégique” (celui de la **performance**[19]), ce qui reviendrait à supposer que la connaissance musicale est entièrement faite de contraintes pragmatiques. Pour employer la terminologie linguistique, on ne s'intéresserait plus qu'aux structures de surface,[20] de sorte qu'on serait dans l'impossibilité de reconstituer un processus intersubjectif de communication musicale.[21]

L'attitude inverse sur la question de la performance — héritée de la linguistique générative de Chomsky[22] — consiste à considérer la performance comme une simple application de règles de compétence, autrement dit les règles de stratégie ne sont que des “déviations tolérables” des règles grammaticales.

Une proposition nuancée sur la relation entre compétence et performance a été énoncée par Laske. Reprenant une typologie (non publiée) de Hagedorn, il distingue quatre types de **porteurs de connaissance** (*knowledge holders*):[23]

1) Le praticien professionnel (*professional practitioner*), qui maîtrise la structure conceptuelle des matériaux qu'il utilise;
2) Le travailleur doté de connaissance pratique (*practical knowledge worker*), dont la compétence est un mélange de connaissances techniques, de procédures conventionnelles et d'expérience;
3) L'exécutant (*performer/samurai*), dont les actes sont des combinaisons spontanées de “ficelles de métier” (*skills*), autrement dit de connaissances précompilées;

---

17 Le terme “*sonic*” est explicité au chapitre VII §1.3.

18 1958, p.17.

19 Laske 1972a, pp.20-51. A propos de la relation compétence/performance en linguistique, Chomsky et Halle (1973) écrivent:

“La performance, c'est-à-dire ce que le locuteur-auditeur réalise effectivement, est fondée non seulement sur la connaissance qu'il a de la langue, mais sur bien d'autres facteurs — comme les limitations de la mémoire, l'inattention, la distraction, les connaissances et croyances non-linguistiques, etc.”

20 C'est une question de cet ordre qui se pose à propos des applications des réseaux neuromimétiques à la production ou à l'analyse musicales. Voir à ce sujet Todd 1989; Leman 1990; Lischka 1990.

21 Laske 1972a, p.21.

22 Pour Claude Hagège, à l'inverse, qui a observé la genèse des langues dans ce qu'il appelle le “laboratoire créole”, la performance est **génératrice** de compétence.

23 Laske 1988b, pp.148-9.

4) Le négociateur (*persuader-negotiator*), qui gère des motivations spécifiques aux situations et aux acteurs.

En musique, cette classification correspond à des types d'activité qui vont de la composition (1-2) à l'improvisation (2-3) et au déchiffrage (3), les premières nécessitant un niveau élevé de compétence, et les dernières faisant plus appel à la performance.

1.3 Œuvres et activités (*works vs. activities*)

Les notions de compétence et de performance s'appliquent à la représentation d'un *ensemble* de productions — un **idiome musical**. En faisant implicitement référence à un "**sujet idéal**", elles mettent au second plan l'individualité de toute œuvre musicale. De même que la linguistique s'intéresse à la fonction de communication verbale et la littérature au caractère unique de certaines productions, il est compréhensible que la musicologie s'interroge sur la réalité de l'œuvre autant que sur la notion de "système musical".

Alors que la musicologie traditionnelle (qu'elle s'intéresse aux œuvres anciennes ou contemporaines) se contente de décrire la structure des œuvres (l'œuvre en tant qu'**objet**), les propositions de la sémiotique post-structuraliste et de la poïétique[24] s'intéressent à l'œuvre en tant que **travail**:

> Alors que le mot *structure* et le mot *signe* étaient les épicentres des mouvements intellectuels tournant autour des œuvres d'art comme "textes à lire", voici que le mot *création* se met à polariser l'attention qu'on porte aux œuvres, non seulement en amont de leurs effets esthétiques et de leur mise en objet de sciences, mais aussi comme "choses à faire".[25]

Parmi les activités afférentes à un système musical, on peut distinguer trois phases: l'élaboration de l'œuvre, sa diffusion et sa réception, notions que l'on rassemble sous le terme d'**instauration**:

> Le procès d'instauration d'une œuvre musicale est un procès d'une rare complexité, qu'on peut analyser au moins en trois cycles successifs:[…] le procès de création d'une œuvre, son procès de diffusion (ou d'institutionnalisation), son procès d'analyse (ou de déchiffrement).[26]

La sémiologie tripartitionnelle de Molino[27] distingue, dans toute œuvre, les niveaux **poïétique** (celui de la création), **esthésique** (celui de la réception) et **neutre**. Ce dernier niveau est celui de l'œuvre en tant qu'objet, celui qui, dans cette approche, devrait être étudié en premier. Toutefois, dans ce que Vecchione appelle **sémiotique écoformative**, l'œuvre est toujours

> considérée à travers la relation d'écoformation, d'éco-instauration, qui la relie à un environnement d'acteurs l'instituant, l'informant, l'interprétant, la recevant, la déchiffrant, etc. En sémiotique écoformative, la notion de niveau d'une œuvre-chose qu'on pourrait connaître objectivement n'a pas de sens. L'œuvre comme objet n'existe qu'en tant qu'instituée par la médiation d'actes d'écoformation: des actes qui, simultanément, informent l'œuvre et, par le moyen de cette œuvre, informent en retour l'environnement anthropologique sur lequel les auteurs de ces actes cherchent plus ou moins consciemment à agir.[28]

---

[24] Science du "faire". Pour Passeron (1984, p.153), elle s'applique plus généralement à "la totalité de son corpus: l'ensemble des conduites opératoires, de tout domaine où peut s'observer l'élaboration, la production, la création des œuvres."

[25] Passeron 1984, p.149.

[26] Vecchione 1990a, p.87

[27] 1975. Cette théorie a été appliquée à la musique par Nattiez (1976).

[28] Vecchione 1990a, p.95

Vue sous cet angle, l'analyse musicale n'est plus seulement une recherche de cohérence dans les structures de l'œuvre (ce que l'on pourrait réduire à un problème d'analyse des données: la recherche d'un espace de représentation pertinent), mais une **reconstruction** de l'œuvre. Elle vise donc l'**explication** et la **compréhension** autant que la **description**.

2. Les domaines d'application de cette étude (*domains of application of this study*)

Un changement important survenu dans la musicologie des années 1970-80 est l'impact des études anthropologiques de la musique — philosophie de l'esthétique et sciences de l'art, psychologie historique de la musique, sémiotique post-structuraliste, poïétique, etc.:

> […] cette révolution de paradigme s'est opérée dans les mêmes années que l'introduction de l'informatique en musicologie. Mais, ne bénéficiant ni de la crédibilité sociale dont bénéficiaient les travaux effectués dans l'orbe des sciences dures et des technologies, ni d'institutions où ces "sciences de la musique" auraient pu s'étudier et s'enseigner, on comprendra la gêne que le musicologue et le compositeur ressentent aujourd'hui face à des travaux qui, bien que révolutionnaires, lui paraissent souvent "passer à côté de l'essentiel". Les travaux scientifiques effectués à propos de la musique manquent souvent d'épaisseur, d'authenticité musicale. Ils ne visent que très rarement à expliciter le musical dans sa nature propre.[29]

L'anthropologie de la musique a eu le mérite de révéler un ethnocentrisme sous-tendu par des positions innéistes et universalistes fortement ancrées dans la musicologie traditionnelle.[30] Les travaux sur les musiques "ethniques"[31] ont contribué fortement à relativiser ces positions tout en suscitant une interrogation en retour sur la culture occidentale:

> Ethnomusicological studies […] have suggested that even European tonal music could be described in non-musical terms that would make it more easily comparable with some non-European musical traditions. And computer music that is composed on the basis of the coherence of its numerical base rather than its sonic product, can be shown to have more in common with some music of pre-industrial societies than with its immediate Euro-American antecedents.[32]

En définitive, que l'on s'intéresse aux activités créatrices d'œuvres "ethniques" ou contemporaines, on en vient à poser les mêmes problèmes.[33] L'intérêt d'une approche formelle est précisément de dégager les convergences de ces diverses approches, contribuant ainsi à réaliser le rêve de Blacking:

> I never thought of ethnomusicology as a separate subject, since I believe it can just as well be called cognitive anthropology […] I believe we may be able to take the "ethno" out of ethnomusicology before long. Then we may once more have a unified musicology, a musicology truly fertilized and enriched by the contributions of ethnomusicology.[34]

---

[29] *Op.cit.* p.79

[30] Ce problème n'est pas typiquement occidental. La musicologie indienne, par exemple, est fortement ethnocentrique dès qu'elle se détache de l'aspect historique de la réalité musicale. (Voir Bel 1988)

[31] Bel 1990d; Kippen 1990.

[32] Blacking 1984, p.366

[33] Une partie de l'"ethnicité" des œuvres extra-européennes tient à l'anonymat des compositeurs et non à un refus présumé de technicité. En Europe même, la personnalisation de l'acte créateur est un fait social récent.

[34] Blacking 1989.

3. Grammaires musicales et grammaires formelles (*musical grammars vs. formal grammars*)

Dans son acception anthropologique, la notion de **grammaire musicale** ne fait pas référence à une classe de formalismes particuliers empruntés à la linguistique ni à la théorie des langages formels:

> Grammars are attempts to codify the regularities of structure that communities generate in order to give coherence to their communication and to enable individuals to share meanings.[35]

Une grammaire musicale, du point de vue des anthropologues, devrait contenir des informations qui rendent compte à la fois du contexte, de la finalité de la performance, et de facteurs non-musicaux:[36]

> If non-musical languages (of social interaction, movement, or speech, etc.) are embedded in some musical languages, a way must be found to include these elements in the musical grammars without reducing the significance of the sonic dimension.[37]

La remarque précédente souligne une divergence fondamentale entre l'approche anthropologique et celle de l'intelligence artificielle. Le souci des anthropologues, en effet, est moins de rendre les modèles opérationnels que de prendre en compte toutes les dimensions de la réalité étudiée. L'intelligence artificielle, par contre, vise une modélisation informatique efficace — même partielle — de cette réalité, orientée vers la **résolution de problèmes**. L'originalité de notre démarche a consisté à introduire l'ordinateur comme un partenaire **actif** dans la phase d'acquisition des connaissances,[38] donnant de ce fait plus d'importance aux modèles **génératifs**, censés expliquer la compétence, qu'aux modèles **analytiques** qui se contentent de l'illustrer.[39] La démarche expérimentale correspondante (**anthropologie dialectique**) met en jeu un schéma relationnel expert-analyste-machine applicable à l'élaboration des **systèmes experts**. (Voir chapitre II)

L'approche dialectique permet de souligner la variabilité des contextes, et, par là même, d'identifier les aspects de la réalité qui restent stables, autrement dit plus proches d'un modèle de compétence.[40] Les données du contexte sont ainsi prises en compte *sans être nécessairement incorporées au modèle informatique*.

Dans la phase initiale de cette étude nous nous sommes intéressés[41] à un système musical (un “langage” des percussionnistes du nord de l'Inde) pour lequel existe un

---

[35] Blacking 1984, p.364. Le terme “communication” ne se réduit pas ici à une fonction sémantique attribuée aux signes, objets ou idées décrits verbalement.

[36] Une étude approfondie de la communication esthétique a été proposée par Laske (1972a). Sur la représentation des paramètres spatio-moteurs dans la technique instrumentale, voir par exemple Baily 1989.

[37] Blacking 1984, p.370

[38] En informatique musicale, le premier système destiné à l'acquisition de connaissances (à partir de la performance, et non de la compétence) a été OBSERVER, utilisé par Otto Laske et Barry Truax à l'Université d'Utrecht de 1972 à 1977. Parmi les systèmes récents on peut citer PRECOMP (Laske & Don Cantor, 1987-89).

[39] Laske 1972b, p.361

[40] sans qu'il soit nécessaire d'émettre une hypothèse préalable sur l'universalité ou l'innéité de cette compétence.

[41] Kippen et Bel 1989b; 1989c; 1990.

système de transcription (voir le chapitre III) utilisant des chaînes de symboles.[42] Les données ainsi collectées sont en principe exemptes d'erreurs. Une partie importante de ce savoir musical, qui avait échappé à l'attention des ethnographes, est engrammée dans des **schémas d'improvisation** transmis oralement. Le "système expert" que nous avons réalisé pour modéliser ce savoir (***Bol Processor BP1***) est basé sur un formalisme emprunté aux grammaires formelles et aux langages de formes (*pattern languages*). (Voir le chapitre IV) Nous avons dû résoudre, pour cette classe de langages formels, le problème du **test d'appartenance** (*membership test*) d'une chaîne arbitraire au langage engendré par une grammaire. Un algorithme déterministe imposant des restrictions sur le formalisme grammatical (un ordre partiel sur les règles, autrement dit une connaissance procédurale) est décrit au chapitre IV. D'autres caractéristiques du formalisme imposées par le contexte expérimental, comme la négation de contexte ou le contrôle stochastique des productions, sont présentées au chapitre V.

Les problèmes d'**acquisition de connaissances** dans un univers incomplètement formalisé, et la **relation d'apprentissage** qui joue un rôle primordial dans le domaine étudié, justifient une étude rigoureuse des mécanismes d'**inférence inductive**. Les langages formels se prêtent bien à une telle approche. Des résultats théoriques sur les langages réguliers (de type 3) représentés par des grammaires hors-contexte (de type 2) sont exposés au chapitre VI. La méthode proposée pour l'apprentissage est originale au sens où elle permet d'inférer simultanément les connaissances syntaxiques et lexicales (et donc la segmentation des chaînes). A l'aide d'un exemple, nous montrons par ailleurs comment une théorie du domaine (en fait, un ensemble de connaissances présupposées sur l'accentuation des phrases musicales) peut être introduite empiriquement par un dialogue entre l'expert et la machine. L'avantage de cette méthode est de faire apparaître exactement, pour un problème d'apprentissage donné, la nature de la connaissance présupposée qui permet de réaliser une inférence correcte.

4. La composition musicale assistée par ordinateur (*computer-aided musical composition*)

Otto Laske s'est fait l'avocat de l'informatique appliquée à la création artistique en affirmant que les ordinateurs répondent à quatre besoins:[43]

> (1) Ils sont utiles pour créer de nouvelles niches physiques ou sociales (environnements de tâches, *task environments*) dans lesquels les artistes peuvent travailler, leur permettant ainsi de se distancier des approches conventionnelles.
>
> (2) Ils sont de remarquables outils de synthèse, capables de reconstituer ce qui a été mis en morceaux par l'analyse, qu'il s'agisse de matériaux ou de pensées.
>
> (3) Ils sont des outils de modélisation pleins de promesses, qui inspirent de nouvelles approches pour comprendre les processus artistiques, qu'ils permettent d'objectiver.
>
> (4) Ils aident à se débarrasser de nombreux pseudo-problèmes causés par le discours esthétique en langage naturel, et par là même des illusions de connaissance verbale.

Sous tous ces aspects, les ordinateurs aident à se concentrer sur l'*imaginable* (le possible) en contraste avec l'*existant*, et sont donc potentiellement libérateurs.

---

[42] Pour les mêmes raisons pragmatiques, Lerdahl et Jackendoff (1983) se sont d'abord intéressés aux formes monodiques de la musique tonale.

[43] Laske 1990a, p.3

Une discussion des stratégies compositionnelles — composition à partir de règles ou à partir d'exemples — n'entre pas dans le cadre de cette étude.[44] Une raison pragmatique de s'intéresser à la composition à partir de règles est qu'il est plus facile de traduire, dans un environnement informatique, un processus de prise de décision qu'un processus de transformation basé sur l'analyse d'œuvres existantes. Dans ce dernier cas se posent en effet les problèmes de la représentation des œuvres, de la modélisation de l'écoute, de la mémorisation, etc.[45]

Une application du *Bol Processor* à la composition assistée par ordinateur est la génération de propositions (des chaînes de symboles) qui sont ensuite traduites par le compositeur sous la forme de structures d'objets sonores. Cette fonction a été implémentée sur une nouvelle version (***Bol Processor* BP2**).[46] La production peut être contrôlée, étape par étape, par des décisions du compositeur, ou encore par des choix arbitraires de la machine, pondérés par les poids des règles. Nous avons aussi introduit une méthode simple de contrôle dynamique des pondérations qui permet à certaines règles de s'auto-renforcer ou de s'auto-inhiber. Nous avons, enfin, étendu considérablement le formalisme grammatical: contextes éloignés[47], substitutions contextuelles[48], grammaires programmées[49], etc.[50] Cette version ne comporte toutefois pas de test d'appartenance étant donné que le formalisme grammatical a été étendu à des classes de langages non décidables.

5. Le traitement du temps (*the processing of time*)

Le *Bol Processor* BP2 est capable de produire, non seulement des représentations symboliques discrètes de structures musicales, mais aussi leur “instanciation” sous la forme de structures finies d'“objets sonores”. Par analogie avec la linguistique, la composante syntaxique a été complétée par une **composante sonologique**. Or le traitement du temps en musique est plus complexe qu'en langage naturel. Différents systèmes musicaux utilisent différentes conceptions du temps (“lisse”, “strié”, etc.), de sorte que le temps métronomique “des physiciens” n'est en général pas adapté aux représentations musicales. Nous proposons une représentation du temps sur deux niveaux: **physique** et **symbolique**, la correspondance entre ces niveaux (la “**structure du temps**”) étant une fonction monotone croissante arbitraire. L'interprétation d'une proposition symbolique se fait en deux étapes:

1. L'attribution, à chaque symbole d'une proposition, d'une date symbolique. L'association d'un symbole et d'une date symbolique a été appelée un **objet temporel**. Les univers d'événements formés de séquences (**structures polymétriques**) peuvent être déterminés à partir de formules incomplètes grâce

[44] On peut se reporter, à ce sujet, à Laske1989a; 1990c; Smoliar 1990a-b; Lischka 1990. Sur la notion de “transformation créatrice”, voir aussi Blevis et al. 1990; Bel & Bel 1990.

[45] Voir Blevis et al. 1990 pour une tentative dans cette direction.

[46] BP1 était écrit en assembleur 6502 pour un Apple IIc. BP2 a été implémenté en C sur Macintosh, avec des procédures d'entrée-sortie sur une liaison de données MIDI.

[47] Voir Bel 1990b pour une application typique.

[48] Une extension des substitutions à longueur constante (cf. Allouche 1987, Allouche & Mouret 1988), qui englobe une classe d'automates cellulaires unidimensionnels.

[49] Salomaa 1973, p.161. Voir aussi Laske 1972b, p.365-ff, pour un formalisme permettant d'introduire un aspect procédural dans une grammaire générative.

[50] Ces extensions ne sont pas présentées dans cette étude. (Voir Bel 1990c)

à un algorithme d'interprétation qui détermine la structure en tenant compte du contexte. (Voir le chapitre VIII)

2. La **mise en temps** (“instanciation”) d'**objets sonores** définis par des **prototypes d'objets** — des séquences de messages commandant des “actions élémentaires”. L'instanciation d'un objet sonore est à la fois fonction de l'objet temporel qui le représente, de la correspondance entre temps symbolique et temps physique, et de propriétés métriques et topologiques associées à l'objet.

   La traduction d'une représentation symbolique en séquence d'actions élémentaires est un problème de satisfaction de contraintes. Nous avons mis au point pour cela un algorithme de complexité polynomiale (linéaire sur la longueur des séquences dans la majorité des cas) de mise en temps de structures polymétriques. La résolution du système de contraintes conduit en général à un ensemble infini de solutions qui comporte une partie “intéressante” finie, les **solutions canoniques**. L'utilisateur peut effectuer lui-même le choix d'une solution canonique, ou encore accepter la première solution proposée par le système.

Le chapitre VII introduit l'environnement de tâches du BP2 et les notions d'objets temporel/atemporel, de temps symbolique et de structure du temps. Le problème de la synchronisation d'un univers d'événements est posé de manière abstraite. Le chapitre VIII présente l'algorithme de résolution des formules polymétriques; un exemple en notation tonale est proposé en annexe 4. Le chapitre IX définit les objets sonores et présente l'algorithme de mise en temps. Des exemples de mise en temps de structures d'objets sonores sont traités en annexe 5.

Les lecteurs intéressés plus particulièrement par les questions de formalisme grammatical trouveront l'essentiel des propositions dans les chapitres III à V. Le chapitre VI, sur l'apprentissage inductif, peut être lu séparément. Les chapitres VII à X peuvent aussi être isolés dans la mesure où le traitement du temps, dans cette approche, est indépendant du formalisme grammatical.

II. Acquisition de connaissances en ethnographie et méthodologie “BP” (*knowledge acquisition in ethnography and the “BP” methodology*)

> L'ethnomusicologie a mis l'accent sur la nature symbolique de la musique et sur la variété des symboles qu'elle met en œuvre. Elle nous a ainsi éloignés des explications acoustiques plutôt crues du son musical, encore que ces dernières continuent à dominer la psychologie de la musique et une grande partie des travaux analytiques sur la musique.[51]

1. Collection ethnographique (*ethnographic collection*)

> L'ethnographie est une tentative *interprétative*, et les interprétations ethnographiques les plus précieuses produisent non seulement des informations indépendantes mais aussi des *perspectives sur ces informations* […] [52]

Les données ethnographiques sont essentiellement des rapports complétés par des documents sonores ou audiovisuels, des objets (instruments), etc. Le premier problème qui se pose en ethnomusicologie est celui de la transcription musicale. L'enregistrement sonore permet de mettre au point des systèmes de transcription. L'enregistrement vidéo permet en plus de conserver la trace de certains processus importants comme les techniques de jeu instrumental, et plus généralement des informations de contexte qui ont échappé à l'observation. Ces techniques interfèrent toutefois inévitablement avec le déroulement des sessions de travail dans la mesure où l'expert, s'il a conscience d'être enregistré ou filmé, adopte souvent un comportement ajusté aux circonstances.[53]

Dans l'ethnographie du *tabla*, instrument de percussion du nord de l'Inde, la transcription écrite est fiable à condition d'utiliser un système de notation **descriptive** comme par exemple celui proposé par Kippen.[54] Une telle transcription reste toutefois fastidieuse et doit se faire en temps différé à partir de pièces enregistrées ou mémorisées. Dans les deux cas, elle nécessite une excellente connaissance du système musical.[55] La technique de transcription utilisée dans les sessions de travail est donc en général limitée à la notation onomatopéique conventionnelle des percussionnistes (notation **prescriptive**).

De manière générale, l'ethnographie classique ne fournit donc que des informations incomplètes, souvent contradictoires et de fiabilité inconnue, soit sous forme verbale, soit

---

[51] Blacking 1989.

[52] James Peacock, cité par Kippen, 1990.

[53] En particulier, il estime souvent qu'il n'a plus droit à l'erreur, le document étant destiné aux générations futures.

[54] 1988, pp.xvi-xxiii.

[55] Les ouvrages de Gottlieb (1977) contiennent de nombreuses transcriptions, en partie erronées car réalisées sur la base d'enregistrements. Les musiciens manifestent peu d'enthousiasme à corriger de telles transcriptions, dont ils ne comprennent pas l'intérêt. La situation est différente lorsque l'analyse se entreprend de noter ou de mémoriser une pièce qui vient de lui être fournie par le musicien.

encore sous la forme d'exemples musicaux. La quasi absence de contrexemples est un obstacle majeur à la formulation de modèles.

Une amélioration de ce processus, l'**observation participante**, consiste à placer l'analyste dans une situation d'*apprenti*.[56] Elle présente toutefois l'inconvénient d'aboutir à une formulation du système musical tel qu'il est perçu par l'analyste à son niveau technique courant: l'expert adapte en effet le contenu de son enseignement aux préférences et aux connaissances de son élève, et le modèle construit dans ce contexte est plus celui d'une expérience (non renouvelable) d'acquisition du savoir, qu'une image représentative de ce savoir ou de ses modes de transmission. Une fois qu'il a dépassé un certain seuil d'expertise, l'analyste tend à perdre la capacité de formuler explicitement une partie des connaissances qu'il a intégrées, au risque de formuler des généralisations hâtives ou incontrôlables.

2. L'anthropologie dialectique (*dialectical anthropology*)

> […] to my mind, any community of musicological practice which excludes from consideration living musicians and restricts itself to accounts of frozen results of musical action, fails to be an inspiring community of inquiry about music.[57]

Avec l'observation participante l'ethnomusicologie est passée du niveau de la **connaissance empirique** à celui de la **connaissance rationnelle** des processus. Un troisième niveau (celui de la **connaissance scientifique**) consiste à formuler des hypothèses qui sont soumises à l'épreuve des faits. Ce niveau est très difficile à atteindre: l'analyste étant immergé dans un environnement d'enseignement normatif (caractéristique d'un enseignement traditionnel), il ne peut s'en tenir à une interprétation “objective” des faits. L'évaluation d'un modèle est encore plus difficile dans la mesure où, pour satisfaire une certaine objectivité scientifique, elle devrait faire appel à des procédures indépendantes des faits observés, ce qui présuppose une séparation (arbitraire) entre les faits qui servent à construire le modèle et ceux qui servent à l'évaluer.

On pourrait en conclure, de manière un peu hâtive, que les traditions orales ne se prêtent pas à l'expérimentation scientifique. Une version romantique de cette conclusion consisterait même à stipuler que toute démarche scientifique est le produit d'une culture scientifique (celle des pays industrialisés), et par là même ethnocentrique. Toutefois, les difficultés qui viennent d'être soulignées proviennent moins du schéma hypothèse-validation/réfutation que de l'impossibilité d'assigner des rôles à l'objet de l'étude (un système musical), à l'informateur (un ou plusieurs experts musiciens), et à l'analyste/expérimentateur.

---

[56] Emmet 1976, p.85

[57] Laske 1991c.

> [Les faits bruts sont moins significatifs que] […] la vision d'une existence (celle de l'autochtone) interprétée par les sensibilités de quelqu'un d'autre (l'ethnographe) dans le but d'informer et d'enrichir la compréhension d'un troisième individu (le lecteur ou l'auditeur).[58]

En réponse à cela, une méthode **dialectique** a été proposée par Blacking[59] et opérationnalisée par Kippen[60], selon laquelle *les modèles sont élaborés par les informateurs eux-mêmes et évalués par ces derniers*. On retrouve donc un schéma hypothèse-validation/réfutation, mais le rôle de l'analyste se limite en quelque sorte à celui de "catalyseur" d'un processus de modélisation par les acteurs.

On peut remarquer, de manière informelle, que le mode de raisonnement qui permet à un acteur de modéliser sa propre activité est plutôt de type *inductif* alors que celui d'un analyste extérieur au domaine s'appuie sur un modèle préexistant, par exemple une théorie étrangère servant de base de comparaison.[61] Par contre, la généralisation par induction opère de manière "aveugle" au sein d'un même modèle. C'est à ce titre qu'il est intéressant de faire intervenir l'informatique:

1) Elle impose un choix délibéré des techniques de représentation des faits et des procédures permettant d'agir sur cette collection de faits, i.e. d'élaborer une **base de connaissances**;
2) Elle permet un contrôle de la **cohérence** des modèles, et donc de la **validité des généralisations**.

La méthode dialectique ne présuppose pas une formalisation poussée du modèle. Tout dépend en fait de son utilisation: le modèle *informe* l'analyste et, en retour, l'informateur. En ce sens la méthode joue un rôle pédagogique non négligeable, instaurant une attitude réflexive de l'informateur sur son propre savoir autant que sur le processus de modélisation. Il est arrivé qu'un musicien nous déclare que le travail accompli en commun l'amenait à un nouvel "*insight*" sur sa tradition.[62]

Il est intéressant toutefois de constater que, dans deux univers musicaux apparemment distincts — les traditions orales et la création contemporaine —, la difficulté de verbaliser les mécanismes de perception, évaluation et production de la musique, contribue à renforcer une opinion courante selon laquelle l'activité musicale ne serait pas aussi systématique que les musiciens l'affirment, autrement dit que toute tentative de modélisation de cette activité serait vouée à l'échec. Une telle conception révèle en premier lieu une mécompréhension de la démarche scientifique expérimentale, où l'**échec** est une source d'enseignement à part entière; en second lieu elle encourage, dans les

---

[58] James Peacock, cité par Kippen 1990, p.48

[59] 1985. Le terme serait emprunté à Dorothy Emmet.

[60] 1985

[61] Les présupposés de l'analyste, issus de sa propre culture musicale, agissent souvent à son insu en arrière-plan. Voir par exemple les travaux de "musicologie comparative" sur la microtonalité, ou encore les conclusions hâtives de N.A. Jairazbhoy sur l'aspect "chaotique" de l'intonation des *rÆga*s (Bel 1988).

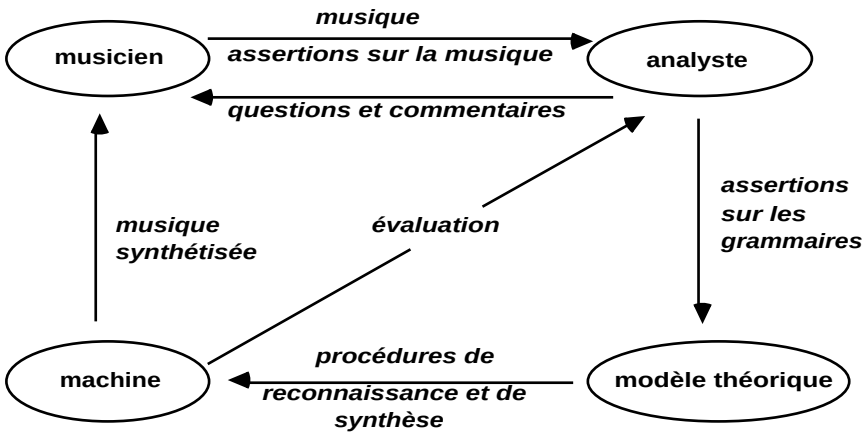
[62] On peut aussi renverser ce processus et partir d'un travail pédagogique orienté de manière réflexive. Dans ce cas, les élèves tiennent lieu d'informateurs et le pédagogue joue le rôle d'un "ethnologue". Un exemple typique de cette démarche est décrit par Marquès (1990). Par "modèle" il faut comprendre, dans ce contexte, un schéma conscient de traduction en mouvements des réactions à l'écoute musicale. Marquès demande aux élèves de mettre sur papier, de manière libre, leurs expériences. Dans un deuxième temps, elle leur propose de commenter l'évolution des expériences ainsi consignées.

sociétés traditionnelles, le reniement de méthodes pédagogiques hautement sophistiquées au profit de ce que nous appelons l'**apprentissage par imitation.**[63-64]

3. Acquisition de connaissances — la "méthodologie BP" (*knowledge acquisition — the "BP methodology"*)

La méthode dialectique a été mise en œuvre, pour la première fois dans un environnement de travail assisté par informatique, à l'occasion de travaux sur les schémas d'improvisation rythmiques des percussionnistes du nord de l'Inde.[65] Ces travaux ont été réalisés à l'aide du logiciel ***Bol Processor BP1***.[66] Il s'agit d'un système à règles de production implémenté dans un micro-ordinateur portable (Apple IIc) utilisé sur le terrain. Un "avantage" non négligeable de cet ordinateur est qu'il possède un écran plat très peu lisible, de sorte que les informateurs ne sont pas perturbés par la présence d'une machine et que la communication entre expert, analyste et machine se limite à l'échange oral conventionnel (le langage des ***bol***s, onomatopées utilisées par les percussionnistes).

La méthodologie de l'acquisition de connaissances avec le *Bol Processor* peut se résumer dans le schéma suivant:[67]



A partir d'informations fournies par l'expert, l'analyste élabore un premier modèle hypothétique du système musical étudié. Ce modèle est ensuite formalisé à l'aide de **règles de production** (grammaires formelles). Les règles sont activées par un **moteur d'inférences** pour générer des exemples de pièces musicales que l'on soumet à l'expert. Les exemples réfutés (**contrexemples**) sont revus par l'analyste qui modifie la grammaire en conséquence. Cette méthode produit en fait un nombre de contrexemples bien supérieur à celui que pourrait proposer l'analyste dans une situation d'apprentissage

---

63 Kippen 1988, pp.104-ff.

64 En ce qui concerne la composition contemporaine, la même attitude, qui nie l'existence d'un "raisonnement créateur" (Vecchione 1990b) tend à disqualifier la composition à partir de règles au profit de la composition à partir d'exemples. Voir à ce sujet Laske 1989a; 1990c; Smoliar 1990a-b.

65 Kippen 1985.

66 Kippen & Bel 1990.

67 Kippen & Bel 1989b.

humain. L'analyste est donc forcé, par ces essais infructueux, d'introduire dans la machine une connaissance présupposée du domaine (le résidu de son expérience antérieure), mais il le fait de manière pragmatique (à un bas niveau théorique).

Au bout d'un certain nombre de sessions de travail, la grammaire est en mesure de produire une majorité de pièces jugées correctes par l'expert. Une deuxième stratégie d'acquisition des connaissances est alors mise en œuvre: l'expert fournit des exemples qu'il soumet à l'évaluation de la machine. La plupart des exemples fournis dans ce contexte sont présumés corrects.[68] Cette méthode conduit l'analyste à formuler une généralisation du modèle; la pertinence de cette généralisation est évaluée en faisant alterner les deux méthodes.

Le schéma d'interaction expert-analyste-modèle-machine que nous venons de présenter a inspiré à Laske un schéma plus général d'activité musicale réflexive, "*The Musicological Hexagon*", où *musical work* désigne aussi bien une œuvre qu'une analyse en cours d'élaboration.[69]

4. La validation des modèles (*model assessment*)

Il n'y a pas de validation rigoureuse des grammaires dans la méthodologie BP. On considère une grammaire comme "correcte" si elle n'a pas été mise en défaut pendant un "certain" nombre de séances de travail.

La méthode dialectique en anthropologie ne présuppose pas — à l'inverse de la collection ethnographique classique — l'existence d'un corpus stable de connaissances. Un modèle de schéma d'improvisation représente donc un "segment de connaissance" fortement lié au contexte de la performance (le possesseur du savoir, son environnement social, l'époque et le lieu de l'expérimentation, etc.) et à son intentionnalité (la consigne convenue avec l'expert: démonstration, enseignement, archivage, etc.). En utilisant une modélisation rigoureuse on peut facilement détecter toute déviation par rapport au modèle hypothétique, soit pour le perfectionner, soit encore pour le réfuter en mettant en évidence une évolution du savoir ou l'effet d'une modification du contexte.[70]

Le mécanisme fondamental de l'acquisition de connaissances, dans la méthode dialectique, est donc la **mise en échec** (par les faits expérimentaux) d'un modèle hypothétique. La difficulté, comme dans toutes les sciences expérimentales, réside dans l'appréciation des causes de cet échec et l'élaboration d'une nouvelle théorie.

A titre d'exemple, Kippen avait élaboré, pour le *qa'ida* dont la grammaire est donnée en annexe 1, une grammaire simple qui reflétait bien la conception de l'expert en situation d'enseignement (Afaq Husain Khan). Quelques jours plus tard, le même expert décidait d'interpréter ce *qa'ida* en concert: la plupart des variations produites dans ce nouveau contexte étaient rejetées par la grammaire initiale.[71] En fait, l'interprétation dans ce cas faisait appel à un modèle plus complexe (décrit partiellement par la grammaire de l'annexe 1). Par contre, dans les situations d'enseignement ou de démonstration sur lesquelles ont

---

[68] Il n'est pas rare qu'un informateur fournisse aussi des exemples incorrects pour défier la machine.

[69] Laske 1991c, fig.9

[70] Cette précision du modèle est un handicap lorsque les données sont bruitées, comme c'est fréquemment le cas en reconnaissance de formes.

[71] On peut dire qu'un aspect esthétique de la performance consiste à pousser le système formel "hors de ses limites".

porté l'essentiel des travaux de Kippen, l'expert cherche délibérément à transmettre un modèle simple et cohérent de son savoir.

III. Transcription musicale et schémas d'improvisation (*musical transcription and improvisation schemata*)

Ce chapitre présente les principes de base de la transcription des pièces de percussion et la notion de “schéma d'improvisation” dans la rythmique du nord de l'Inde. L'idée de modéliser ces schémas d'improvisation à l'aide de grammaires formelles, et le développement d'un formalisme de représentation implémenté dans le *Bol Processor* BP1, sont issus de problèmes de transcription et d'archivage de pièces musicales.[72]

1. Transcription des pièces rythmiques (*transcribing rhythmic items*)

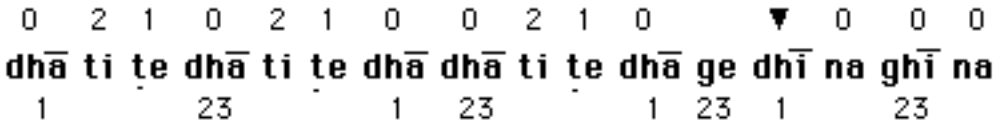
La notation des phrases du *tabla* (tambour accordé en deux parties) fait appel à un alphabet d'onomatopées: Vt = {*dhin, dha, ge, ne, na, ka, dhee, ke, tee, ta, ti, tira, kita,* etc…}, appelées ***bols***[73], utilisés pour la transmission du répertoire et, occasionnellement, en situation de concert. Les onomatopées peuvent désigner à la fois les gestes sur l'instrument de percussion et les sons qui en résultent.[74]

En transcription romane, Vt est en général un **code préfixe**. Autrement dit, toute chaîne de Vt* (ex: *dheenedheenagena*) peut être segmentée de manière unique de gauche à droite (ex:*dhee/ne/dhee/na/ge/na*). C'est pourquoi les phrases sont notées sans espaces séparateurs. Le symbole “-” est utilisé pour représenter les silences.

Il n'existe pas de correspondance biunivoque entre la représentation simplifiée (de nature **prescriptive** ou simplement **mnémonique**) de ces onomatopées et une **notation descriptive complète** comme par exemple celle de Kippen[75]. A titre d'exemple, la phrase que l'on écrit

dha te te dha te te dha dha te te dha ge dhee na ge na

en notation informatique, correspond aux syllabes et aux doigtés suivants:



[72] Kippen & Bel 1989c.

[73] Du verbe *bolna*, “parler” en hindi/ourdou.

[74] L'utilisation d'onomatopées est très largement répandue dans les cultures musicales traditionnelles, particulièrement en Afrique.

[75] 1988, pp.xvi-xxiii.

Les symboles de la ligne inférieure correspondent aux doigtés de la main gauche (tambour appelé *bayan*) et ceux de la ligne supérieure aux doigtés de la main droite (tambour *dahina*). Le “0” indique une résonnance entièrement libre du *dahina*, alors que le triangle indique une zone de frappe qui s'étend de la périphérie du *dahina* jusqu'à celle de la pastille noire (*sihayi*) placée au centre de ce tambour. Les nombres 1, 2 et 3 désignent respectivement l'index, le majeur et l'annulaire.

On constate, dans cet exemple, qu'il existe deux manières d'exécuter le même *bol*, “dha”, dépendantes à la fois du contexte (par exemple un changement de doigté est nécessaire pour deux “dha” consécutifs dans un tempo rapide) et du “sens” musical, particulièrement des accentuations.

En ce qui concerne la prononciation orale des *bol*s, dans l'exemple précédent les voyelles surlignées sont élonguées et le “†” est rétroflexe. De manière générale, en utilisant des signes diacritiques ou, comme les percussionnistes indiens, l'alphabet arabo-persan ou *devanagari* (sanskrit), on peut faire apparaître une correspondance biunivoque entre la notation des *bol*s et le langage oral utilisé par les percussionnistes.

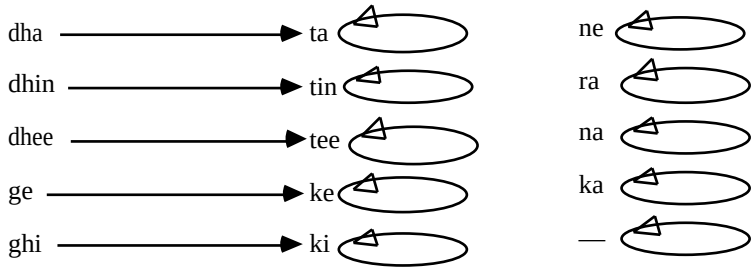
Les variantes phonologiques/orthographiques d'un *bol* (ex: “ti†e” devient “tete”) sont déterminées par son contexte et par la vitesse d'élocution, la prononciation rapide des phrases rythmiques étant considérée comme un exercice de virtuosité. A chaque classe de variantes on peut associer — pour une pièce musicale donnée — un représentant (en général la prononciation utilisée à vitesse lente). C'est ce représentant que nous utilisons en notation informatique.

Il est possible — bien que la résolution manuelle d'un tel problème s'avère difficile — de formaliser un système de transcodage (un ensemble de règles contextuelles) permettant de passer de la notation informatique à la notation descriptive complète et vice-versa. La recherche d'un tel système reste toutefois un problème marginal dans l'ethnographie du *tabla*. Le langage parlé, en effet, possède un statut autonome dans la mesure où il sert à véhiculer (comme à mémoriser) l'essentiel du répertoire musical. C'est donc exclusivement sur ce langage parlé (très proche de la notation informatique) qu'a porté notre étude.

2. Homomorphismes (*homomorphisms*)

Les sons du tabla sont résonnants, **ouverts** (*voiced, open*) ou amortis, **fermés** (*unvoiced, closed*). A tout *bol* représentant un son ouvert on peut faire correspondre un *bol* (unique) représentant un son fermé, par une transformation que les musiciens appellent “miroir”.[76] Etendue aux chaînes de *bol*s, l'application miroir peut être représentée par un endomorphisme non-effaçant (λ-*free homomorphism*) *mir* de Vt* dont la réduction à Vt est une application idempotente de Vt dans lui-même, par exemple:

[76] Le terme “miroir” est parfois utilisé en théorie des langages formels pour désigner ce que les musiciens appellent *rétrogradation*.



On note *ta* = mir(*dha*), et de même *teenakena* = mir(*dheenagena*).

3. Schéma d'improvisation et grammaticalité (*improvisation schemata and grammaticality*)

Nos recherches sur les langages de percussion ont porté plus particulièrement sur le ***qa'ida*** des tablistes (interprètes du *tabla*) du nord de l'Inde. "*Qa'ida*" est un mot du vocabulaire *ourdou*/persan/arabe qui signifie, dans ce contexte, "règles" ou "système". C'est aussi le pluriel du mot arabo-persan *qavaid* qui signifie "grammaire". On peut aussi traduire *qa'ida* par "**schéma d'improvisation**". Un *qa'ida* est en effet défini par un **thème** ([1] ci-dessous) et un ensemble incomplètement décrit de **variations** ([2] et suivantes ci-dessous):

[1]	**dhin--dhagena**	**dha--dhagena**	**dhatigegenaka**	**dheenedheenagena**
	tagetirakita	dhin--dhagena	dhatigegenaka	teeneteenakena
	tin--takena	**ta--takena**	**tatikekenaka**	**teeneteenakena**
	tagetirakita	dhin--dhagena	dhatigegenaka	dheenedheenagena

[2]	**dhin--dhatige**	**genakadhin--**	**tirakitatira**	**kitatirakita**
	tagetirakita	dhin--dhagena	dhatigegenaka	teeneteenakena
	tin--tatike	**kenakatin--**	**tirakitatira**	**kitatirakita**
	tagetirakita	dhin--dhagena	dhatigegenaka	dheenedheenagena

[3]	dhin--dhagena	dha--dhagena	dhatigegenaka	dheenedheenagena
	tagetirakita	dhin--dhagena	dhatigegenaka	teeneteenakena
	dhin--dhagena	**dha-dha-dha-**	dhatigegenaka	teeneteenakena
	tagetirakita	dhin--dhagena	dhatigegenaka	teeneteenakena
	tin--takena	ta--takena	tatikekenaka	teeneteenakena
	taketirakita	tin--takena	tatikekenaka	teeneteenakena
	dhin--dhagena	**dha-dha-dha-**	dhatigegenaka	teeneteenakena
	tagetirakita	dhin--dhagena	dhatigegenaka	dheenedheenagena

[4]	dhin--dhagena	dha--dhagena	dhatigegenaka	dheenedheenagena
	tagetirakita	dhin--dhagena	dhatigegenaka	teeneteenakena
	dheenedheenagena	**dheenedha-dheene**	dhatigegenaka	teeneteenakena
	tagetirakita	dhin--dhagena	dhatigegenaka	teeneteenakena
	tin--takena	ta--takena	tatikekenaka	teeneteenakena
	taketirakita	tin--takena	tatikekenaka	teeneteenakena
	dheenedheenagena	**dheenedha-dheene**	dhatigegenaka	teeneteenakena
	tagetirakita	dhin--dhagena	dhatigegenaka	dheenedheenagena

Dans les exemples ci-dessus on a utilisé des tabulations pour marquer les temps de la mesure.[77] Chaque temps contient six onomatopées; on appelle cet agencement “la vitesse six” (*chegun*).[78] Les phrases [1] et [2] comportent quatre lignes de quatre temps, ce qui coorespond à une mesure de 16 temps. Les variantes [2] à [4] durent 32 temps, soit deux mesures.[79]

Le problème posé ici consiste à **caractériser** l'ensemble des variations **correctes** du *qa‘ida* à partir d'un petit nombre de variations connues (rarement plus d'une vingtaine). On part de l'hypothèse que les musiciens sont capables de répondre de manière cohérente à la question de l'appartenance d'une chaîne arbitraire de *bol*s à un *qa‘ida* donné. Kippen a montré que cette hypothèse était valide pour certains musiciens et dans des protocoles particuliers de collection ethnographique. Dans ces conditions, la caractérisation d'un schéma d'improvisation est donc un problème de **grammaticalité** posé en termes de **syntaxe**.

Il est intéressant de noter (et certains musiciens comme Ilmas Husain Khan en font eux-mêmes la remarque) qu'il existe une similarité entre les structures du *qa‘ida* (le schéma thème+variations par excellence) et celles de la poésie en langue ourdou qui s'est développée à la même époque à Lucknow (18ème-20ème siècles). Le poème suivant de Ghalib[80] illustre cette similarité:

dil-e-nÆdÆn tujhe huÆ *kyÆ hai?*	O, cœur innocent, qu'es-tu devenu?
Ækhir is dard kí davÆ *kyÆ hai?*	Quel est le remède idéal de cette douleur?
ham hain mushtÆq aur voh bezÆr	Je suis ardent et pourtant il est insatisfait
yÆ ilÆhí yeh mÆjrÆ *kyÆ hai?*	Mon dieu, qu'est-ce que ce dilemne?

[77] La rythmique nord-indienne utilise un très grand nombre de mesures (*tala*) qui se caractérisent à la fois par leur nombre de temps (pas nécessairement entier) et une liste des temps accentués ou inhibés. Raja Chatrapati Singh de Bijna, un maître du *pakhawaj* (tambour horizontal), en a répertorié plus de 500. Les tablistes de Lucknow accordent beaucoup plus d'importance aux mesures “simples”, comme ici celle à 16 temps (*tintal*).

[78] La représentation de changements de tempo au sein d'une même pièce (par exemple de la vitesse 4 à la vitesse 8) sera abordée ultérieurement (voir chapitre V §4).

[79] Ce doublement de durée est une technique d'improvisation fréquente chez les musiciens de Lucknow. Cf. Kippen 1988, pp.166-7.

[80] Kippen & Bel 1989c.

IV. Grammaires BP (*BP grammars*)[81]

Nous désignons par "**grammaires BP**" une classe de grammaires formelles appartenant à la famille des systèmes de réécriture, permettant une représentation compréhensible de formes syntaxiques utilisées dans les "langages de percussions": contextes, répétitions, répétitions avec image homomorphique. Ces grammaires sont présentées ici comme une extension de la notion de **motif** (*pattern*) proposée par Angluin[82].

Pour ces grammaires, un algorithme de **test d'appartenance** (*membership test*) déterministe a été implémenté dans la première version du *Bol Processor* (BP1). Le test ne peut être appliqué que si les règles sont partiellement ordonnées. Cet ordre correspond à une heuristique de reconnaissance de formes ("la priorité aux plus grands agrégats de symboles") et peut être géré par le moteur d'inférences. En mode production les grammaires BP peuvent donc être considérées comme une forme de connaissance **déclarative**, alors qu'en analyse c'est l'aspect **procédural** qui est mis en évidence. L'algorithme déterministe de test d'appartenance permet ainsi, en suivant l'analyse pas à pas, de fournir une explication de l'échec et donc, le cas échéant, d'envisager une généralisation de la grammaire.

1. Aperçu historique (*historical survey*)

La conception "atomiste" de la musique — "la mélodie n'est autre chose qu'une succession de sons déterminés" (Vincent d'Indy) — est à l'origine du sérialisme de Schoenberg[83], suivi par Webern, puis par les tenants de l'approche stochastique:[84] Elisabeth Shannon[85], Meyer[86], Coons & Kraehenbuehl[87], Fuchs[88], Moles[89], Xenakis[90], Philippot[91], etc.

Une autre manière d'appréhender un idiome musical consiste à supposer l'existence de structures profondes, autrement dit un ensemble de relations syntagmatiques et

---

81 Les quatre premiers paragraphes de ce chapitre ont fait l'objet d'une communication au *Workshop on AI & Music, 11th International Joint Conference for Artificial Intelligence* (IJCAI). Voir Bel 1989a. Les paragraphes suivants ont été exposés au 6ème Congrès de l'AFCET — *Reconnaissance des Formes et Intelligence Artificielle* (Bel 1987b). Une version anglaise simplifiée (Bel 1991b) sera publiée.

82 1980b

83 Voir Chemillier 1990b pour un commentaire sur l'aspect formel de la théorie sérielle.

84 Références données par Vecchione.

85 1951

86 1956; 1957

87 1958

88 1961

89 1958

90 1963

91 1970

paradigmatiques qui permettent de caractériser les productions d'un idiome musical lorsque celles-ci sont représentées par des structures discrètes.

L'idée de formaliser une “syntaxe musicale” est très ancienne. On peut citer — parmi les modèles qui n'utilisent pas le formalisme des règles de réécriture — le Jeu de dés musical (*Musikalisches Würfelspiel*) de W.A. Mozart, puis, au 20ème siècle, les travaux de Schenker[92] qui servent encore de substrat à de nombreuses approches théoriques. La **théorie générative de la musique tonale**[93] est une synthèse de travaux en linguistique formelle, en psychologie et en musicologie: partant d'une monodie tonale, le modèle permet de déterminer à la fois une structure de surface (à l'aide de règles de structure métrique et de règles de regroupement, ces dernières étant empruntées à la psychologie gestaltiste) et une structure profonde (réduction prolongationnelle Schenkerienne et structure prosodique inspirée de la théorie des rythmes de Cooper et Meyer[94]). Le terme “génératif”, dans ce contexte, se réfère donc à l'**analyse** d'une pièce unique, et non à la production musicale d'un ensemble de variantes:

> Our theory of music is therefore base on structural considerations; it reflects the importance of structure by concerning itself not with the composition of pieces but with assigning structures to already existing pieces.[95]

Pour une œuvre donnée il existe plusieurs analyses possibles. Dans un deuxième temps on fait donc appel à des règles d'un ordre supérieur (règles de préférence[96]), tenant compte des consonances/dissonances, cadences tonales, etc., pour déterminer la *meilleure* analyse.[97]

La théorie de Lerdahl et Jackendoff, que les auteurs ont plus tard commencé à étendre à la musique atonale, reste une des tentatives les plus achevées de modélisation de la musique tonale.[98] Elle est à la source de nombreux travaux actuels, certains visant à évaluer expérimentalement sa validité psychologique, d'autres à concevoir des systèmes experts pour la composition[99] ou l'analyse[100]. Une critique et une extension de la théorie générative ont été formulées par Célestin Deliège[101].

La recherche de grammaires formelles servant à modéliser une **activité compositionnelle** (ou improvisationnelle) a suscité quelques travaux dans le domaine de l'ethnomusicologie.[102] Feld s'est élevé toutefois contre ce qu'il appelle “la coquille vide du formalisme” (*the hollow shell of formalism*) dans les emprunts de la musicologie à la linguistique générative. A propos des travaux de Lidov, il écrit notamment:

---

92 1935

93 Jackendoff & Lerdahl 1982, pp.92-ff; Lerdahl & Jackendoff 1983.

94 1960

95 Jackendoff & Lerdahl 1982, p.85

96 *op.cit.* pp.99-102.

97 Reprenant ainsi l'hypothèse Chomskienne d'un “sujet idéal”.

98 C'est aussi à ce travail sur la musique que l'on doit l'introduction des règles de préférence en linguistique formelle.

99 Par exemple, Camilleri 1984.

100 Notamment pour simuler la perception du rythme: voir Jones et al. 1990.

101 1983; 1984.

102 Laloum & Rouget 1965; Lidov 1972; etc., cités par Feld 1974.

> […] I find the approach absurdly formal and totally pretentious, not because I can't appreciate his mathematics, but because he purports to do a theoretical task and then says nothing about what kind of ethnomusicological theory he is talking about. Yet worse, the music is considered as nothing more than one dimensional transcriptions — a set of abstractions which a mathematician may retranslate into other abstractions of another logical order. This method of analysis is based on the completely false assumption that transcriptions of music have some sort of objective reality [,…][103]

Dans le même article, Feld cite Blacking pour ce qu'on pourrait appeler un fondement épistémologique de l'ethnomusicologie scientifique:

> Analytic tools cannot be borrowed freely and used as short cuts to greater achievements in ethnomusicological research as can electronic devices such as the tape recorder: they must emerge from the nature of the subject studied.[104]

Un problème essentiel des théories de la performance ou de la compétence est celui de leur capacité explicative (*explanatory adequacy*). Selon Laske[105], une théorie de la performance peut être:

— adéquate pour les observations (*observationally adequate*) si elle donne un compte-rendu compréhensible des données;

— descriptivement adéquate (*descriptively adequate*) si la sortie est conçue en accord avec une condition de bonne formation (*a well-formedness condition*), le cas échéant, avec une mesure de la grammaticalité;

— explicatoire (*explanatory adequate*) si et seulement si elle est capable de définir la structure interne de la performance en relation avec les règles de compétence dont cette performance dépend — au moins partiellement.

Une grammaire formelle qui, pour un corpus (fini) de productions donné, permet de décider si une production arbitraire appartient ou non au corpus, n'a pas de capacité explicatoire. Pour accéder au niveau de l'adéquation explicatoire il faut en fait disposer d'un modèle susceptible d'engendrer de nouvelles productions correctes ou de prédire l'appartenance d'une production au langage. Un tel modèle doit donc se prêter à la généralisation, opération que l'analyste peut tenter d'effectuer lui-même si la représentation est assez explicite.[106]

2. Grammaires de motifs (*pattern grammars*)

2.1 Définitions et notations (*definitions and notations*)

Soient Vt un alphabet terminal fini de **symboles terminaux**, et Vn un alphabet dénombrable de symboles non-terminaux (des **variables**). Si l'on note $\mathcal{H}$ l'ensemble des homomorphismes non-effaçants pour la concaténation (λ-*free homomorphisms*) de (Vn ∪ Vt)* sur lui-même, tout élément de $\mathcal{H}$ dont la restriction à Vt est l'application

---

[103] Feld 1974, pp.209-10.

[104] Blacking 1972, p.1. Cité par Feld 1974.

[105] 1972a, p.5. Laske reprend ici une classification des théories de la compétence proposée par Chomsky (1964).

[106] Une autre approche consiste à implémenter une technique d'inférence inductive. (Voir chapitre VI)

identique est appelé une **substitution**. Si l'ensemble image d'une substitution est Vt*, cette substitution est dite **terminale**. Toute substitution dont la restriction à Vn est une bijection de Vn dans Vn est un **réétiquetage de variables** (*renaming of variables*).

On appelle **motif** (*pattern*) tout élément de (Vn ∪ Vt)*. Si *p* est un motif et *s* une substitution, alors s(p) est appelé une **dérivation** de *p*. Un motif qui ne contient aucune variable est appelé **dérivation terminale** ou **phrase**. Deux motifs, *p* et *q*, sont **équivalents** (on note p ≈ q) si et seulement s'il existe un réétiquetage de variables *r* tel que p = r(q). Une autre relation binaire (notée p ≤ q) est définie de la manière suivante: *p* est **moins général que** (ou **plus spécifique que**) *q* si et seulement si pour une substitution *s*, p = s(q). Puisque la substitution est un homomorphisme non effaçant, p ≤ q => |p| ≥ |q|.

Le **langage** généré par le motif *p* est l'ensemble des dérivations terminales de *p*, à savoir L(p) = {s ∈ Vt^+ : s ≤ p}. On peut prouver que:

≤ est transitive
p ≤ q => L(q) ⊇ L(p)
p ≈ q <=> p ≤ q et q ≤ p

Toutefois, la question de trouver une procédure effective pour décider si L(q) ⊇ L(p), étant donnés deux motifs arbitraires *p* et *q*, semble être ouverte.[107]

2.2 Application à un exemple musical (*musical example*)

Les ensembles de variations du *qa'ida* présenté au chapitre III §2 peuvent être, en première analyse, décrits à l'aide des motifs *p1* et *p2* suivants, avec les variables X, Y et Z:

Variations simples:

p1 = X tagetirakitadhin--dhagenadhatigegenakateeneteenakena
Y tagetirakitadhin--dhagenadhatigegenakadheenedheenagena

Variations doubles:

p2 = dhin--dhagenadha--dhagenadhatigegenakadheenedheenagena
tagetirakitadhin--dhagenadhatigegenakateeneteenakena
Z dhatigegenakateeneteenakena
tagetirakitadhin--dhagenadhatigegenakateeneteenakena
tin--takenata--takenatatikekenakateeneteenakena
taketirakitatin--takenatatikekenakateeneteenakena
Z dhatigegenakateeneteenakena
tagetirakitadhin--dhagenadhatigegenakadheenedheenagena

Soit L le langage représentant toutes les variations acceptables du *qa'ida*, L1 le sous-ensemble des variations simples et L2 celui des variations doubles. On a évidemment: L = L1 ∪ L2. L(p1) et L(p2) sont les langages générés respectivement par *p1* et *p2*. Admettons (sur la base d'une étude expérimentale) que L(p1) et L(p2) contiennent des sous-ensembles non triviaux de L1 et L2.

On cherche, de manière idéale, des motifs *descriptifs* de L1 et L2. Un motif *d* est **descriptif** d'un langage L si L(d) ⊇ L et si pour tout motif *q* tel que L(q) ⊇ L, L(q) n'est pas strictement contenu dans L(d).

L est en fait un ensemble fini. Par exemple, dans ce *qa'ida*, toutes les chaînes acceptables de Vt* ont des longueurs inférieures ou égales à 32 x 6 = 192 symboles, ce

[107] Angluin 1980b, pp.49-52.

qui permet de calculer un majorant de card(L).[108] On ne connaît toutefois de L qu'un ensemble S^+ d'**exemples** (*positive instances*) et un ensemble S^- de **contrexemples** (*negative instances*). On appelle présentation l'ensemble $S = S^+ \cup S^-$. On dit qu'un motif *p* est **compatible avec** (*matches*) une présentation si et seulement si *p* est descriptif de S^+ et que $S^- \cap L(p) = \varnothing$. Le motif *p* est **ajusté sur** S (*tight fit*) si $L(p) = S^+$. De manière évidente, tout motif ajusté sur S est descriptif de S^+ et compatible avec S.

Il existe une procédure effective pour inférer un motif descriptif à partir d'un ensemble d'exemples S^+, mais le problème est NP-difficile dans la plupart des cas.[109] De plus, la classe des langages de motifs n'est pas fermée pour la plupart des opérations ensemblistes de base: union, complément et intersection. En sorte qu'il n'est pas facile de construire de manière systématique une description par motifs d'un langage formel.

2.3 Langages de motifs restreints (*restricted pattern languages*, RPL)

Etant donné un motif *p*, toute sous-classe de L(p) peut être définie en contraignant les dérivations acceptables de p. Par exemple, on peut définir la sous-classe de longueur *n*: $L_n(p) = \{s \in Vt^+ : s \leq p \text{ et } |s| = n\}$. Dans l'exemple traité, évidemment |X| = |Y| = 24 et |Z| = 12. De plus on peut écrire Y = mir(X).

Une autre manière de formuler des contraintes consiste à utiliser des **règles de réécriture**. Nous notons

$$p \longrightarrow q$$

pour indiquer que toute dérivation acceptable de *q* est une dérivation acceptable de *p*, autrement dit $q \leq p$. Si l'on note $\mathcal{S}$ l'ensemble de toutes les substitutions, $p \longrightarrow q$ dénote le sous-ensemble

$$\mathcal{S}_{p\to q} \quad \text{tel que} \quad \forall s \in \mathcal{S}_{p\to q},\ s(p) = q.$$

Puisque le problème de décider si $p \leq q$ pour deux motifs arbitraires p et q est NP-complet,[110] on doit se contenter de **règles bien formées**:

Définition:

> La règle $p \longrightarrow q$ est bien formée si et si seulement si $p \in Vn^+$, $q \in (Vt \cup Vn)^+$, $|q| \geq |p|$, et aucune variable n'apparaît plus d'une fois dans p.

Théorème:

> Si la règle $p \longrightarrow q$ est bien formée alors $L(p) \supseteq L(q)$.

Preuve:

Il suffit de trouver une substitution *s* telle que q = s(p), ce qui entraîne $q \leq p$. Une procédure pour construire *s* est la suivante:

---

[108] Le nombre de phrases de longueur inférieure ou égale à *n* sur un alphabet de cardinal k > 1 est en effet:

$$\frac{k \cdot (k^n - 1)}{k - 1}$$

[109] Angluin 1980b, pp.54-55.

[110] *Op.cit.* p.52

(1) remplacer les |p|-1 variables les plus à gauche (leftmost) de *p* par les |p|-1 variables les plus à gauche de *q*;

(2) remplacer la variable la plus à droite dans *p* par la chaîne formée des |q|-|p|+1 symboles les plus à droite (rightmost) de *q*.

Puisqu'aucune variable ne figure deux fois dans *p*, il n'existe aucune contrainte sur *q* qui rende impossibles ces remplacements.[111] Par exemple, une substitution de XYZ produisant abcX serait {X/a, Y/b, Z/cX}.■

Soit L(p) un langage de motif et $\mathcal{R}$ un ensemble de règles bien formées. Toute règle (p —> q) définit un ensemble de substitutions $\mathcal{S}_{p->q}$. Pour construire une phrase du **langage de motif restreint** $L_{\mathcal{R}}(p)$ on procède comme suit:

(a) sélectionner un sous-ensemble $\mathcal{R}_o$ de $\mathcal{R}$;

(b) soit $\mathcal{S}_o = \cap_i (\mathcal{S}_{pi->qi})$ tel que $(p_i \text{ —> } q_i) \in \mathcal{R}_o$;

(c) si $\mathcal{S}_o \neq \varnothing$ et $\mathcal{S}_o$ est fini, alors $\mathcal{S}_o = \{s\}$ tel que la phrase est w = s(p).

La sélection de $\mathcal{R}_o$ appelle certains commentaires:

(1) Si une variable X n'apparaît dans la partie gauche d'aucune règle sélectionnée, alors elle peut être remplacée par n'importe quel $q \in (Vt \cup Vn)^+$. Dans ce cas, $\mathcal{S}_o$ est infini et par conséquent $L_{\mathcal{R}}(p) = \varnothing$.

(2) Il existe des sous-ensembles de $\mathcal{R}$ qui donnent $\mathcal{S}_o = \varnothing$, autrement dit des **dérivations avortées**. Par exemple, si une variable X apparaît dans le membre gauche de deux règles distinctes de $\mathcal{R}_o$, par exemple X —> q_i et X —> q_j, alors $\mathcal{S}_{X->qi} \cap \mathcal{S}_{X->qj} = \varnothing$, et donc $\mathcal{S}_o = \varnothing$.

(3) Dans tous les cas autres que (1) et (2), toute variable apparaît dans un motif avec une dérivation unique, de sorte que $\text{card}(\mathcal{S}_o) = 1$.

L'exemple suivant est destiné à clarifier les points (2) et (3). Un RPL pour les variations doubles serait $L_{\mathcal{R}}(p2)$ avec:

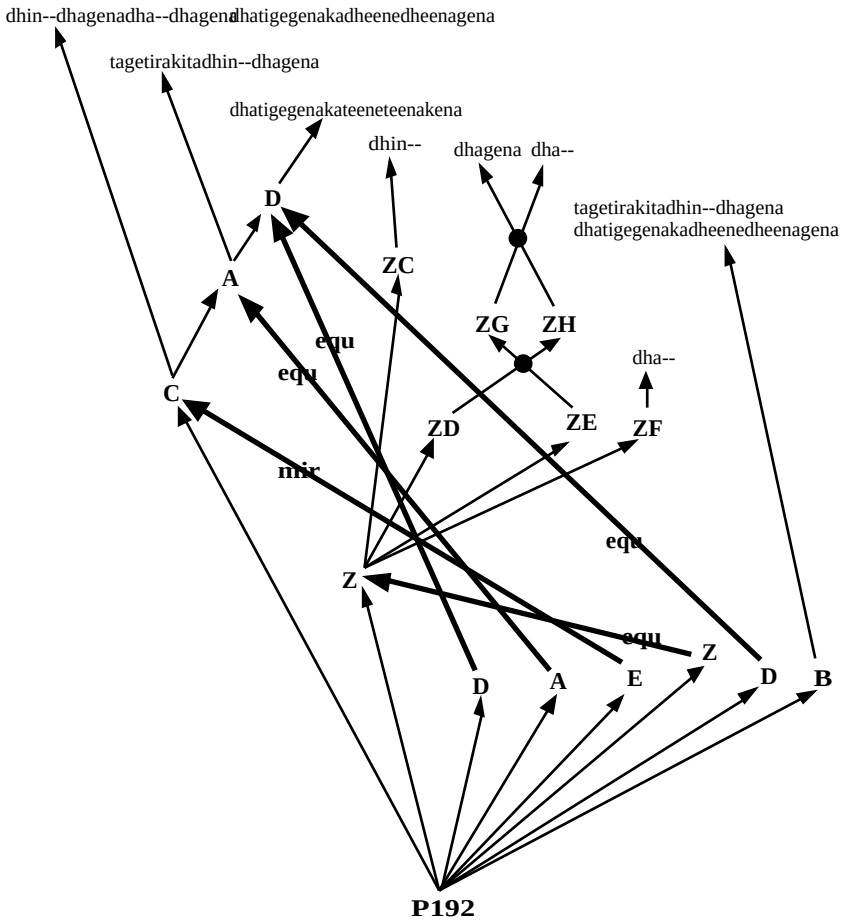
p2 = P192 et $\mathcal{R}$ est l'ensemble de règles:
(1) P192 —> C Z D A E Z D B avec E = mir(C)
(2) A —> tagetirakitadhin--dhagena D
(3) B —> tagetirakitadhin--dhagenadhatigegenakadheenedheenagena
(4) C —> dhin--dhagenadha--dhagenadhatigegenakadheenedheenagena A
(5) D —> dhatigegenakateeneteenakena
(6) Z —> tirakitatirakitatirakita (7) Z —> ZA ZB (8) Z —> ZC ZD ZE ZF
(9) ZC ZD —> ZG ZH (10) ZD ZE —> ZG ZH (11) ZE ZF —> ZG ZH
(12) ZG ZH —> dhagenadhin-- (13) ZG ZH —> dhagenadha--
(14) ZC —> dhin-- (15) ZD —> dhin-- (16) ZE —> dhin-- (17) ZF —> dhin--
(18) ZC —> dha-- (19) ZD —> dha-- (20) ZE —> dha-- (21) ZF —> dha--
(22) ZA —> dheenedheenedheene (23) ZB —> dheenedheenedheene
(24) ZA —> dha-dha-dha- (25) ZB —> dha-dha-dha-

Si l'on choisit $\mathcal{R}_o$ ={1,2,3,4,5,8,10,13,14,21}, on obtient la phrase:

[111] Cette procédure est inspirée de l'algorithme de Makanin (Makanin 1977, Roussel 1987).

dhin--dhagena dha--dhagena dhatigegenaka dheenedheenagena **C**
tagetirakita dhin--dhagena dhatigegenaka teeneteenakena
dhin-- dhagena dha--dha-- dhatigegenaka teeneteenakena **Z D**
tagetirakita dhin--dhagena dhatigegenaka teeneteenakena **A**
tin--takena ta--takena tatikekenaka teeneteenakena **E**
taketirakita tin--takena tatikekenaka teeneteenakena
dhin--dhagena dha--dha-- dhatigegenaka teeneteenakena **Z D**
tagetirakita dhin--dhagena dhatigegenaka dheenedheenagena **B**

qui peut être représentée par le graphe syntaxique (sensible au contexte) suivant, où ‘equ’ et ‘mir’ indiquent les structures de répétition stricte et de répétition avec miroir:



Les ensembles de substitutions des règles 10 et 14 peuvent être représentés par:

$$\mathcal{S}_{p10->q10} = \{..., ZD\ ZE/ZG\ ZH, ...\}$$

et

$$\mathcal{S}_{p14->q14} = \{..., ZC/dhin\text{-}\text{-}\ ,...\}$$

où ‘...’ indique l'ensemble (dénombrable) de toutes les substitutions possibles des chaînes de $(Vt \cup Vn)^+$ à l'exception de celles qui ont déjà été spécifiées (ZD ZE et ZC). On voit que

$$\mathcal{S}_{p10->q10} \cap \mathcal{S}_{p14->q14} = \{..., ZD\ ZE/ZG\ ZH, ...\ , ZC/dhin\text{-}\text{-}\ ,...\}.$$

Le résultat des intersections est:

$\mathcal{S}_o$ = {P192/C Z D A E Z D B, A/tagetirakitadhin--dhagena D,
B/tagetirakitadhin--dhagenadhatigegenakadheenedheenagena,
C/dhin--dhagenadha--dhagenadhatigegenakadheenedheenagena A, D/dhatigegenakateeneteenakena,
Z/ZC ZD ZE ZF, ZC/dhin--, ZD ZE/ZG ZH, ZF/dha--, ZG ZH/dhagenadha--}

où chaque dérivation est explicite. Par conséquent, $\mathcal{S}_o$ contient une substitution unique qui produit la phrase considérée.

Si l'on sélectionne $\mathcal{R}_o$ = {1,2,3,4,5,8,10,13,14,15,21} on obtient $\mathcal{S}_{p10->q10}$ = {..., ZD ZE/ZG ZH, ...}, $\mathcal{S}_{p15->q15}$ = {..., ZD/*dhin*-- ,...}, et $\mathcal{S}_{p20->q20}$ = {..., ZE/*dha*-- ,...}. Appelons $\mathcal{S}_{15,20} = \mathcal{S}_{p15->q15} \cap \mathcal{S}_{p20->q20}$ = {..., ZD/*dhin*-- ,..., ZE/*dha*-- ,...}. Pour toute substitution $s \in \mathcal{S}_{15,20}$, s(ZD ZE) = s(ZD) s(ZE) = *dhin--dha--*. Par conséquent, ZD ZE ne peut jamais être substitué à ZG ZH, ce qui contredit $\mathcal{S}_{p10->q10}$. Dans ce cas, $\mathcal{S}_o = \varnothing$ et la dérivation est avortée.

Remarque: on pourrait aussi dire de $\mathcal{S}_o$ qu'elle est l'expression conjonctive maximale (donc la plus spécifique) utilisant les prédicats ($p_i = q_i$) tels que (p_i —> q_i) $\in \mathcal{R}$.

Théorème:

La classe des langages finis coincide exactement avec celle des RPLs.

Preuve:

Tout sous-ensemble de l'ensemble (fini) de règles dans un RPL produit au plus une dérivation terminale. L'ensemble des dérivations terminales est donc fini. Réciproquement, pour tout langage fini L il existe une grammaire linéaire à droite sans règle récursive (*a non-imbedding right-linear grammar*) qui génère exactement L. Soit G = (Vt, Vn, S, $\mathcal{R}$) où S est le symbole de départ et $\mathcal{R}$ un ensemble fini de règles de format A —> a ou A —> aB tel que A,B $\in$ Vn, A≠B, et a $\in$ Vt. Il est facile de montrer que G engendre exactement le RPL $L_{\mathcal{R}}(p)$ tel que p = S, ce qui complète la preuve.■

Corollaire:

La classe des RPL est récursive et fermée pour l'union, la concaténation et l'intersection.

On construit une procédure effective pour construire $L_{\mathcal{R}}(p) = L_{\mathcal{R}1}(p1) \cup L_{\mathcal{R}2}(p2)$ de la manière suivante:

Supposons que $L_{\mathcal{R}1}(p1)$ et $L_{\mathcal{R}2}(p2)$ sont définis avec:

p1 = S1 et $\mathcal{R}1$ = {S1 —> q1, ...}

p2 = S2 et $\mathcal{R}2$ = {S2 —> q2, ...} dans lequel on a renommé les variables de $\mathcal{R}2$ de sorte qu'aucune variable n'apparaisse à la fois dans $\mathcal{R}1$ et dans $\mathcal{R}2$. On construit $L_{\mathcal{R}}(p)$ défini avec:

p = S et $\mathcal{R}$ = {S —> S1, S —> S2} $\cup$ $\mathcal{R}1$ $\cup$ $\mathcal{R}2$.

L'ensemble $L_{\mathcal{R}}(p)$ des dérivations terminales de S est l'union des ensembles de dérivations terminales de S1 et S2, par conséquent $L_{\mathcal{R}}(p) = L_{\mathcal{R}1}(p1) \cup L_{\mathcal{R}2}(p2)$. De plus, puisque $\mathcal{R}1$ et $\mathcal{R}2$ ne contiennent que des règles bien formées, $\mathcal{R}$ ne contient aussi que des règles bien formées. Par conséquent, $L_{\mathcal{R}}(p)$ est un RPL.

Les RPLs forment une classe de langages formels fermée sur l'union, et il existe une procédure permettant de construire un langage à partir de ses sous-ensembles. La classe

est par ailleurs récursive et par conséquent le test d'appartenance est algorithmique. Ces propriétés sont intéressantes pour construire un modèle de représentation qui permette d'opérer des généralisations descriptives.

3. Grammaires BP (*BP grammars*)

3.1 Règles de motifs (*pattern rules*)

Les règles de réécriture du §2.3 peuvent être remplacées par des **règles de motifs**:

P96 —> X A Y B *avec* mirror(X,Y) et longueur(X,24)
P192 —> C Z1 D A E Z2 D B *avec* equal(Z2,Z1) et longueur(Z1,12) et mirror(C,E)

Ces règles comportent une partie de réécriture classique et des prédicats *mirror*, *longueur* et *equal* qui expriment des contraintes.

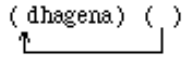
Il est clair que si deux variables liées par une prédicat *equal/mirror* apparaissent dans la partie droite d'une règle, la variable la plus à gauche dénote une chaîne qui sert de référence, et l'autre une copie de cette référence. Par exemple, avec les règles suivantes:

A —> B C where equal(B,C)
B —> dhagena

la dérivation terminale *dhagenadhagena* peut être notée

(= dhagena) (: dhagena)

où la première parenthèse (marquée avec "=") sert de référence (**maître**), et la suivante (marquée avec ":") de copie (**esclave**). Dans un implémentation informatique, la seconde parenthèse contient en fait un pointeur vers la référence:



La même convention d'écriture est utilisée dans les règles, par exemple:

A —> (= B) (: B)

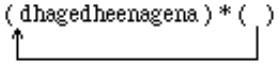
Pour ce qui est des homomorphismes, on utilise un symbole réservé pour indiquer que le contenu de la parenthèse suivante (maître ou esclave) est une image homomorphique. Par exemple, le miroir des langages de percussion est noté "*". Etant donné l'homomorphisme miroir défini au chapitre III §2, dans la grammaire suivante

S —> (= D) * (: D)
D —> dhagedheenagena

l'unique dérivation terminale est

(= dhagedheenagena) * (: taketeenakena)

et sa représentation interne:



A l'aide de ces conventions il est possible de simplifier le formalisme de représentation des RPLs:

(1) Si une variable apparaît plusieurs fois dans une règle de RPL, elle doit être indiquée avec des marqueurs de répétition. Par exemple,
A —> B C B C B doit s'écrire: A —> (=B) (=C) (:B) (:C) (:B).

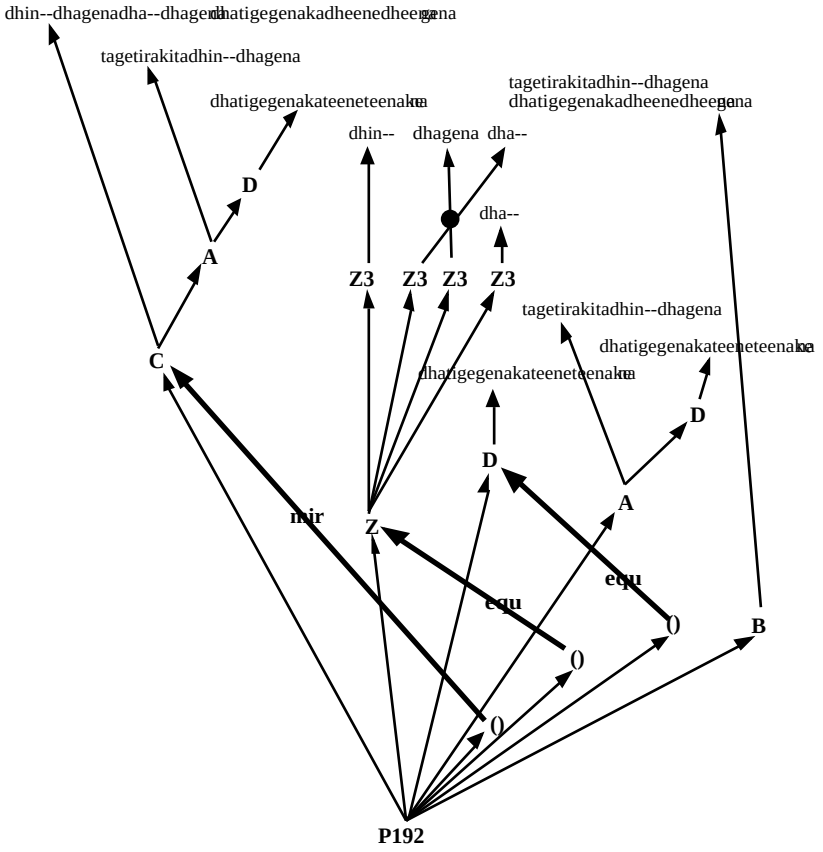
(2) La même variable peut apparaître deux fois dans un membre gauche de règle (si ces deux occurences ne sont pas liées par un marqueur de répétition). Ainsi, la règle A A —> q est correcte, mais pas la règle (=A) (:A) —> q.

(3) Les miroirs sont représentés par un astérisque.

Avec ces conventions, la grammaire BP générant le langage RPL défini au §2.3 est:

(1) P192 —> (=C) (=Z) (=D) A * (:C) (:Z) (:D) B
(2) A —> tagetirakitadhin--dhagena D
(3) B —> tagetirakitadhin--dhagenadhatigegenakadheenedheenagena
(4) C —> dhin--dhagenadha--dhagenadhatigegenakadheenedheenagena A
(5) D —> dhatigegenakateeneteenakena
(6) Z —> tirakitatirakitatirakita
(7) Z —> Z6 Z6
(8) Z —> Z3 Z3 Z3 Z3
(12) Z3 Z3 —> dhagenadhin--
(13) Z3 Z3 —> dhagenadha--
(14) Z3 —> dhin--
(18) Z3 —> dha--
(22) Z6 —> dheenedheenedheene
(24) Z6 —> dha-dha-dha-

dans laquelle les règles 9, 10, 11, 15, 16, 17, 20, 21 et 23 ont été supprimées. Le graphe syntaxique de la phrase générée au §2.3 est maintenant:

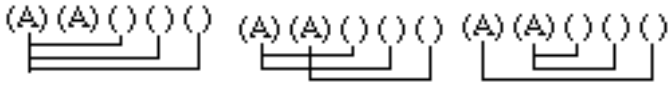


3.2 Affectation maître-esclave (*master-slave assignment*)

La représentation de motifs définie au §3.1 peut être ambiguë. Par exemple, l'expression

(= A) (= A) (: A) (:A) (:A)

peut représenter diverses affectations de pointeurs:



etc…

Seule la première affectation est prise en compte dans la version BP2 du *Bol Processor*. Dans la version BP1, toutes les affectations sont réalisables au moyen de l'éditeur. Dans les deux cas, l'ambiguïté est levée même si l'affectation réalisée n'est pas explicitement visible à l'écran.

L'affectation d'une parenthèse est caractérisée par un pointeur (un entier positif ou nul) *i*, que nous appelons **marqueur de parenthèse**, calculé de la manière suivante:

1) Parenthèse maître: i = 0 ("**marqueur zéro**")
2) Parenthèse esclave: (i-1) est le nombre de marqueurs zéro qui séparent la parenthèse esclave de la parenthèse maître à laquelle elle est rattachée.

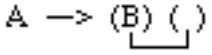
Les valeurs de *i* pour les parenthèses esclaves des trois affectations précédentes sont les suivantes:

(=A) (=A) (2) (2) (2) (=A) (=A) (2) (2) (1) (=A) (=A) (1) (1) (2) .

Nous montrons maintenant comment ces affectations sont recalculées lors des réécritures. Supposons qu'à l'expression précédente soit appliquée la règle:

A —> (= B) (:B)

à la deuxième position de dérivation, i.e. sur la deuxième occurrence de A. La règle s'écrit



ou encore:

A —> (=B) (1)

Après réécriture on obtient:

(=A) (=(=B) (:B)) (:A) (:A) (:A)

pour la première affectation, ou

(=A) (=(=B) (:B)) (:A) (:A) (:(=B) (:B))

pour la seconde, ou enfin

(=A) (=(=B) (:B)) (:(=B) (:B)) (:(=B) (:B)) (:A)

pour la troisième.

Ces formules se représentent plus simplement avec les marqueurs, respectivement:

(=A) (=(=B) (1)) (**3**) (**3**) (**3**) (=A) (=(=B) (1)) (**3**) (**3**) (**2**) (=A) (=(=B) (1)) (**2**) (**2**) (**3**)

Il est facile de voir que les marqueurs indiqués en gras ont été incrémentés d'une unité après le remplacement de A par (=B) (1).

L'algorithme de réaffectation des marqueurs dans la chaîne de travail (*workstring*) X(i) après une réécriture est le suivant:

```
inmark = n2 - n1; /* n1 et n2 représentent les nombres de marqueurs
   zéro dans les membres gauche et droit de la règle, respectivement.
   */
i0 = position du symbole le plus à droite réécrit par la règle;
q = 0;
Pour (i = i0; i < longueur de l'expression)
|    Si (X[i] est un marqueur zéro)
|    alors
|    |    q = q +1;
|    sinon
|    |    Si(X[i] est un marqueur esclave et que X[i] ≥ q)
|    |    alors
|    |    |    X[i] = X[i] + inmark;
|    |    Finsi
|    Finsi
|    i = i + 1;
Finpour
```

Il est facile de voir que seuls les marqueurs pointant vers des parenthèses maîtres à gauche de la partie réécrite sont incrémentés de *inmark*.

En conclusion, la réécriture dans une grammaire BP se fait en deux étapes:

1) Réécriture des symboles et des marqueurs;

2) Correction des marqueurs avec l'algorithme précédent.

Le formalisme que nous avons introduit pour la représentation de motifs ne modifie donc pas la procédure de réécriture (étape 1): les marqueurs sont copiés comme des symboles quelconques de l'alphabet Vn. Dans ce qui suit, on peut donc traiter toute grammaire BP comme une grammaire formelle de type 0, sachant que l'étape 2 doit être effectuée après chaque réécriture.

4. Grammaires BP transformationnelles (*BP transformational grammars*)

Ce terme est utilisé ici dans un sens plus restrictif qu'en linguistique. L'idée, empruntée à Kain[112], consiste à considérer toute dérivation comme une séquence de dérivations dans plusieurs grammaires; les symboles de départ A_D d'une grammaire, sauf “S”, le symbole initial, sont des symboles terminaux d'une grammaire de niveau supérieur.

Définition

> Une **grammaire transformationnelle** (sans règle d'effacement) G est un quintuplet ordonné (Ve,Vn,Vt,Vd,F) tel que:

[112] 1981, p.24

$V = Ve \cup Vn \cup Vt \cup Vd$, $Vt \neq \emptyset$, $Vd \neq \emptyset$

(Ve,Vn,Vt,Vd) est une partition d'un alphabet fini V,

et F est un ensemble fini de couples ordonnés (LCR, LDR) tel que:

$C \in (Vn \cup Vt \cup Vd)^+$, $L \in V^*$, $R \in V^*$ et $D \in (Vn \cup Vt)^+$

Vd est l'alphabet de départ, Vt l'alphabet terminal, Vn l'alphabet intermédiaire (les variables), Ve l'alphabet extérieur à G, et F l'ensemble des règles de production.

L'alphabet de départ d'une grammaire est nécessairement inclus dans la réunion des alphabets terminaux des grammaires précédentes, sauf pour la première où Vd = {S}. Si le langage est décrit par *n* grammaires, l'alphabet terminal de G_n est inclus dans l'alphabet terminal du langage. Nous convenons d'autre part que l'alphabet extérieur de G_1 est vide et que tout symbole de l'alphabet extérieur d'une grammaire G_i est un terminal d'une grammaire G_j avec $j < i$. Les symboles terminaux qui ne représentent pas des *bol*s sont appelés **symboles structurels**. Nous verrons plus loin (chapitre V §5) comment ces symboles sont gérés par les *gabarits*. Les symboles structurels reconnus par le *Bol Processor* BP1 sont:

les parenthèses et pointeurs de répétition: **(=) (:)**

le symbole de miroir: *****

les entiers (1..99) qui représentent la vitesse courante

des **symboles spéciaux** choisis parmi: {**+ : ; =** }

5. Contrôle des dérivations dans les grammaires BP (*derivation control in a BP grammar*)

Diverses stratégies de contrôle ont été implémentées dans les versions BP1 et BP2 du *Bol Processor*. Une justification détaillée de ces stratégies a fait l'objet d'une publication.[113] On présente en premier les dérivations en mode “production”. Le mode “reconnaissance” est étudié au §6.

5.1 Grammaires ‘RND’ (‘*RND’ grammars*)

La production de phrases avec un contrôle stochastique de l'ordre des règles et de la position de dérivation utilise la procédure suivante:

```
Procédure GENERER
Début
  Tant que (une règle est appliquable)
  |    Choisir une règle candidate;
  |    Choisir une position de dérivation;
  |    Si (le membre gauche de la règle est reconnu)
  |    alors
  |    |    Remplacer l'occurrence trouvée par le membre droit;
  |    Finsi
  Fintant
Fin
```

[113] Kippen & Bel 1990.

Le choix d'une règle se fait par un tirage pondéré par les poids des règles candidates (voir chapitre V §6). La position de dérivation est aléatoire. Le moteur d'inférence du BP1 peut aussi générer dans l'ordre toutes les phrases du langage.

Ce mode de contrôle est indiqué en plaçant l'instruction 'RND' (*random*) au dessus des règles de la grammaire transformationnelle.

5.2 Contrôle de position de dérivation: règles 'LEFT' et 'RIGHT' (*controlling the derivation position: 'LEFT' and 'RIGHT' rules*)

Il est souvent nécessaire, et dans tous les cas plus rapide, de prendre l'occurrence la plus à gauche (*leftmost*) ou la plus à droite (*righmost*) de l'argument gauche de la règle sélectionnée dans la chaîne de travail. Ce contrôle peut s'effectuer au niveau de chaque règle, en plaçant l'instruction correspondante, 'LEFT' ou 'RIGHT' en début de règle.

5.3 Grammaires 'LIN' (*'LIN' grammars*)

En production, le contrôle des dérivations d'une grammaire 'LIN' est identique à celui d'une grammaire'RND' dont toutes les règles seraient de type 'LEFT'.

5.4 Grammaires 'ORD' (*'ORD' grammars*)

Dans une grammaire 'ORD' les règles sont choisies dans l'ordre où elles apparaissent dans la grammaire. Chaque règle est appliquée jusqu'à saturation, puis la grammaire est analysée de nouveau pour trouver la première règle candidate. La position de dérivation est choisie comme avec l'instruction 'LEFT'.

Les grammaires 'ORD' sont utilisées surtout dans le cas où aucun choix stochastique, ni sur la règle candidate ni sur la position de dérivation, n'est nécessaire.

6. Test d'appartenance des grammaires BP (*membership test for a BP grammar*)

Un exemple d'analyse syntaxique par une grammaire BP (formalisme étendu) est proposé en annexe.

6.1 Algorithme (*algorithm*)

Le principe général de l'algorithme de test d'appartenance est le suivant:

1) Etant donnée une grammaire transformationnelle G, permuter les membres gauche et droit de toutes les règles de G. Soit G' la grammaire ainsi obtenue (**grammaire duale**).
2) Sachant que le langage est généré par l'application des grammaires transformationnelles G_1, ... G_n dans cet ordre, le test d'appartenance est le résultat de la **dérivation canonique à droite** de la chaîne analysée par G'_n, ... G'_1 dans cet ordre.
3) La chaîne appartient au langage si et seulement si le test d'appartenance se termine avec le symbole de phrase S.

La permutation des arguments dans les règles justifie l'utilisation de flèches doubles dans les grammaires: <—>

L'intérêt de cet algorithme est multiple. En premier lieu, il permet d'utiliser sensiblement les mêmes procédures qu'en synthèse pour réaliser le test d'appartenance. De plus, il est déterministe, économique en espace mémoire, et le nombre de réécritures est inférieur à la longueur de la chaîne analysée.

La seule difficulté est de définir une dérivation canonique pour des grammaires contextuelles. Par *dérivation à droite* il faut entendre une dérivation qui réécrit le symbole non terminal situé le plus à droite dans la chaîne. Cette définition est ambiguë: le contexte fait-il partie des symboles réécrits? Si l'on répond 'oui' à cette question, il n'est pas possible de donner une solution générale au problème de l'ambiguïté de la grammaire contextuelle. Par contre, une définition plus générale des dérivations canoniques[114] permet de définir des contraintes sur les règles de production qui garantissent que la grammaire ne sera pas ambiguë.

6.2 Dérivation contextuelle à droite (*context-sensitive rightmost derivation*)

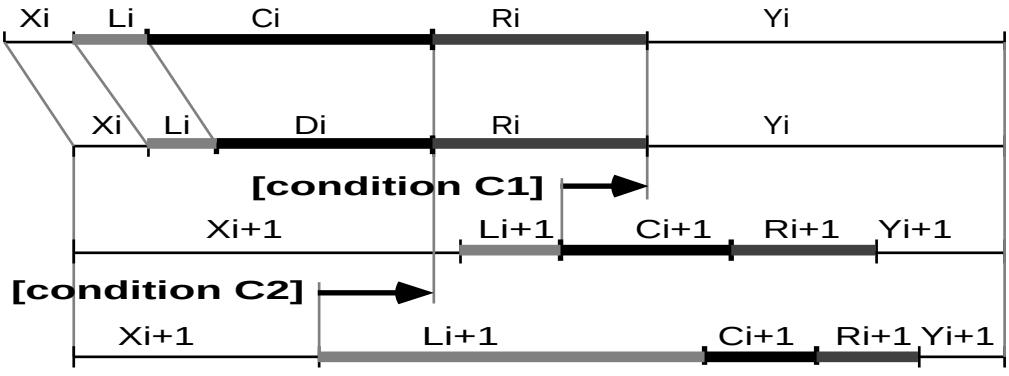
Soit G une grammaire contextuelle. La dérivation de G:

$W_0 => W_1 => ... => W_n$

est **contextuelle à droite** (*context-sensitive rightmost*) si et seulement si

$\forall i \in [0, n-1]$, $W_i = X_iL_iC_iR_iY_i$
et $W_{i+1} = X_iL_iD_iR_iY_i$ en appliquant la règle f_i: $L_iC_iR_i \rightarrow L_iD_iR_i$,
et l'une au moins des conditions suivantes est réalisée:
(C1) $|C_{i+1}R_{i+1}Y_{i+1}| > |Y_i|$
(C2) $|L_{i+1}C_{i+1}R_{i+1}Y_{i+1}| > |R_iY_i|$

Cette définition est adaptée de celle de Hart[115]. Hart n'étudie que le cas des grammaires **strictement contextuelles** pour lesquelles $|C_i| = |C_{i+1}| = 1$, et la condition C1 peut alors s'écrire $|R_{i+1}\ Y_{i+1}| \geq |Y_i|$. La figure ci-dessous illustre les conditions **C1** et **C2** qui seront justifiées plus loin:



Supposons que les conditions **C1** et **C2** ne soient pas respectées. Du fait que **C2** n'est pas vérifiée, la règle f_{i+1} aurait pu être appliquée avant f_i puisque $L_{i+1}C_{i+1}R_{i+1}$ serait une sous-chaîne de R_iY_i. D'autre part, puisque **C1** n'est pas vérifiée, l'application de la

114 Hart 1980.
115 *op.cit.* p.82

règle f_{i+1} ne modifierait que la chaîne Y_i sans toucher au contexte R_i. Dans ce cas, l'ordre de f_i et de f_{i+1} pourrait être inversé. Cette permutation serait légitime puisque les symboles réécrits par f_{i+1} se trouvent à droite de ceux réécrits par f_i.

6.3 Application de la dérivation canonique à l'algorithme de reconnaissance (*applying the canonic derivation to the membership test*)

Supposons que pour une chaîne Wi deux règles soient appliquables:

f_i $L_iC_iR_i$ -> $L_iD_iR_i$

f'_i $L'_iC'_iR'_i$ -> $L'_iD'_iR'_i$

sachant que $X_iL_iC_iR_iY_i = X'_iL'_iC'_iR'_iY'_i = W_i$

Ordonnons ces règles selon les critères suivants: f_i sera choisie en priorité sur f'_i si l'une des conditions suivantes, prises dans l'ordre, est vérifiée:

(D1) $|X_iL_iC_i| > |X'_iL'_iC'_i|$
(D2) $|X_iL_iC_iR_i| > |X'_iL'_iC'_iR'_i|$
(D3) $|L_iC_iR_i| > |L'_iC'_iR'_i|$
(D4) $i > i'$

Commentaire:

D1 permet à un nombre maximum de règles d'être candidates à l'étape suivante de réécriture en remplissant la condition **C2**.

D2 permet à un nombre maximum de règles d'être candidates à l'étape suivante de réécriture en remplissant la condition **C1**.

Lorsque **D3** est appliqué, les relations **D1** et **D2** se ramènent à des égalités, et par conséquent $R_i = R'_i$ et $L'_iC'_iR'_i$ est une sous-chaîne de $L_iC_iR_i$. L'ambiguïté est alors levée en appliquant le principe suivant:

> Les motifs les plus longs (contextes inclus) sont reconnus en priorité.

Lorsque **D3** ne permet pas de lever l'ambiguïté, les membres droits des règles de production sont identiques. Il y a réellement ambiguïté, mais cette situation peut être évitée par une écriture attentive des grammaires. Dans ce cas, le moteur d'inférences utilise un critère de sélection arbitraire: l'ordre inverse d'apparition des règles dans la grammaire.

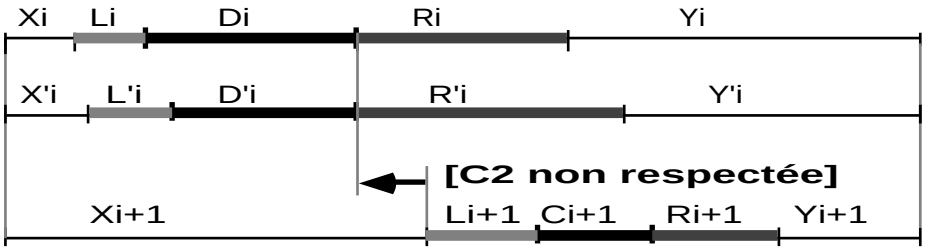
6.4 Simplification des critères dans le *Bol Processor* (*simplifying these criteria in the BP*)

Supposons qu'après l'application du critère **D1** deux règles f_i et f'_i soient candidates, et donc que $|X_iL_iC_i| = |X'_iL'_iC'_i|$. Après réécriture par f_i et f'_i respectivement, les chaînes seraient:

$W_{i+1} = X_iL_iD_iR_iY_i$ et $W'_{i+1} = X'_iL'_iD'_iR'_iY'_i$

avec $R_iY_i = R'_iY'_i$

Est-il nécessaire d'utiliser le critère **D2** pour choisir entre f_i et f'_i ? Ce critère permet de réaliser au mieux la condition **C1** sur les règles candidates f_{i+1}. Prenons l'exemple d'une règle f_{i+1} qui vérifie **C1** et pas **C2**:



Il est clair que la situation ci-dessus ne peut se produire car dans ce cas la règle f_{i+1} devait être appliquée avant f_i ou f_{i+1} selon le critère **D1**. Il s'ensuit que l'examen du critère **D2** n'est pas utile pour lever les ambiguïtés nées de l'application de **D1**.

Les situations d'ambiguïté découlant de l'application du critère **D1** seront donc immédiatement traitées par **D3**. Dans ce cas, $L'_iC'_iR'_i$ n'est pas nécessairement une sous-chaîne de $L_iC_iR_i$. Dans la version BP1, le critère **D3** n'est pas pris en considération par le moteur d'inférences, et c'est en définitive **D4**, c'est à dire l'ordre des règles, qui lève les ambiguïtés. Il s'ensuit que l'utilisateur doit respecter la règle suivante:

Règle des chunks

> Le membre droit d'une règle f_i ne peut être une sous-chaîne de celui d'une règle f_j telle que j<i.

6.5 Dépendance du contexte et dérivation canonique (*context dependency and canonic derivation*)

Les dérivations dans les grammaires contextuelles peuvent être représentées par des arbres syntaxiques.[116] Toutefois, ces arbres ne permettent pas de déduire la forme canonique d'une dérivation, c'est à dire l'ordre d'application des règles. Une solution satisfaisante a été proposée par Hart[117]. Elle consiste à insérer dans l'arbre syntaxique des arcs orientés exprimant deux types de contraintes:

> $<_{\mathbf{c}}$ la dépendance de contexte: la règle pointée par l'extrêmité de cet arc ne peut être appliquée que si le symbole placé à son origine est présent;
> $<_{\mathbf{f}}$ la 'libération' de contexte: la règle pointée par l'extrêmité de cet arc est 'gelée' jusqu'à ce que la règle pointée par son origine ait été appliquée.

On peut en outre ajouter à l'arbre syntaxique des arcs orientés $<_{\mathbf{d}}$ définis ainsi:

> Un arc relie f_i et f_j s'il n'existe pas de chemin entre ces noeuds;
> $f_i <_{\mathbf{d}} f_j$ si la dérivation de f_i se fait à droite de celle de f_j.

Ces arcs représentés simultanément forment un **graphe** 3-coloré **doublement ordonné** (*doubly ordered graph*), et ce graphe permet de déduire un ordre des règles qui correspond à la dérivation canonique.[118] Considérons par exemple la grammaire **LIN** suivante:

$\mathbf{f_1}$	S	<—> EBA
$\mathbf{f_2}$	B	<—> CD

[116] Révész 1985, p.57
[117] 1980
[118] Révész 1985, p.182

f_3 DA <—> DEF
f_4 EC <—> AAC

dans laquelle nous avons souligné les contextes, et la dérivation canonique à droite du mot:

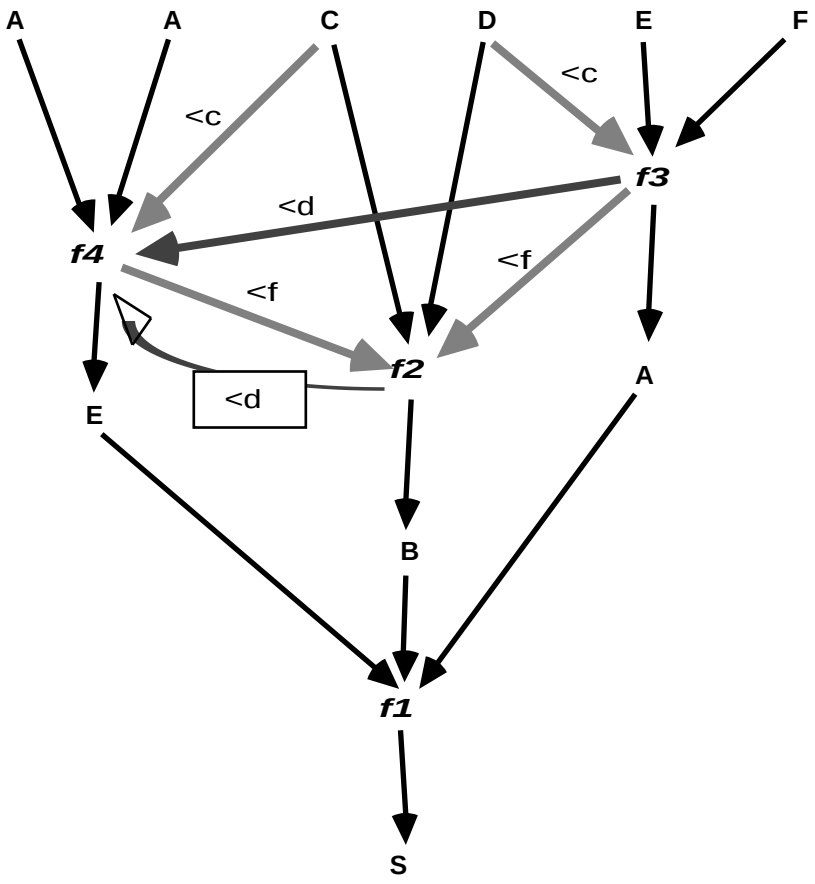
AACDEF => AACDA => ECDA => EBA => S
f_3 f_4 f_2 f_1

Si l'on appliquait le critère **D1** défini au paragraphe précédent, la dérivation serait:

AACDEF => AACDA => AABA
f_3 f_2

Par conséquent, le critère **D1** est insuffisant ici pour définir la dérivation canonique à droite. Nous avons représenté ci-dessous le graphe syntaxique contextuel (*context-sensitive syntactical graph*)[119] correspondant à l'analyse correcte. L'arc marqué $<_{\mathbf{d}}$ encadré ne fait pas partie de ce graphe, puisqu'il existe déjà un arc $<_{\mathbf{f}}$ entre f_4 et f_2. Il est clair que le gel de la règle f_2 contredit l'ordre imposé par la contrainte **D1**. Il est clair, d'autre part, que les arcs $<_{\mathbf{f}}$ qui proviennent d'un noeud f_i représentant une règle avec contexte à gauche sont tous orientés de droite à gauche, c'est à dire dans le même sens que les arcs $<_{\mathbf{d}}$. Enfin, il n'existe pas de paire d'arcs ($<_{\mathbf{c}}$,$<_{\mathbf{f}}$) ayant pour origine un contexte qui n'appartient pas à l'alphabet d'une grammaire transformationnelle.

[119] *ibid.*



Nous en tirons la règle suivante:

Règle des contextes

> Dans une grammaire LIN, un contexte à droite ne peut être composé que de symboles de l'alphabet extérieur de cette grammaire.

6.6 Test d'appartenance des grammaires ‘RND’ et ‘ORD’ (*membership test for ‘RND’ and ‘ORD’ grammars*)

Les grammaires ‘LIN’ permettent, à condition de respecter la règle des chunks et celle des contextes, de réaliser la dérivation canonique, en analyse, de n'importe quelle phrase du langage. Dans les grammaires ‘RND’ et ‘ORD’, c'est l'ordre des règles qui a priorité sur la position de dérivation (critère **D4**). De plus, lorsqu'une règle est sélectionnée, elle est appliquée jusqu'à saturation.

L'algorithme correspondant est le suivant:

```
Procédure REECRIRE(i)
Début
  Chercher le membre droit de f_i dans la phrase à partir de la droite;
  Remplacer l'occurence trouvée par le membre gauche de f_i ;
Fin
```

```
Procédure SATURER(i)
Début
  Tant que (la règle f_i est applicable)
  |    REECRIRE(i);
  |    test = 1;
  Fintant
Fin

Procédure ANALYSER
Début
  Répéter
  |    i = numéro de la dernière règle;
  |    test = 0;
  |    Répéter
  |    |    SATURER(i);
  |    |    i = i - 1;
  |    jusqu'à ce que (i = 0) ou (test = 1)
  jusqu'à ce que (test = 0)
Fin
```

Cet algorithme est acceptable lorsque tous les contextes des règles utilisent exclusivement des symboles de l'alphabet extérieur, et lorsque les motifs des membres droits des règles ne se chevauchent pas partiellement. Le problème du chevauchement de motifs est traité ailleurs.[120]

[120] Bel 1987.

V. Formalisme BP1 (*BP1 formalism*)

Une extension du modèle des grammaires BP, implémentée dans le ***Bol Processor BP1***, est présentée dans ce chapitre. Les quatre premiers paragraphes décrivent un élargissement de la syntaxe des règles de réécriture, les deux suivants s'intéressent au contrôle des inférences. Dans les chapitres suivants seront présentées d'autres extensions du formalisme qui concernent essentiellement la représentation du temps et des objets sonores, et qui ont été implémentées dans la version BP2 du *Bol Processor*, orientée vers l'assistance à la composition. Les extensions syntaxiques du BP2 (substitutions, grammaires programmées, contextes éloignés, etc.) sortent du cadre de cette étude.

1. Règles de type 0 (*type-0 rules*)

Nous appelons règle de type 0, dans une grammaire BP1, toute règle de réécriture:

$$p \longrightarrow q$$

telle que

$$p , q \in (Vt \cup Vn \cup Vd \cup Ve)^{+}$$

Si l'on compare cette définition avec celle des règles bien formées dans un RPL (Chapitre IV §2.3), en tenant compte des conventions d'écriture (§3.1), on constate que:

1) *p* peut contenir des symboles autres que des variables;
2) il n'y a pas de restriction sur les longueurs des arguments;
3) *p* peut contenir plusieurs fois la même variable (dans un motif).

Ces trois extensions sont illustrées par les exemples suivants:

1) a X Y —> a b Z T
2) X Y —> Z
3) (= X) (: X) —> Y Z

où $a,b \in Vt$ et $X,Y,Z,T \in Vn$.

C'est à l'utilisateur qu'il incombre de déterminer si une grammaire utilisant ce formalisme étendu génère un langage non vide, et si ce langage est exactement le même que celui reconnu par l'algorithme de test d'appartenance (voir chapitre IV §6).

2. Négation de contexte (*negative context*)

Soit une grammaire transformationnelle (Ve,Vn,Vt,Vd,F) définie comme au chapitre IV §4. L'alphabet est: $V = Ve \cup Vn \cup Vt \cup Vd$.

2.1 Règle à contexte négatif (*negative-context rules*)

Considérons un alphabet V' arbitraire tel que card(V') = card(V), $V \cap V' = \varnothing$, et appelons *Neg* une bijection arbitraire entre V et V'. Soient V'e, V'n, V't et V'd les images de Ve, Vn, Vt et Vd par *Neg*. Nous appellons **règle à contexte négatif** un couple ordonné (LCR, LDR) tel que:

$C \in (Vn \cup Vt \cup Vd \cup V'n \cup V't \cup V'd)^+$,

$L,R \in (V \cup V')^*$,

$D \in (Vn \cup Vt \cup V'n \cup V't)^+$,

$Proj_{V'}(L\ C\ R) = Proj_{V'}(L\ D\ R)$

où $Proj_{V'}()$ désigne la projection sur l'alphabet V' d'une chaîne de symboles. Informellement, cette projection est obtenue en effaçant, dans la chaîne, tous les symboles n'appartenant pas à V'.

2.2 Unification de contexte négatif (*negative context unification*)

Pour déterminer si une règle à contexte négatif est candidate, on compare son argument gauche X avec une sous-chaîne Y de la chaîne de travail (*workstring*). Nous appelons **unification** cette opération. La mise en correspondance (*matching*) de X avec Y, en l'absence de contexte négatif, est un cas particulier de l'unification.

X est une chaîne appartenant à $(V \cup V')^+$, tandis que Y appartient à V^+ .

Appelons X' l'argument droit de la règle, X1 le plus long préfixe commun à X et X' tel que $X1 \in V'^*$, X3 le plus long suffixe commun à X et X' tel que $X3 \in V'^*$, et X2 la sous-chaîne de X telle que X = X1.X2.X3.

L'unification ne peut réussir que si $|X| \geq |Y|$. Formons une nouvelle chaîne

$$Y' = \varphi^i \,.\, Y \,.\, \varphi^j$$

telle que |Y'| = |X|, où φ représente un symbole spécial, avec les restrictions suivantes:

1) $i > 0$ seulement si Y est un préfixe de la chaîne de travail;

2) $j > 0$ seulement si Y est un suffixe de la chaîne de travail;

3) $i \leq |X1|$ et $j \leq |X3|$.

On définit maintenant un vecteur bidimensionnel *dneg* qui set à mesurer la ressemblance entre les chaînes de longueurs identiques X et Y.

Si a1, a2, T1 et T2 désignent un symbole et une sous-chaîne de X et Y' respectivement,

1) dneg(a1.T1, a2.T2) = dneg(a1,a2) + dneg(T1,T2);

2) dneg(λ,λ) = (0,0); (λ est la chaîne vide.)

3) dneg(a1,φ) = (0,0);

4) dneg(a1,a2) = (1,0) si et seulement si a1 ∈ V, a1 ≠ a2, et a2 ≠ φ;

5) dneg(a1,a2) = (0,1) si et seulement si a1 ∈ V' , a1 ≠ Neg(a2), et a2 ≠ φ;

On convient de dire que X **s'unifie avec** Y si et seulement si :

$$\exists\ m > 0 \text{ tel que } dneg(X,Y') = (0,m)\ .$$

Commentaires

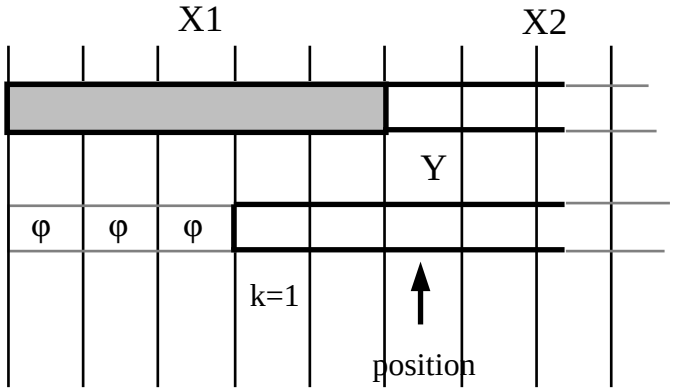
En calculant dneg(X,Y') on calcule en fait:

1) le nombre de symboles différents au même rang dans X et Y', et qui appartiennent à V (1ère coordonnée);
2) le nombre de symboles différents au même rang dans X et Y', et qui appartiennent à V' (2ème coordonnée);

On néglige en outre les symboles de X qui correspondent à des symboles φ de Y'. D'après les conditions énoncées plus haut sur i et j, ces symboles de X sont en effet des symboles de V', autrement dit, informellement, des contextes négatifs qui sont “instanciés” avant le début ou après la fin de la chaîne de travail.

La condition d'unification exprime donc l'idée que les symboles de V doivent être identiques, alors qu'au moins un symbole de V' doit être différent de l'image par *Neg* du symbole correspondant dans Y'.

Il peut exister plusieurs valeurs de *i* pour lesquelles l'unification est vérifiée, qui correspondent à plusieurs positions de dérivation. Si k est le rang du premier symbole de Y dans la chaîne de travail, nous appelons **position de dérivation** le nombre (k - i + |X1|). (Rappelons que i = 0 lorsque k > 1). Les deux situations, pour k = 1 et pour k > 1, sont les suivantes:



Il est facile de montrer que, dans le cas d'une règle sans contexte négatif, *dneg* représente la distance de Hamming entre X et Y.

2.3 Réécriture avec contexte négatif (*rewriting an expression with negative context*)

La condition d'unification étant vérifiée entre X et Y, on étudie maintenant la réécriture de Y.

Puisque $Proj_{V'}(X) = Proj_{V'}(X')$, où X et X' représentent les arguments gauche et droit de la règle, respectivement, on peut définir une bijection

$$t : \{X(g) \in V'\} \longrightarrow \{X'(h) \in V'\}$$

où X(g) représente le symbole de rang *g* dans X, et X'(h) celui de rang *h* dans X', telle que

X'(h) = t(X(g)) <=> $Proj_{V'}(X(g))$ et $Proj_{V'}(X'(h))$ sont deux symboles de même rang.

Appelons λ l'élément neutre pour la concaténation. Les significations de i et j ont été données précédemment (§2.2). Posons n = |X'|.

La réécriture de Y revient à remplacer Y par Y'' défini comme suit:

1) Y''(h) = λ si et seulement si $h \leq i$ ou $h \geq n - j$;

2) Y''(h) = Y(g-i), où *h* est tel que X'(h) = t(X(g))

si et seulement si $h > i$, $h < n - j$, et $X(g) \in V'$;

3) Y''(h) = X'(h) dans tous les autres cas.

2.4 Notation et exemple (*notation and example*)

La bijection *Neg* est notée '#' dans BP1 et BP2. Prenons le cas d'un alphabet Vt = {a,b,c,…} et Vn = {A,B,C}. Soit une chaîne de travail

b b B B E E e a B c E

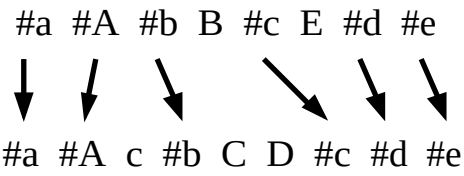
et la règle

#a #A #b B #c E #d #e —> #a #A c #b C D #c #d #e

telle que:

$$Proj_{V'}(X) = Proj_{V'}(X') = \#a\ \#A\ \#b\ \#c\ \#d\ \#e\ .$$

Le graphe de l'application *t* est évidemment:



On a d'autre part

X1 = #a #A , X2 = #b B #c E, et X3 = #d #e .

L'argument gauche X = #a #A #b B #c E #d #e s'unifie avec trois sous-chaînes distinctes de la chaîne de travail. Etudions d'abord l'unification la plus à gauche (*leftmost*), qui se fait pour k = 1, soit

Y = b b B B E E e

avec i = 1 et j = 0, ce qui donne:

X	=	#a	#A	#b	B	#c	E	#d	#e
Y'	=	φ	b	b	B	B	E	E	e

pour lequel on a dneg(X,Y') = (0,3). La position de dérivation est

$$k - i + |X1| = 1 - 1 + 2 = 2.$$

La réécriture se fait en calculant Y" à partir de X'. Le tableau ci-dessous met en correspondance ces deux chaînes, en indiquant, pour chaque valeur du rang *h*, le numéro du cas rencontré:

h		1	2	3	4	5	6	7	8	9
X'	=	#a	#A	c	#b	C	D	#c	#d	#e
cas n°		1	2	3	2	3	3	2	2	2
Y"	=	λ	b	c	b	C	D	B	E	e

La dérivation de la chaîne de travail est donc

b b B B E E e a B c E => **b c b C D B E e** a B c E

où Y et Y" ont été soulignés en gras.

Une deuxième position de dérivation est possible. Prenons, toujours avec k = 1,

Y = b b B B E E e a

avec i = j = 0, ce qui donne

X = #a #A #b B #c E #d #e

Y' = b b B B E E e a

pour lequel on a dneg(X,Y') = (0,6). La position de dérivation est dans ce cas

$$k - i + |X1| = 1 - 0 + 2 = 3.$$

On peut trouver Y" de la même manière:

h		1	2	3	4	5	6	7	8	9
X'	=	#a	#A	c	#b	C	D	#c	#d	#e
cas n°		2	2	3	2	3	3	2	2	2
Y"	=	b	b	c	B	C	D	E	e	a

ce qui aboutit à la dérivation:

b b B B E E e a B c E => **b b c B C D E e a** B c E .

Il existe enfin une troisième position de dérivation possible. Prenons, avec k = 6,

Y = E e a B c E

avec i = 0 et j = 2, ce qui donne

X = #a #A #b B #c E #d #e

Y' = E e a B c E φ φ

pour lequel on a dneg(X,Y') = (0,3). La position de dérivation est dans ce cas

k - i + |X1| = 6 - 0 + 2 = 8.

On calcule Y'':

h		1	2	3	4	5	6	7	8	9
X'	=	#a	#A	c	#b	C	D	#c	#d	#e
cas n°		2	2	3	2	3	3	2	1	1
Y''	=	E	e	c	a	C	D	c	λ	λ

ce qui aboutit à la dérivation:

b b B B E **E e a B c E** => b b B B E **E e c a C D** .

3. Valeurs nulles (*wildcards*)

Un symbole réservé *w*, noté "?" dans les règles BP, a été introduit dans les arguments de règles. On modifie V't de sorte que w = Vt ∩ V't, et la bijection *t* de sorte que Neg(w) = w.

L'algorithme d'unification (du membre gauche avec une sous-chaîne de la chaîne de travail) a été modifié ainsi (voir §2.2):

> dneg(w,a2) = (0,0) ;
>
> et a1 ≠ w dans les cas (4) et (5).

En ce qui concerne la réécriture de Y (voir §2.3), on est maintenant dans le cas (2) chaque fois qu'il faut réécrire *w*:

> 2) Y''(h) = Y(g-i), où *h* est tel que X'(h) = t(X(g))
>
> si et seulement si h > i, h < n - j, et X(g) ∈ V'.

On peut appeler *w* une **métavariable** et Y(g-i) une **instance** de cette métavariable.

4. Tempo (*tempo*)

Dans la notation rythmique nord-indienne, on utilise en général des espaces ou des tabulations pour marquer les temps de la mesure. Dans l'exemple de *qa'ida* présenté au chapitre III §3, chaque temps comprenait six objets sonores (*bol*s). On parle dans ce cas de "vitesse 6", ou "densité 6". Le problème est maintenant de représenter des variations de tempo au sein d'une même phrase, par exemple

dhatidhage	nadhatrkt	dhatidhage	dheenatrkt
dhadhatrkt	dhatidha-	dhatidhage	teena-ta
teena-ta	titakena	tatitake	teenakena
dhatidhagenadhatrkt	dhatidhagedheenagena	gena-dhatidhagena	dhatidhagedheenagena

où les trois premières lignes (12 temps) sont à vitesse 4 et la dernière (4 temps) à vitesse 8, sachant que les espaces et sauts de lignes ne sont pas codés. Une manière simple d'y parvenir est d'indiquer explicitement la vitesse en début de phrase et chaque fois qu'elle est modifiée. On utilise à cet effet un entier précédé d'un *slash* "/":[121]

/4 dha ti dha ge na dha tr kt dha ti dha ge dhee na tr kt dha dha tr kt dha ti dha - dha ti dha ge tee na - ta tee na - ta ti ta ke na ta ti ta ke tee na ke na /8 dha ti dha ge na dha tr kt dha ti dha ge dhee na ge na ge na - dha ti dha ge na dha ti dha ge dhee na ge na

Ces symboles sont manipulés par les règles de réécriture comme des symboles terminaux.

Une procédure a été implémentée dans le *Bol Processor* pour restituer à l'écran les tabulations en fonction des indications de tempo.

5. Gabarits (*templates*)

Les phrases générées par une grammaire BP contiennent des symboles terminaux réservés "(", ")", "=", ":", "*" qui servent à marquer les structures de motifs. La phrase terminale est obtenue en effaçant ces symboles. Toutefois, une phrase donnée ne peut être analysée que si sa structure est explicitement représentée.

De plus, la représentation graphique des structures est distincte de la représentation interne, qui fait intervenir des pointeurs (marqueurs de parenthèses). Dans l'éditeur de la version BP1, il est possible de construire cette représentation interne lors de la saisie d'une phrase au clavier. Le problème de l'analyste est toutefois que la structure de motifs est souvent difficile à déterminer, voire ambiguë.

Il a donc été nécessaire de mettre au point une procédure qui permette d'assigner à une phrase "brute" une (ou plusieurs) structure(s) avant de lui appliquer le test d'appartenance. On est parti du constat que le nombre de structures possibles, pour un *qa'ida* donné, est toujours peu élevé (rarement plus d'une dizaine). Etant donnée une grammaire, il est donc possible d'énumérer et de mettre en mémoire toutes les structures possibles sous forme de **gabarits**. Informellement, un gabarit est une représentation squelettique de la phrase dans laquelle tous les symboles terminaux qui représentent des *bol*s sont remplacés par un symbole unique ".", par exemple:

(=.....)(=.....)(=.....)*(.....)(:.....)(=.....)

(=.....)(=.....)*(.....)(:.....)

(=.....)(=.....)*(.....)(=.....)

[121] Le *slash* a été introduit seulement dans la version BP2, afin de représenter les silences par des fractions. (Voir chapitre VIII §1)

Un gabarit peut aussi contenir les symboles qui indiquent des changements de tempo, par exemple:

(=+.....+.....+ * (=/6+...../4.....+) +.....;)

(Voir la grammaire en annexe 1)

5.1 Génération des gabarits (*template generation*)

Il n'est pas nécessaire en général d'énumérer toutes les phrases d'un langage pour déterminer tous les gabarits. Par exemple, dans la grammaire citée en annexe 1, seules les sous-grammaires transformationnelles 1 - 2 définissent des structures (en générant des symboles "(=", etc., et des indications de tempo). Les grammaires suivantes n'utilisent ces symboles que comme contextes. La génération des gabarits se limite donc à l'énumération des phrases générées par les sous-grammaires 1 - 2.

Pour chaque phrase ainsi générée, on continue à appliquer les règles des grammaires 3 - 4 - 5 - 6, jusqu'à obtenir une phrase terminale, mais sans énumérer toutes les phrases terminales. On sait en effet que toutes ces phrases ont la même structure, en particulier que **leurs longueurs (en nombre de *bol*s) sont identiques**.

Une remarque méthodologique s'impose ici: étant donné l'ensemble L des variations acceptables d'un *qa'ida*, on ne peut représenter L à l'aide d'une grammaire BP qu'à condition de connaître à l'avance les longueurs des éléments de L. Or la métrique musicale impose que ces longueurs soient identiques ou choisies dans une liste simple (ex: {16,32}). Si l'on envisage maintenant la grammaire d'un *qa'ida* comme un ensemble de "décisions" dans un processus compositionnel, il est nécessaire que la décision sur la longueur de la phrase soit prise "à un haut niveau", autrement dit dans une des grammaires qui définissent la structure des motifs.

On remarque, dans la grammaire de l'annexe 1, que les étiquettes des variables contiennent en général un nombre entier qui représente la longueur de la partie de la phrase générée par cette variable. Ainsi, dans la règle

GRAM#2 [5] <100> S64 <—> L24 S40

il est clair que pour générer une phrase de 64 unités on utilise une variable L24 "gouvernant" les 24 premières unités suivie de S40 gouvernant les 40 suivantes.

5.2 Utilisation des gabarits — ambiguïtés structurelles (*using templates — structural ambiguity*)

Il est facile de mettre en correspondance une phrase "brute", sans indication de structure (sauf, le cas échéant, les variations de tempo), et un gabarit. Informellement, on remplace chaque "." du gabarit par le *bol* correspondant, de gauche à droite, tout en vérifiant la cohérence sur les répétitions et l'homomorphisme miroir.

Une fois cette mise en correspondance réalisée avec succès, on lance l'algorithme de test d'appartenance (voir annexe 1). Même en cas de succès le système peut analyser tous les gabarits, ceci afin de mettre en évidence les ambiguïtés structurelles. Les cas d'ambiguïté proviennent en général de répétitions ou de miroirs que l'on peut considérer, soit comme "accidentels", soit comme "délibérés". Certains *qa'ida*s utilisent cette

ambiguïté de manière systématique.[122] Dans une grammaire citée ailleurs,[123] par exemple, les structures sont définies comme suit:

```
GRAM#1 [1]          RND
GRAM#1 [2]          S <—> S48
GRAM#1 [3]          S <—> S96
-------------------------------------------------------------------------------------------------------------
GRAM#2 [1]          RND
GRAM#2 [2]          S96 <—> (= F48 ) (= V24 ) F'24 *(: F48 ) (: V24 ) F24
GRAM#2 [3]          S48 <—> (= V24 ) F'24 *(: V24 ) F24
GRAM#2 [4]          V24 <—> (= V12 ) (: V12 )
GRAM#2 [5]          V24 <—> (= V12 ) *(: V12 )
GRAM#2 [6]          V24 <—> Q24
GRAM#2 [7]          V24 <—> V12 V12
GRAM#2 [8]          V24 <—> B24
GRAM#2 [9]          V12 <—> (= B6 ) *(: B6 )
GRAM#2 [10]         V12 <—> B12
-------------------------------------------------------------------------------------------------------------
... etc.  (sous-grammaires suivantes)
```

On peut remarquer que les règles [4] et [5] de la sous-grammaire 2 dérivent V24 explicitement avec une répétition et un miroir, alors que les règles [6], [7] et [8] définissent des dérivations plus générales, où ces répétitions ou miroirs ne peuvent être qu'accidentels. On a ordonné de même les règles dérivant V12, de la plus spécifique à la plus générale. Cet ordre partiel est reflété de la même manière sur les gabarits générés automatiquement par la grammaire. Par conséquent, lors de l'analyse, en cas d'ambiguïté, la première structure suggérée à l'analyste est la plus spécifique, autrement dit, celle qui fait apparaître le plus d'informations sur une structure produite intentionnellement ou “accidentellement”.

6. Modèle stochastique (*stochastic model*)

Cette extension concerne essentiellement le contrôle des inférences lors de la production de phrases. Dans les grammaires ‘RND’ ou ‘LIN’, le choix des règles se fait par tirage au hasard d'une règle candidate. Dans un premier temps, il est intéressant de pouvoir temporairement inhiber certaines règles afin de n'examiner qu'une partie jugée intéressante du langage. Ces règles inhibées, même candidates, ne participent pas au tirage. Dans un deuxième temps, certaines productions sont estimées exceptionnelles, et l'on désire pondérer les tirages afin que l'ensemble des phrases générées pendant une session expérimentale reflète autant que possible — à l'ordre près — un segment “représentatif” du langage.[124]

Des modèles probabilistes de grammaires ou d'automates ont été proposés, comme par exemple celui de Booth & Thompson[125] qui s'applique à certaines classes de grammaires pour lesquelles on peut définir une fonction probabiliste de phrase. Il s'agit

---

[122] Un exemple est donné dans Bel 1987b.

[123] Kippen & Bel 1990, annexe.

[124] L'introduction d'un modèle stochastique a permis d'améliorer considérablement l'acquisition de connaissances en augmentant la crédibilité du système auprès des informateurs (Kippen & Bel 1990).

[125] 1973

nécessairement de grammaires de type 2 (*context-free*) non ambiguës; à toute phrase est affectée une probabilité calculée comme le produit de probabilités des règles utilisées dans sa dérivation.

Maryanski & Booth[126] ont proposé un algorithme pour inférer les probabilités de règles (d'une grammaire probabiliste) à partir d'un échantillon *complet* de phrases. Par "complet" il faut entendre un échantillon pour lequel toutes les règles ont été utilisées. Une variante de cet algorithme, qui n'impose pas que l'échantillon soit complet, peut être utilisée pour générer des "**poids**" de règles, dont nous montrerons qu'on peut déduire des probabilités:

> 1) Supposons que l'on dispose d'une grammaire probabiliste G, dont les probabilités de règles sont inconnues, et d'un échantillon de phrases reconnues par la grammaire. On affecte à chaque règle un entier, appelé le poids, initialement zéro. On analyse ensuite chaque phrase et on incrémente d'une unité tous les poids des règles utilisées pour l'analyse. (A noter que l'analyse produit un arbre unique puisque la grammaire est de type 2 non ambiguë.)

> 2) Les poids ainsi inférés sont utilisés en production comme suit:

> Pour une chaîne de travail (*workstring*) donnée, on établit la liste des règles candidates (dérivation à gauche, *leftmost*). Soit N la somme des poids des règles candidates. La probabilité d'une règle candidate de poids *n* est n/N.

Il est facile de montrer que l'algorithme d'inférence (partie (1)) est identique à celui de Maryanski et Booth, sauf que, si une règle n'est jamais utilisé (échantillon incomplet), son poids reste nul, et par conséquent (partie (2)) sa probabilité est toujours nulle.

Un exemple de calcul de probabilité de règle candidate est donné maintenant. Soit la grammaire:

```
[1]   <100>   V3    —>   dhagena
[2]   <100>   V3    —>   dhatrkt
[3]   <50>    V3    —>   dha--
[4]   <5>     V3    —>   dhati-
```

et supposons que la chaîne de travail contienne V3. La somme des règles candidates étant 255, la probabilité de choisir la règle [4] est 5/255 = 0,0196.

En appliquant (1) et (2) à des grammaires non probabilistes, on obtient néanmoins des poids qui reflètent, au moins qualitativement, l'importance des règles[127]. Lorsque l'inférence des poids se fait à partir des gabarits, le système ne retient que le premier gabarit aboutissant à un test d'appartenance positif. Il est donc important d'ordonner correctement les règles structurelles (voir §5.2).

Un exemple de grammaire avec poids inférés à partir d'exemples a été publié.[128]

Des exemples d'applications de modèles stochastiques de type grammaire/automate ont été proposés par Kevin Jones.[129]

---

[126] 1977, p.525

[127] En fait, une grammaire strictement probabiliste ne présente aucun intérêt pour l'acquisition de connaissances: elle ne génère et ne reconnaît en effet que des phrases déjà connues (i.e. faisant partie de l'échantillon utilisé pour inférer les probabilités).

[128] Kippen & Bel 1990, annexe 3.

[129] 1989

VI. Apprentissage inductif (*inductive learning*)[130]

Les travaux avec le *Bol Processor* BP1 ont mis à jour deux problèmes d'acquisition de connaissances:

(1) L'amélioration des grammaires dépend fortement de la grammaire utilisée comme première hypothèse.

(2) Lorsqu'une grammaire produit une pièce ou une analyse incorrecte, il n'existe pas de procédure rigoureuse pour corriger ses règles.

Autrement dit, il est difficile de maîtriser manuellement les opérations de **généralisation** et de **spécialisation** qui sont essentielles en acquisition de connaissances.

Une méthode permettant d'induire un schéma d'improvisation à partir d'exemples fournis par un expert musicien est présentée dans ce chapitre. Le formalisme grammatical est limité à celui des grammaires de type 2 (*context-free*).

L'algorithme proposé résout deux types de problèmes:

— la **segmentation** de chaînes de symboles;[131]

— la mise en évidence de **structures syntaxiques** décrivant des familles de chaînes.

Le problème de la segmentation revient à identifier, sur un corpus de données codées conventionnellement, les plus petites “unités signifiantes” qui sont assemblées pour constituer des “formes acceptables”. La terminologie utilisée ici est essentiellement empruntée à la linguistique, ce qui suppose que la notion d'**acceptabilité syntaxique** ait un sens dans le domaine considéré.[132]

La méthode de segmentation de Ruwet revient à énumérer, dans une chaîne[133] de symboles musicaux, toutes les sous-chaînes identiques dans l'ordre de longueurs décroissantes. Vecchione a montré que si l'on appliquait strictement l'algorithme proposé on n'obtenait pas toujours le résultat escompté.[134] Gilbert Rouget, dont s'est inspiré Ruwet, avait proposé de chercher les répétitions de segments “similaires”. La notion de

---

[130] Ce chapitre a fait l'objet d'une communication aux *Journées Françaises de l'Apprentissage*. Voir Bel 1990a.

[131] Problème posé en musique par: Ruwet 1972; Nattiez 1975; Camilleri & al. 1990; Smaill & Wiggins 1990.

[132] Jakobson a défini la musique comme une “syntaxe d'équivalences”, remplaçant en quelque sorte une sémantique de la vérité des propositions par une “sémantique des ressemblances”. Voir à ce sujet Ruwet 1972 p.10.

[133] La portée d'une méthode d'analyse de pièces tonales ne s'appliquant qu'aux formes monodiques est limitée. S'inspirant des travaux de Nattiez, Smaill & Wiggins (1990) ont adapté l'algorithme de segmentation à l'analyse de pièces polyphoniques en ne prenant en compte que les répétitions strictes et les transpositions tonales.

[134] Vecchione 1984, pp.433-ff.

ressemblance est dépendante du domaine, même si certaines ressemblances comme l'identité, la transposition tonale, la rétrogradation, etc., peuvent être considérées comme générales.

Nattiez[135] a proposé qu'au lieu d'analyser une pièce musicale isolée on parte d'un corpus de pièces apparentées. Il n'est pas prouvé toutefois qu'en mettant les pièces bout à bout on puisse surmonter les limitations de l'algorithme de Ruwet. Nous avons tenté une expérience de ce type sur un petit nombre (environ 10) de variations d'un même *qa'ida* en utilisant une définition formelle des unités signifiantes.[136] Le résultat n'est pas dépourvu d'intérêt mais il est obtenu en formulant des hypothèses sur le langage qui ne sont pas universellement vérifiées.[137]

Diverses méthodes ont été proposées — liées à certains domaines d'application — pour inférer une segmentation pertinente ou un "squelette de structure" à partir d'un ensemble de chaînes.[138]

La méthode que nous avons adoptée pour l'analyse des schémas d'improvisation consiste à analyser un nombre de variations aussi grand que possible en inférant à la fois la **segmentation** des exemples et leur **structure** syntaxique (i.e. une grammaire formelle de type 2). Cette méthode est justifiée par la difficulté, pour un analyste, de segmenter de manière cohérente un petit corpus d'exemples, sauf peut-être pour certains *qa'ida*s qui utilisent des permutations d'unités courtes faciles à identifier parce que communes à d'autres *qa'ida*s. Dans de nombreux cas, le vocabulaire de ces unités (que nous appelons des "**mots**") n'est que partiellement connu. Une hypothèse fausse sur ce vocabulaire peut placer l'analyste devant des problèmes difficiles à résoudre, malgré — et souvent, à cause de — sa connaissance du domaine.

L'expérience a montré que les difficultés de formalisation de schémas d'improvisation portaient moins sur les structures profondes que sur les permutations des "mots". En effet, dans ce dernier cas, l'information fournie par l'expert se ramène à une proposition du type: "prendre tel alphabet et former toutes les permutations possibles de longueur donnée". L'insuffisance des indications fournies tient essentiellement à la difficulté de formaliser des règles d'acceptabilité des permutations.

Les connaissances que l'on cherche à acquérir sont de trois types:

(1) une grammaire décrivant les productions licites;

(2) un monoïde syntaxique à l'aide duquel on peut coder les exemples en faisant apparaître une segmentation acceptable par les experts;

(3) un ensemble de probabilités (grammaire stochastique) ou de poids (grammaire BP) permettant d'affiner (1): le système est alors capable, en fonctionnement stochastique, de produire des exemples "plus ou moins probables" du point de vue de l'expert.

---

135 1975. La proposition était une adaptation d'une idée de Molino.

136 Bel 1987a

137 Par exemple, il est nécessaire de considérer tout silence comme la prolongation de l'événement qui le précède; cette hypothèse n'est pas vraie dans certains *qa'ida*s.

138 Par exemple, Landraud & Chrétienne (1989) proposent une méthode pour découvrir des répétitions approximatives de mots occupant des positions similaires sur une famille de chaînes.

Une première version de l'algorithme permettant de résoudre (1) et (2) a été implémentée en Prolog II, et son *modus operandi* décrit dans un article.[139] Le problème (3) est facile à résoudre avec l'algorithme décrit au chapitre V §6.[140] Dans ce chapitre, la méthode utilisée pour (1) et (2) est introduite formellement.

1. Problème de l'inférence de langage (*the problem of language inference*)

Etant donné un ensemble L, partitionné en *p* classes, de chaînes sur un alphabet A (un **langage**), on cherche à construire des caractérisations minimales de chaque classe pertinentes du point de vue d'un expert du domaine. Caractériser les classes d'un ensemble fini de chaînes est un problème de **discrimination**[141]. Lorsque l'ensemble est infini (on ne connaît pas de borne supérieure de la longueur des chaînes) on dispose d'un **échantillon d'exemples** $\{\Sigma_1,\ldots,\Sigma_p\}$, où Σ_i désigne un ensemble de chaînes appartenant à la classe *i*. On connaît[142] par ailleurs une famille de **fonctions discriminantes** parmi lesquelles on cherche une caractérisation de la partition de L. Ce problème appartient à l'**inférence inductive**.[143]

Supposons ce problème résolu, et soit ϕ la fonction discriminante de la partition de L. Pour toute chaîne *x* de L, $\phi(x)$ retourne un entier qui est le numéro de la classe correspondante. Le résultat de $\phi(x)$ pour $x \notin L$ est ignoré. Lorsque L est infini, ce type de caractérisation ne permet donc pas de se prononcer sur une chaîne arbitraire de A*. Il est indispensable dans ce cas de rechercher des fonctions **caractéristiques**: pour tout $x \notin L$, $\phi(x) \notin \{1,\ldots,p\}$.

Concevoir L comme un ensemble infini revient à dire que, pour toute fonction caractéristique ϕ_{Si} construite à partir d'un échantillon S_i de L, il peut exister une chaîne $x \notin S_i$ telle que $\phi_{Si}(x)$ fournit une réponse incorrecte. Dans ce cas, $\phi_{Si\cup\{x\}} \neq \phi_{Si}$. Les méthodes qui permettent de calculer $\phi_{Si\cup\{x\}}$ connaissant seulement ϕ_{Si}, x et $\phi_L(x)$ (méthodes **incrémentales**) présentent un intérêt évident pour la construction de systèmes d'apprentissage.

Les systèmes de réécriture, plus particulièrement les grammaires formelles, permettent de caractériser des ensembles de chaînes en s'appuyant sur un formalisme susceptible de mettre en évidence des connaissances pertinentes pour un domaine, par exemple les catégories syntaxiques ou le lexique d'un langage. Outre les grammaires[144] il existe

---

139 Kippen & Bel 1989a.

140 Cet algorithme a été implémentée dans la version BP1 du Bol Processor.

141 Une approche consiste par exemple à énumérer les plus courtes sous-chaînes de chaque classe qui ne sont sous-chaînes d'aucune autre classe, et à construire des fonctions discriminantes qui sont des disjonctions d'opérateurs de la forme “présence d'une sous-chaîne” (Guénoche 1989). Nous n'abordons pas ici les problèmes de classification ou de discrimination conceptuelle (construction simultanée des classes et de leurs caractérisations). Nous supposons que la partition est connue ou qu'elle a été construite, par exemple à l'aide d'une fonction de similarité ou de distance entre chaînes.

142 L'existence d'une telle famille dépend en fait de propriétés connues (ou supposées connues) de L et de sa partition; par exemple le type du langage dans la classification de Chomsky.

143 Case 1982; Angluin 1983.

144 et les automates qui leur sont équivalents, cf. Kain 1981.

d'autres classes de fonctions caractéristiques récursives, par exemple les dérivations de **motifs** (*patterns*).[145]

On rappelle au §2 quelques résultats sur les classes de fonctions caractéristiques apprenables. Par “apprenable” on entend ici “identifiable”[146] ou “fortement approchable” à la limite[147], notions qui ont été reformulées depuis d'un point de vue plus général.[148]

Dans un système d'apprentissage incrémental il est nécessaire d'effectuer des inférences “correctes” en présence d'exemples seuls. Nous énonçons au §4 les conditions qui permettent d'éviter toute surgénéralisation dans le cas des langages réguliers.

Nous nous intéressons enfin (§5-ff) à l'identification de langages réguliers décrits par des grammaires de type 2. Ces grammaires font apparaître des règles **lexicales**, dont les membres droits sont des agrégats de symboles qui peuvent avoir un “sens” pour un expert du domaine (§7-8). Les agrégats forment un **lexique** du langage incomplètement connu des informateurs, mais nous supposons que tout informateur est en mesure de valider toute segmentation d'une chaîne du langage proposée par le système. Le système d'apprentissage que nous avons réalisé sur ce modèle fonctionne comme suit:

1) Au départ le système n'a pas de connaissance du domaine, sauf certaines propriétés relatives à la classe de langage à inférer, ex. langage fini, etc.
2) On soumet au système un échantillon d'exemples. Le système construit un automate “presque minimal” qui reconnaît exactement les exemples et qui respecte la segmentation de chaque chaîne. Toute décision liée à la segmentation est validée au moyen de questions posées à l'informateur.
3) Etape facultative: le système complète la segmentation des chaînes connues en utilisant les agrégats (mots ou suites de mots) déjà connus. La nouvelle segmentation est soumise à l'informateur.
4) Généralisation: le système recherche une partition des états de l'automate construit en (2) afin de construire un automate quotient. L'espace de recherche (i.e. l'ensemble des partitions plausibles) est limité par des contraintes liées aux propriétés du langage, et ordonné partiellement par des heuristiques. Toute équivalence entre deux états qui ne peut être déduite d'une propriété du langage est validée par l'examen des chaînes nouvelles reconnues par l'automate quotient (production de contrexemples par **oracle**, §9-10).
5) On reprend les étapes (2), (3), (4) jusqu'à ce que tout échantillon d'exemples soumis au système soit reconnu par l'automate.

2. Paradigmes d'apprenabilité (*learning paradigms*)

Gold[149] s'est intéressé au cas suivant: un langage fini ou infini L partitionné en p classes, ou encore le monoïde A* partitionné en (p+1) classes telles que L est la réunion de p classes. Caractériser une telle partition revient à chercher une fonction caractéristique de

[145] Angluin 1980b.
[146] Gold 1967.
[147] Bierman 1972.
[148] Case 1982.
[149] 1967; 1978.

chaque classe à partir d'un échantillon d'**exemples** (des chaînes de cette classe) et de **contrexemples** (des chaînes d'autres classes).

On dit qu'un langage L est **identifiable à la limite** s'il existe un système qui, quelle que soit la séquence d'exemples et de contrexemples, propose toujours la même fonction caractéristique ϕ après un nombre fini d'étapes, et si par ailleurs cette fonction caractérise L:

$$\exists i0 \in \mathbb{N} \mid \forall j \geq 0 , \ \phi_{i0+j} = \phi_{i0}$$
$$\text{et } L(\phi_{i0}) = L$$

Autrement dit, le système commet toujours un nombre fini d'erreurs avant de reconnaître L.

Le théorème de Gold[150], établit les limites d'apprenabilité de classes de langages dans une hiérarchie compatible[151] avec celle de Chomsky:

> (1) La classe la plus générale de langages formels identifiables à la limite à l'aide d'exemples ou de contrexemples est celle des langages primitivement récursifs.[152]
> (2) Seule la classe des langages finis est identifiable à la limite à partir d'exemples seuls.

Selon la proposition (2) du théorème, toute classe de langages qui contient **tous** les langages finis et au moins un langage infini n'est pas identifiable à la limite à partir d'exemples seuls. Certaines classes de langages non comparables avec celle des langages finis sont toutefois apprenables à la limite à partir d'exemples seuls, par exemple les **langages pivots**, sous-classe des langages de type 2,[153] les **langages de motifs**[154] et les **langages k-réversibles**[155].

Un autre paradigme d'apprenabilité de langage moins contraignant que celui de l'identification est celui d'**approche (forte) à la limite**[156]. Informellement, à chaque étape le système doit proposer une fonction caractéristique de plus en plus proche de celle qui est recherchée.[157] On montre que tout système de réécriture décidable peut être approché fortement à la limite à partir d'exemples seuls.

---

[150] Gold 1967, p.452; Bierman 1972, p.33

[151] C'est à dire toute famille indexée de classes $(C_1,...,C_n)$ telle que C_i contient strictement C_{i+1} et pour toute classe $\mathcal{L}_i$ de la classification de Chomsky il existe j tel que $C_j = \mathcal{L}_i$.

[152] Les langages primitivement récursifs sont ceux générés par une classe énumérable de grammaires décidables.

[153] Fu & Booth 1975a, p.105

[154] Angluin 1980b.

[155] Angluin 1982; Berwick & Pilato 1987.

[156] Bierman 1972, p.34

[157] La distinction de ces deux paradigmes est de peu d'intérêt pratique: ni l'informateur ni le système d'apprentissage ne savent si le langage a été identifié ou non. Même si l'hypothèse reste stable on ne peut garantir qu'un nouvel exemple ou contrexemple ne viendra pas l'infirmer.

3. Définitions et notations (*definitions and notations*)

3.1 Automate fini, accepteur fini (*finite automaton, finite acceptor*)

Nous appellons **automate fini** le sextuplet (A,E,S,T,Y,φ) tel que:

> $A = \{a_1, ..., a_p\}$ est un alphabet de cardinal *p*;
> $E = \{e_1, ..., e_n\}$ est un ensemble fini d'*états*;
> $S = \{s_1, ..., s_q\}$ est une partie non vide de E appelée ensemble des **états initiaux**;
> $T = \{t_1, ..., t_p\}$ est un ensemble fini de *p* correspondances de E dans lui-même appelé **ensemble des transitions**;
> φ est une application (univoque), que l'on appelle **fonction de sortie**, de E dans un ensemble quelconque Y.

L'automate est **déterministe** si les deux conditions suivantes sont remplies:

> (1) $\forall\ i \le p,\ \forall\ e_j \in E,\ card(t_i(e_j)) < 2$ (2) $card(S) \le 1$

L'inégalité dans la condition (2) permet de classer l'**automate vide** (*empty automaton*) parmi les automates déterministes. A partir de l'ensemble des transitions $T = \{t_1, ..., t_p\}$ on peut construire la **fonction de transition**

$$f: \quad E \times A^* \longrightarrow \mathcal{P}(E)$$

que nous définissons récursivement:

> $\forall\ u \in A^*$
> (1) s'il existe $a_i \in A$ tel que $u = a_i$, alors $\forall\ e_j \in E$, $f(u,e_j) = t_i(e_j)$;
> (2) si $u = a_i v$ avec $a_i \in A$ et $v \in A^*$, alors $\forall\ e_j \in E$,
> $f(u,e_j) = \{e_k \mid \exists\ l,\ e_l \in t_i(e_j)$ et $e_k \in f(v,e_l)\}$

Cette fonction fait correspondre, à tout état e_j et à toute chaîne *u*, l'ensemble des états que l'on peut atteindre en parcourant l'automate à partir de e_j de sorte qu'à la k-ième étape on ne franchisse une transition que si elle est étiquetée par le k-ième symbole de u.

A tout automate fini (A,E,S,T,Y,φ) on peut associer un **graphe p-coloré** dont les sommets sont les états $\{e_1,..., e_n\}$ étiquetés par $\varphi(e_i)$, et dont les arcs de couleur *q* sont le graphe de la correspondance $t_q \in T$.

On appelle **accepteur fini** (*finite-state acceptor*) un automate fini (déterministe ou non) (A,E,S,T,Y,φ) tel que $Y = \{0,1\}$. Tout état $e_i \in E$ tel que $\varphi(e_i) = 1$ est un état **acceptable** ou **final** (*accepting state*).

Une chaîne $x = x_1...x_m$ de A* est **acceptée** (reconnue) par l'accepteur $\mathcal{A}$ si et seulement s'il existe une suite d'états de $\mathcal{A}$: $(e_1,...,e_m)$ telle que:

> (1) $e_1 \in S$;
> (2) $\forall\ k \in (1,...,m)$, $e_{k+1} \in t_i(e_k)$ pour *i* tel que $a_i = x_k$ et $t_i \in T$;
> (3) $\varphi(e_m) = 1$, c'est à dire e_m est un état acceptable de $\mathcal{A}$.

Nous appelons **chemin de x dans** $\mathcal{A}$, et nous notons $E_{\mathcal{A}}(x)$, la suite $(e_1,...,e_m)$.

Un langage L sur l'alphabet A est **accepté** (reconnu) par $\mathcal{A}$ si et seulement si toute chaîne du langage est acceptée par $\mathcal{A}$. Si de toute chaîne de A* acceptée par $\mathcal{A}$ appartient à L, on dit que $\mathcal{A}$ reconnaît **exactement** L. On note L($\mathcal{A}$) le langage exactement reconnu par $\mathcal{A}$.

3.2 k-suiveur, k-leader (*k-follower*, *k-leader*)

Soit (A,E,S,T,Y,φ) un automate fini tel que A = {a_1, ..., a_p}, E = {e_1, ..., e_n}, T = {t_1,...,t_p}. Pour tout état $e_i \in E$, un symbole a_x de A est un **1-suiveur** de e_i si et seulement si:

$$\exists\, j \in (1,...,n) \mid e_j \in t_x(e_i)$$

Soit $k \in N$. Une chaîne $u \in A^*$ est un **k-suiveur** de e_i dans un des deux cas suivants:

(1) k = 0 et u = λ ;

(2) u = a_x v tel que $a_x \in A$, $v \in A^*$, a_x est un **1-suiveur** de e_i , et, pour *j* tel que $e_j \in t_x(e_i)$, k' = k-1, *v* est un k'-suiveur de e_j .

Il est facile de montrer que dans tous les cas |u| = k. On définit un **k-leader** de manière duale: il suffit de remplacer, dans la définition précédente, "$e_j \in t_x(e_i)$" par "$e_i \in t_x(e_j)$", et "a_x v" par "v a_x".

3.3 Accepteurs finis isomorphes, sous-accepteur, accepteur minimal (*isomorphic finite acceptors*, *sub-acceptor*, *minimal acceptor*)

On dit que deux accepteurs finis: (A,E,S,T,{0,1},φ) et (A,E',S',T',{0,1},φ') avec n = card(E) et p = card(A) sont **isomorphes** si et seulement s'il existe une bijection *h* de E vers E' telle que:

$$h(S) = S'$$
$$\forall\, i \in (1,...,n),$$
$$\varphi'(h(e_i)) = \varphi(e_i) \text{ et } \forall\, j \in (1,...,p)\ ,\ t'_j(h(e_i))) = h(t_j(e_i))$$

Informellement, les accepteurs sont identiques à l'étiquettage des états près.

(A,E',S',T',{0,1},φ') est un **sous-accepteur** de (A,E,S,T,{0,1},φ), avec n = card(E) et n' = card(E') si et seulement si:

$$E \supseteq E',\ S \supseteq S'$$
$$\forall\, i \in (1,...,n')$$
$$\varphi'(e_i) = \varphi(e_i) \text{ et } \forall\, j \in (1,...,p)\ ,\ t_j(e_i) \supseteq t'_j(e_i)$$

On montre facilement que si $\mathcal{A}'$ est un sous-accepteur de $\mathcal{A}$, L($\mathcal{A}$) ⊇ L($\mathcal{A}'$).

Soient $\mathcal{A}$ = (A,E,S,T,{0,1},φ) un accepteur fini et E' une partie de E avec n' = card(E'). On dit que $\mathcal{A}'$ = (A,E',S',T',{0,1},φ') est un **sous-accepteur** de $\mathcal{A}$ **induit** par E' si et seulement si:

$S' = S \cap E'$
$\forall\ i \in (1,...,n')$
$\varphi'(e_i) = \varphi(e_i)$ et $\forall\ j \in (1,...,p)$, $t_j(e_i) = t'_j(e_i) \cap E'$

Un état e_i d'un accepteur fini $\mathcal{A} = (A,E,S,T,\{0,1\},\varphi)$ est **inutile** si le langage reconnu par le sous-accepteur induit par $E \setminus \{e_i\}$ est exactement celui reconnu par $\mathcal{A}$. Dans le cas contraire, l'état considéré est **utile**. Tout accepteur fini qui ne contient pas d'état inutile est appelé accepteur **restreint** (*stripped*). Un accepteur $(A,E,S,T,\{0,1\},\varphi)$ est dit **minimal** pour un langage L s'il reconnaît exactement L et s'il n'existe pas d'accepteur $(A,E',S',T',\{0,1\},\varphi')$ reconnaissant exactement L tel que $card(E') < card(E)$. Tout accepteur minimal d'un langage est nécessairement restreint, mais la réciproque n'est pas vraie en général.

3.4 Quotient d'un accepteur fini par une relation d'équivalence sur ses états (*quotient of a finite acceptor by an equivalence relation on its states*)

Soit un accepteur fini $\mathcal{A} = (A,E,S,T,\{0,1\},\varphi)$ et une relation d'équivalence totale $\mathcal{R}$ sur l'ensemble des états E. On appelle **quotient** de $\mathcal{A}$ par $\mathcal{R}$, et on note $\mathcal{A}/\mathcal{R}$ l'accepteur fini $(A,E',S',T',\{0,1\},\varphi')$ tel que:

E' est l'ensemble des classes $E/\mathcal{R}$;
$\forall\ e_i \in S$, $\forall\ e'_j \in E'$, $e_i \in e'_j$ => $e'_j \in S'$;
$\forall\ e_i \in E$, $\forall\ e'_j \in E'$, $e_i \in e'_j$ et $\varphi(e_i) = 1$ => $\varphi(e'_j) = 1$;
$\forall\ e_i , e_j \in E$, $\forall\ t_q \in T$, $\forall\ e'_k, e'_l \in E'$,
$t'_q \in T'$, $e_i \in e'_k$, $e_j \in e'_l$ et $e_j \in t_q(e_i)$ => $e'_l \in t'_q(e'_k)$

Il est facile de montrer que, si *f* est la fonction de transition de $\mathcal{A}$, *f'* celle de $\mathcal{A}/\mathcal{R}$, pour toute chaîne $u \in A^*$, s'il existe $e_i \in E$ tel que $f(u,e_i) \neq \emptyset$, alors il existe $e_j \in S$ tel que $f(u,e_j) \neq \emptyset$. Autrement dit, à tout parcours sur $\mathcal{A}$ correspond un parcours sur $\mathcal{A}/\mathcal{R}$ étiqueté identiquement par les symboles de *u*. En particulier, si $e_i \in S$ et $\varphi(f(u,e_i)) = 1$ alors $u \in L$; ce qui prouve que tout langage accepté par $\mathcal{A}$ est aussi accepté par $\mathcal{A}/\mathcal{R}$. La réciproque n'est généralement pas vraie. On peut donc écrire

$$L(\mathcal{A}/\mathcal{R}) \supseteq L(\mathcal{A})$$

où L(X) désigne le langage exactement reconnu par l'accepteur X.

On peut prouver que si $\mathcal{A}$ est déterministe alors $\mathcal{A}/\mathcal{R}$ l'est aussi.[158]

3.5 Préfixe, suffixe, tête et queue d'un langage (*prefix, suffix, head, tail of a language*)

On appelle **préfixe** (resp. **suffixe**) **d'une chaîne** *u* sur l'alphabet A toute chaîne *v* (resp. w) telle que u = v w. Nous appelons **préfixe** (resp. **suffixe**) **d'un langage** L l'ensemble des préfixes (resp. suffixes) des chaînes de L, à savoir:

[158] Bel 1989c, p.40

Pref(L) =

$\{u \in A^* \mid \exists\, v \in A^*,\ u\,v \in L\}$; Suf(L) = $\{v \in A^* \mid \exists\, u \in A^*,\ u\,v \in L\}$

Nous appelons **tête** (resp. **queue**) d'une chaîne *u* dans un langage L, et nous notons Head(L,u) (resp. Tail(L,u)) l'ensemble des chaînes *v* de A* telles que *v u* (resp. *u v*) est une chaîne de L:

Head(L,u) = $\{v \in A^* \mid v\,u \in L\}$; Tail(L,u) = $\{v \in A^* \mid u\,v \in L\}$

On a toujours: Head(∅,u) = Tail(∅,u) = ∅.

3.6 Accepteur canonique d'un langage régulier (*canonic acceptor of a regular language*)

Soit L un langage régulier sur l'alphabet A. Notons Pref(L) l'ensemble des préfixes de L, et Tail(L,u) la queue d'une chaîne *u* dans L. L'**accepteur canonique** de L est l'accepteur fini $\mathcal{A}$(L) = (A,E,S,T,{0,1},φ) avec p = card(A) et T = $\{t_1,\ldots,t_p\}$ tel que:

(1) $E = \{Tail(L,u) \mid u \in Pref(L)\}$;
(2) $S = \{Tail(L,\lambda)\}$ lorsque $L \neq \emptyset$; $S = \emptyset$ dans le cas contraire ;
(3) $\forall e_i \in E$, $e_i = Tail(L,u)$ avec $u \in L$ => $\varphi(e_i) = 1$;
(4) $\forall e_i = Tail(L,u) \mid u \in Pref(L)$,
$\forall a_j \in A$, $u\,a_j \in Pref(L)$ => $t_j(e_i) \supseteq Tail(L,u\,a_j)$.

Par extension, tout accepteur fini d'un langage L est appelé **canonique** s'il est isomorphe de l'accepteur canonique $\mathcal{A}$(L).

On peut prouver que l'accepteur canonique est déterministe, reconnaît exactement L, et qu'il est minimal.[159]

3.7 Accepteur préfixe arborescent d'un langage fini (*tree prefix acceptor of a finite language*)

Soit L un langage **fini** sur l'alphabet A. L'accepteur fini $\mathcal{A}pref$(L) = (A,E,S,T,{0,1},φ) tel que

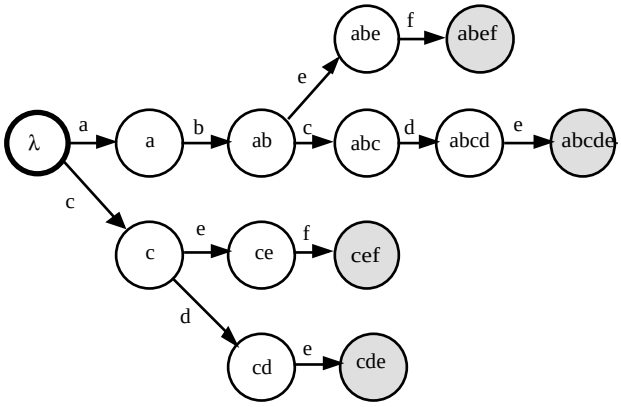
(1) $E = Pref(L)$;
(2) $S = \{Tail(L,u) \mid u \in L\} = \{\lambda\}$;
(3) $\forall e_i \in E$, $e_i \in L$ <=> $\varphi(e_i) = 1$; sinon, $\varphi(e_i) = 0$;
(4) $\forall e_i = Tail(L,u) \mid u \in Pref(L)$,
$\forall a_j \in A$, $u\,a_j \in Pref(L)$ => $t_j(e_i) \supseteq Tail(L,u\,a_j)$.

est appelé **accepteur préfixe arborescent** de L. L'identité de la proposition (4) pour $\mathcal{A}pref$(L) et pour l'accepteur canonique de L permet de montrer que $\mathcal{A}pref$(L) est déterministe et qu'il accepte exactement L. L'ensemble des états finals est L, qu'il ne faut pas confondre avec {L}, l'état initial de l'accepteur canonique de L.

Exemple: $\mathcal{A}pref$({abcde, abef, cef, cde}) est:

---

[159] Bel 1989b, p.41



Par convention, nous hâchurons les états e_i tels que $\varphi(e_i) = 1$ (états acceptables) et nous cerclons en gras les états initiaux.

4. Généralisation d'une fonction caractéristique de langage régulier (*generalizing the characteristic function of a regular language*)

Dans la suite nous nous intéressons au cas où le monoïde A* est partitionné en deux classes, à savoir un langage régulier (inconnu) L et son complément. Etant donné un échantillon d'exemples S_i du langage, on peut construire l'accepteur préfixe arborescent $\mathcal{A}pref(S_i)$ qui reconnaît exactement S_i. **Généraliser** cette fonction caractéristique revient à trouver un accepteur fini qui reconnaisse un langage M tel que $L \supseteq M \supseteq S_i$.

Théorème VI.1[160]

> Soient S_i un échantillon d'exemples d'un langage régulier inconnu L, et $\mathcal{A}pref(S_i) = (A,E,\{\lambda\},T,\{0,1\},\varphi)$ son accepteur préfixe arborescent. Soit $\mathcal{R}$ la congruence[161] sur A* telle que: $\mathcal{R}(u,v) <=> Tail(L,u) = Tail(L,v)$.
> L'accepteur quotient $\mathcal{A}' = \mathcal{A}pref(S_i) / \mathcal{R}$ est isomorphe à un sous-accepteur de l'accepteur canonique $\mathcal{A}(L)$.

Tout langage reconnu par $\mathcal{A}$ est aussi reconnu par $\mathcal{A}/\mathcal{R}$. Par conséquent, $L \supseteq L(\mathcal{A}/\mathcal{R}) \supseteq S_i$. La généralisation n'est possible toutefois que si $\mathcal{R}$ est connue, ce qui n'est pas vrai en général.

Le théorème suivant[162] permet d'énoncer la condition à remplir pour que cette construction réalise effectivement l'identification à la limite. Rappelons que, pour tout langage L_i d'une famille énumérable, on appelle **échantillon caractéristique** un

---

[160] *op.cit.* p.45

[161] $\mathcal{R}$ est aussi une relation d'équivalence sur E puisque E = Pref(L).

[162] Angluin 1980a, p.121

ensemble fini (maximal) de chaînes S_c tel que $L_i \supseteq S_c$, et pour tout $j \geq 1$, si L_j contient S_c alors L_j n'est pas inclus dans L_i .([163])

Théorème VI.2

> Une famille énumérable de langages récursifs non vides (L_1, ..., L_i, ...) est identifiable à la limite à partir d'exemples seuls si et seulement s'il existe une procédure effective qui à toute valeur de $i \geq 1$ associe un échantillon caractéristique de L_i .

Il est clair que dans ce cas lorsque $S_i \supseteq S_c$ alors M = L.

5. Accepteur "presque minimal" d'un langage fini (*"almost minimal" acceptor of a finite language*)

Le théorème VI.1 ne permet pas en général de réaliser un système d'apprentissage incrémental: à l'étape suivant la généralisation on ne peut pas remplacer $\mathcal{A}pref(S_i)$ par $\mathcal{A}pref(S_i)/\mathcal{R}$, puisque la construction doit se faire sur l'accepteur préfixe arborescent qui reconnaît exactement S_{i+1}. Nous présentons maintenant une méthode incrémentale qui, de plus, tente de minimiser la représentation courante de l'accepteur reconnaissant exactement S_i . Nous appelons "**presque minimal**" un tel accepteur. La généralisation proprement dite est effectuée indépendamment de cette construction (voir §9).

Soient S_i un échantillon de *i* exemples du langage L, et $\mathcal{A}_i = (A,E,S,T,\{0,1\},\varphi)$ un accepteur fini (non déterministe)[164] reconnaissant exactement S_i . Appelons x_{i+1} le (i+1)ème exemple présenté. Nous cherchons à construire un accepteur $\mathcal{A}_{i+1}$ qui reconnaisse exactement $S_i \cup \{x_{i+1}\}$ tout en remplissant certaines conditions de minimalité. Construisons en premier lieu l'accepteur $\mathcal{A}' = (A,E\cup E',S\cup\{e'_0\},T',\{0,1\},\varphi')$ tel que:

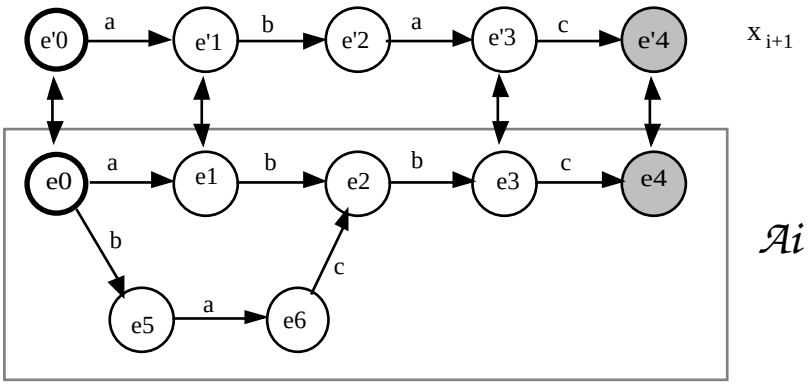
> $E' = Pref(\{x_{i+1}\})$; $E \cap E' = \varnothing$;
> $\forall e \in E'$, $e \neq x_{i+1}$ <=> $\varphi'(e) = 0$, et $e = x_{i+1}$ <=> $\varphi'(e) = 1$;
> $\forall e \in E$, $\varphi'(e) = \varphi(e)$;
> $\forall t'_j \in T'$,
> $\quad \forall e \in E$, $t'_j(e) = t_j(e)$;
> $\quad \forall e \in E'$, $t'_j(e) = Tail(\{x_{i+1}\}, e\, a_j)$ avec $a_j \in A$

Il est facile de prouver que $\mathcal{A}'$ reconnaît exactement $S_i \cup \{x_{i+1}\}$. La figure ci-dessous illustre le cas où S_i = {abbc,bcbc} et x_{i+1} = abac :

---

[163] Autrement dit, L_i est le plus "petit" langage de la famille qui contienne S_c.

[164] On a évidemment $S_0 = \varnothing$, et $\mathcal{A}_0$ est l'accepteur vide.



Pour tout état *e* d'un accepteur (non-déterministe) (A,E,S,T,{0,1},φ) dont *f* est la fonction de transition, nous appelons H(e) (resp. T(e)) les langages générés en considérant cet état comme unique état acceptable (resp. initial), soit:

$$H(e) = \{u \in A^* \mid \exists e_0 \in S\ ,\ e \in f(e_0,u)\ \}$$
$$T(e) = \{v \in A^* \mid \exists e_f \in f(e,v)\ ,\ \varphi(e_f) = 1\ \}$$

Nous considérons maintenant la relation $\mathcal{R}r$ sur l'ensemble des états E telle que:

$\forall e_1,e_2 \in E,\ \mathcal{R}r(e_1,e_2) <=>$
$H(e_1) = H(e_2)$ ou $T(e_1) = T(e_2)$, et $\varphi(e_1) = \varphi(e_2)$

Il est évident qu'on a $\mathcal{R}r(e_2,e_1)$ pour tout couple d'états initiaux (resp. acceptables), les fonctions H (resp. T) valant {λ} dans ce cas. $\mathcal{R}r$ est une relation de **ressemblance** (pas nécessairement transitive).

Dans la figure ci-dessus, la relation $\mathcal{R}r$ est indiquée à l'aide d'une flèche en gras.

On construit maintenant une relation d'équivalence $\mathcal{R}0$ comme suit: les états de l'accepteur sont numérotés et l'on obtient :

$\forall e_i,e_j \in E,\ \mathcal{R}0(e_i,e_j) <=>$
$\mathcal{R}0(e_j,e_i)$ ou
$i \le j$, $\mathcal{R}r(e_i,e_j)$, et l'on n'a pas $\mathcal{R}0(e_i,e_k)$ avec $k < j$ ni $\mathcal{R}0(e_k,e_i)$ avec $k < i$.

Puisqu'on ne peut avoir $\mathcal{R}0(e_i,e_j)$ et $\mathcal{R}0(e_j,e_k)$ avec $i \neq j$, $j \neq k$ et $i \neq k$, la transitivité de $\mathcal{R}0$ est toujours vérifiée. Il n'est pas difficile de démontrer que $\mathcal{A}'/\mathcal{R}0$ reconnaît exactement $S_i \cup \{x_{i+1}\}$.

Pour une relation $\mathcal{R}r$ donnée, $\mathcal{R}0$ dépend de la numérotation des états. Dans le cas de l'accepteur $\mathcal{A}'$, on numérote les états en commençant par E', dans l'ordre des longueurs croissantes des préfixes:

(1) $\forall e_i,e_j \in E \cup E'$, $e_i \in E$ et $e_j \in E'$ => $i > j$
(2) $\forall e_i,e_j \in E'$, $|e_j| > |e_i|$ => $i > j$

Dans la proposition (2) il faut comprendre que e_i et e_j ne sont autres que des préfixes de x_{i+1}, c'est à dire des chaînes sur A. Avec cette numérotation on a toujours $\mathcal{R}0(e_i,e_j)$ pour e_i, e_j états initiaux, mais pas nécessairement lorsque $\varphi(e_i) = \varphi(e_i) = 1$ (états acceptables).

Pour l'accepteur de la figure précédente on a $\mathcal{R}0 = \mathcal{R}r$. Nous montrons plus loin que cette équivalence n'est pas toujours vérifiée.

On peut montrer que si l'on a $\mathcal{R}0(e_j,e_k)$ avec $e_j \in E'$, l'ensemble des états nouvellement créés, on a nécessairement $e_k \in E$ et:

> Si $H(e_j) = H(e_k)$ alors $\mathcal{R}0(e_{j'},e_{k'})$ pour $e_{j'}$ et $e_{k'}$ 1-leaders de e_j et e_k respectivement;
> Si $T(e_j) = T(e_k)$ alors $\mathcal{R}0(e_{j'},e_{k'})$ pour $e_{j'}$ et $e_{k'}$ 1-suiveurs de e_j et e_k respectivement.

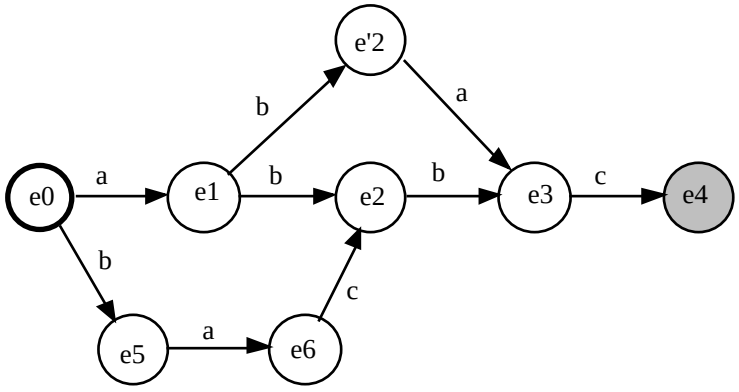
Par exemple, $\mathcal{R}0(e'_1,e_1) \Rightarrow \mathcal{R}0(e'_0,e_0)$, et $\mathcal{R}0(e'_3,e_3) \Rightarrow \mathcal{R}0(e'_4,e_4)$ ci-dessus. Pour calculer $\mathcal{R}0$il suffit donc d'énumérer les préfixes *u* de x_{i+1} , par longueur croissante à partir de λ, en cherchant le sous-ensemble E_u de E tel que $\forall e \in E_u$, $H(e) = \{u\}$. Concrètement, on cherche l'état e_m de E le plus éloigné d'un[165] état initial de $\mathcal{A}_i$ tel que l'unique élément de $H(e_m)$ est un préfixe de x_{i+1} . (Si cet état est final, x_{i+1} est reconnu par $\mathcal{A}_i$ et on peut passer à x_{i+2} .) On cherche de même l'état e_p de E le plus éloigné d'un état acceptable de $\mathcal{A}_i$ tel que l'unique élément de $T(e_p)$ est un suffixe de x_{i+1} . Dans cette énumération on a trouvé tous les couples (e_j,e_k) qui satisfont la relation $\mathcal{R}r$, avec $e_j \in E'$. L'accepteur $\mathcal{A}_i$ étant simplifié (algorithme ci-dessous), ces couples sont les seuls qui peuvent satisfaire la relation $\mathcal{R}0$. On construit enfin la relation $\mathcal{R}0$ en énumérant dans l'ordre les couples satisfaisant $\mathcal{R}r$.

5.1 Simplification de l'accepteur

L'algorithme de simplification de l'accepteur $\mathcal{A}'$ est le suivant:

> Tant qu'il existe e_i, $e_j \in E$ tels que $\mathcal{R}0(e_i,e_j)$ et $i \neq j$, où E est l'ensemble des états de $\mathcal{A}'$, remplacer l'accepteur $\mathcal{A}'$ par $\mathcal{A}'/\mathcal{R}0$ et recalculer $\mathcal{R}0$.

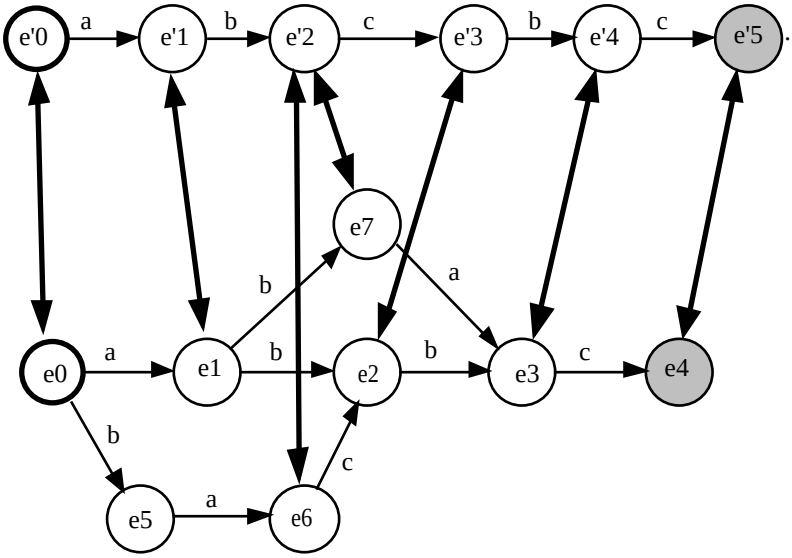
L'algorithme s'arrête nécessairement puisque card(E) décroît strictement et qu'on ne peut avoir $\mathcal{R}0(e_i,e_j)$ et $i \neq j$ lorsque card(E) = 1. L'accepteur fini obtenu après simplification est $\mathcal{A}_{i+1}$. Pour l'accepteur $\mathcal{A}'$ ci-dessus on obtient $\mathcal{A}_3$:



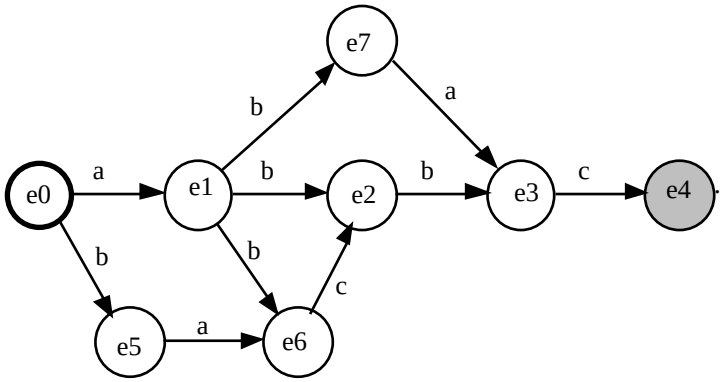
Nous montrons maintenant un cas où l'on n'a pas $\mathcal{R}r = \mathcal{R}0$. Supposons que l'exemple suivant soit “abcbc”. Nous montrons ci-dessous l'accepteur $\mathcal{A}'$ et la relation $\mathcal{R}r$:

---

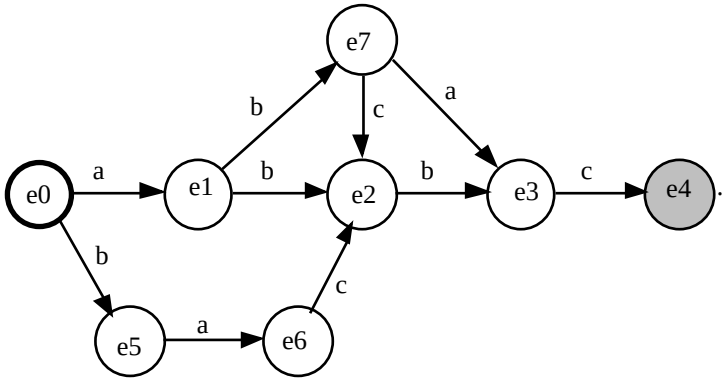
[165] Nous montrons plus loin que cet état initial est unique.



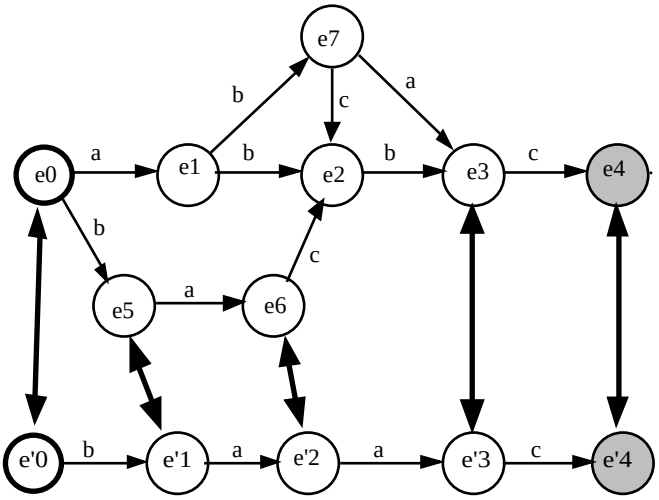
Pour construire $\mathcal{R}0$ on examine dans l'ordre les couples (e'_0,e_0), (e'_1,e_1), (e'_2,e_6), (e'_2,e_7), (e'_3,e_2), (e'_4,e_3) et (e'_5,e_4). On ne peut avoir à la fois $\mathcal{R}0(e'_2,e_6)$ et $\mathcal{R}0(e'_2,e_7)$, c'est donc le premier couple énuméré qui est choisi. L'accepteur $\mathcal{A}_4$ est dans ce cas:



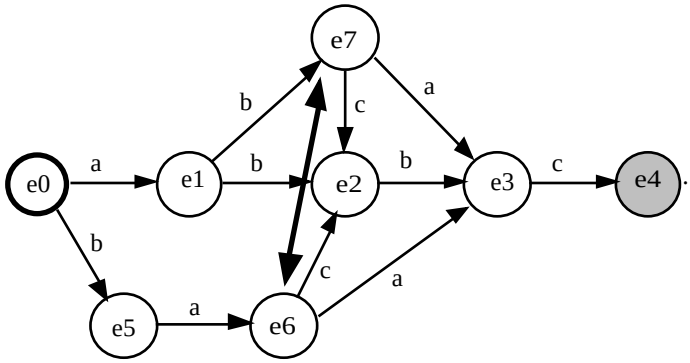
En choisissant une autre numérotation de E telle que l'ordre (e'_2,e_6), (e'_2,e_7) soit inverse, on obtient pour $\mathcal{A}_4$:



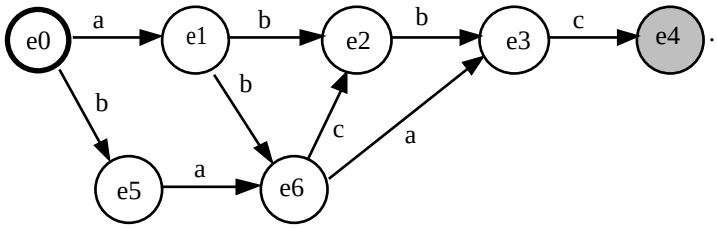
Gardons ce dernier accepteur et supposons que le prochain exemple soit “baac”. On obtient alors



où $\mathcal{R}r = \mathcal{R}0$, ce qui donne après simplification:



On a mis en évidence un nouveau couple (e_6,e_7) vérifiant $\mathcal{R}r$ et $\mathcal{R}0$. En effet, T(e_6) = T(e_7) = {cbc,ac}. La simplification se poursuit donc et on obtient finalement $\mathcal{A}_5$:



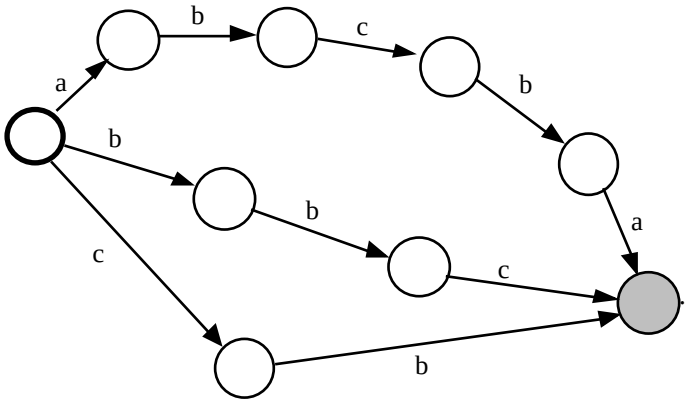
5.2 Complexité de l'accepteur construit

Dans l'exemple que nous avons traité ci-dessus, la dernière simplification n'aurait pas été possible avec le premier choix de numérotation de E. Le nombre d'états de l'accepteur construit dépend donc en général de la numérotation, c'est à dire pratiquement de l'ordre dans lequel les exemples sont fournis.

Dans le cas le plus défavorable, la relation $\mathcal{R}r$ n'est vérifiée que pour les états initiaux et les états finaux. On obtient alors un accepteur trivial $\mathcal{A}_T$ tel que:

> $\forall$ e $\in$ E ,
> e $\notin$ S et φ(e) = 0 <=> card(H(e)) = card(T(e)) = 1

comme par exemple:



Le nombre d'états de l'accepteur trivial $\mathcal{A}_T$ est 2+n-i, où *i* est le nombre de chaînes et *n* la somme des longueurs des chaînes. $\mathcal{A}_T$ est nécessairement déterministe: il n'a qu'un état initial, et si l'on avait deux transitions de même étiquette de cet état initial vers deux états distincts e_1 et e_2, on aurait alors $H(e_1) = H(e_2)$ et $T(e_1) \supseteq T(e_2)$, soit $\mathcal{R}r(e_1,e_2)$ et $\mathcal{R}0(e_1,e_2)$; la simplification permettrait donc de fusionner e_1 et e_2 . De manière similaire on peut montrer que l'accepteur inverse $\mathcal{A}_T^{-1}$ est aussi déterministe.

Théorème VI.3

> Tout accepteur trivial construit par l'algorithme est canonique.

Preuve

Soit $\mathcal{A}pref(S_i)$ l'accepteur préfixe arborescent du langage S_i . Puisque $\mathcal{A}_T^{-1}$ est déterministe, la congruence $\mathcal{R}_t$ sur A* telle que $\mathcal{R}_t(u,v)$ <=> $Tail(S_i,u) = Tail(S_i,v)$ n'est vérifiée que pour $u = v = \lambda$ ou bien $u,v \in S_i$. L'ensemble des états de $\mathcal{A}pref(S_i)$ est par définition l'ensemble des préfixes de S_i , que nous notons $Pref(S_i)$. La relation $\mathcal{R}_t$ n'est vraie, dans l'ensemble des états de $\mathcal{A}pref(S_i)$, que pour l'état initial $\{\lambda\}$ avec lui-même, et pour l'ensemble des états finaux S_i . L'accepteur quotient $\mathcal{A}pref(S_i)/\mathcal{R}_t$ est donc exactement l'accepteur trivial $\mathcal{A}_T$. D'après le théorème VI.1, cet accepteur est un sous-accepteur de l'accepteur canonique du langage S_i . Puisque $\mathcal{A}_T$ reconnaît S_i , il s'agit de l'accepteur canonique. ■

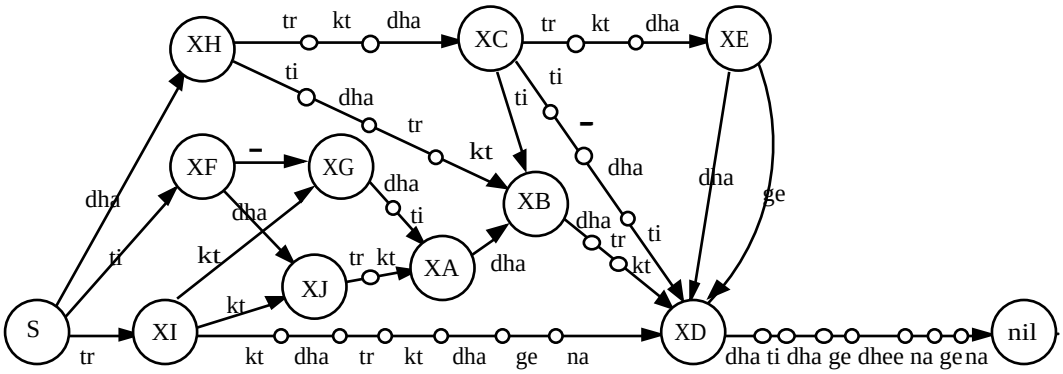
Nous venons de montrer que l'algorithme ne peut construire l'accepteur trivial que lorsque cet accepteur est lui-même canonique et donc minimal. Ce résultat ne dépend pas de l'ordre de présentation des exemples. En dehors de ce cas particulier l'accepteur construit peut se “rapprocher” de l'accepteur canonique de S_i lorsque les exemples ont des préfixes ou des suffixes communs.

6. Exemple musical (*musical example*)

Dix chaînes qui représentent les premières lignes de dix variations d'un *qa'ida* sont proposées.[166] Chaque syllabe est un symbole terminal. Le tiret est un silence. L'échantillon d'exemples est le suivant:

1. tr kt dha ti dha dha tr kt dha ti dha ge dhee na ge na
2. ti dha tr kt dha dha tr kt dha ti dha ge dhee na ge na
3. dha tr kt dha ti dha tr kt dha ti dha ge dhee na ge na
4. dha tr kt dha tr kt dha ge dha ti dha ge dhee na ge na
5. dha tr kt dha tr kt dha dha dha ti dha ge dhee na ge na
6. dha tr kt dha ti - dha ti dha ti dha ge dhee na ge na
7. ti - dha ti dha dha tr kt dha ti dha ge dhee na ge na
8. dha ti dha tr kt dha tr kt dha ti dha ge dhee na ge na
9. tr kt tr kt dha dha tr kt dha ti dha ge dhee na ge na
10. tr kt dha tr kt dha ge na dha ti dha ge dhee na ge na

et l'accepteur résultant:



7. Connaissances lexicales (*lexical knowledge*)

Sur l'accepteur précédent seuls les états correspondant à des noeuds du graphe coloré ont été étiquetés. Cette disposition suggère d'écrire la grammaire sur deux “niveaux”:

[166] Kippen & Bel 1989a, pp.204-5.

S —> TE1 XI
XI —> TA7 XD
XD —> TA8
XI —> TF1 XJ
XJ —> TC2 XA
XA —> TA1 XB
XB —> TB3 XD
XI —> TF1 XG
XG —> TB2 XA
S —> TA1 XH
XH —> TF4 XB
XH —> TA3 XC
XC —> TE4 XD
XC —> TA3 XE
XE —> TA1 XD
XE —> TC1 XD
XC —> TB1 XB
S —> TB1 XF
XF —> TA1 XJ
XF —> TD1 XG

TA7 —> ktdhatrktdhagena
TC2 —> trkt
TE1 —> tr
TF1 —> kt
TF4 —> tidhatrkt
TD1 —> -
TB2 —> dhati
TE4 —> ti-dhati
TC1 —> ge
TB3 —> dhatrkt
TA8 —> dhatidhagedheenagena
TA3 —> trktdha
TB1 —> ti
TA1 —> dha

dans lesquels les règles de type 2 (celles qui contiennent deux variables dans l'argument droit) sont des règles **structurelles**, et les règles de type 3 (celles du bas) sont des règles **lexicales**. Ce mode de représentation est rigoureusement équivalent à celui d'un accepteur fini, mais il met en évidence des agrégats de symboles pertinents au domaine, et que nous allons définir formellement.

Considérons la partition de A* induite par la relation d'équivalence $\mathcal{R}_m$ telle que:

$\mathcal{R}_m(x,y)$ <=>
$\forall u,v \in A^*,\ uxv \in L$ <=> $uyv \in L$.

On appelle **monoïde syntaxique** de L l'ensemble des classes de $A^*/\mathcal{R}_m$. Deux cas peuvent se présenter:

(1) Si *x* et *y* ne sont facteurs d'aucune chaîne de L, on a $uxv \notin L$ et $uyv \notin L$ pour tout *u*, *v*, et par conséquent $\mathcal{R}_m(x,y)$. Appelons C_0 la classe qui contient toutes ces chaînes n'apparaissant dans aucune chaîne de L.

(2) Considérons *x* et *y*, deux facteurs de chaînes de L vérifiant $\mathcal{R}_m(x,y)$ et appartenant à la classe C_j ($j \neq 0$) de $A^*/\mathcal{R}_m$. Appelons e_0 l'état initial de l'accepteur canonique $\mathcal{A}_{can}(L)$ et *f* sa fonction de transition. Posons $f(e_0,u) = \{e_1\}$, $f(e_0,ux) = \{e_2\}$, $f(e_0,uy) = \{e_3\}$.

L'accepteur canonique étant déterministe, les images de *f* sont des ensembles vides ou de cardinal 1. Nous pouvons imposer, pour satisfaire la condition du théorème VI.2, que L possède un échantillon caractéristique S_c. D'après le théorème VI.1, $\mathcal{A}_{can}(L)$ n'est autre que $\mathcal{A}_{pref}(S_i)/\mathcal{R}$ avec $S_i \supseteq S_c$, où $\mathcal{A}_{pref}(S_i)$ est l'accepteur préfixe arborescent de S_i et $\mathcal{R}$ la relation d'équivalence sur A* telle que: $\mathcal{R}(x,y)$ <=> Tail(L,x) = Tail(L,y) .

Par définition, la relation $\mathcal{R}_m(x,y)$ impose Tail(L,ux) = Tail(L,uy), et donc $\mathcal{R}(ux,uy)$. Par conséquent, e_2 et e_3 appartiennent à la même classe de $\mathcal{A}_{pref}(S_i)/\mathcal{R}$, ce sont deux états fusionnés de l'automate préfixe arborescent.

A chaque classe C_j ($j \neq 0$) de $A^*/\mathcal{R}_m$ on peut donc associer deux états: $e(j) = e_1$ et $e'(j) = e_2$ (pas nécessairement distincts) de $\mathcal{A}_{can}(L)$. Il est facile de montrer qu'à toute paire d'états de $\mathcal{A}_{can}(L)$ correspond au plus une seule classe C_j, par conséquent le monoïde

syntaxique d'un langage régulier est fini. Tous les états ainsi associés à des classes C_j sont des noeuds du graphe représentant $\mathcal{A}can(L)$. Chaque classe contient un nombre fini ou infini de chaînes de A*; appelons Xj l'ensemble de chaînes de la classe C_j :

$$Xj = \{x \in A^* \mid f(e(j),x) = \{e'(j)\}\}$$

Appelons X la réunion des Xj pour j = 1,…, n où *n* est le nombre de classes de $A^*/\mathcal{R}m$. Nous appelons **vocabulaire** de L la partie V de X telle que:

$$\forall x \in V,\ x = uv \Rightarrow u \notin V \text{ ou(et) } v \notin V .$$

Tout élément de V est appelé un **mot**. On peut montrer que V est un **code**[167] sur l'alphabet A et que L est un ensemble de **messages** dans V. Concrètement, V est représenté par l'ensemble des plus courts chemins (orientés) qui relient deux noeuds du graphe sans traverser un troisième noeud.

Si nous représentons $\mathcal{A}can(L)$ sous la forme d'une grammaire de type 2, comme ci-dessus, dans laquelle chaque noeud est désigné par une variable, le vocabulaire est l'ensemble des arguments droits des règles lexicales. Nous appelons **segmentation** de $x \in L$ la factorisation (unique) $x = u_1 \ldots u_{n(x)}$ avec $u_j \in V$ pour tout j.

Il est clair qu'une partie importante de l'acquisition de connaissances sur un langage fini ou infini L consiste à trouver la segmentation de L. Toutefois il n'est pas garanti que la segmentation, telle que nous venons de la définir, soit entièrement compatible avec celle qu'un informateur pourrait proposer. Nous étudions donc maintenant le cas où un informateur est chargé de valider toute décision du type: “x se segmente uv” pour x = uv. Appelons Seg(x,u,v) la décision. Evidemment, on a toujours Seg(x,λ,x), Seg(x,x,λ), et l'on peut supposer que les deux règles de transitivité sont vérifiées:

$$\text{Seg}(x,u,v) \text{ et } \text{Seg}(u,w,z) \Rightarrow \text{Seg}(x,w,zv)$$
$$\text{Seg}(x,u,v) \text{ et } \text{Seg}(v,w,z) \Rightarrow \text{Seg}(x,uw,z)$$

Nous appelons **accepteur étoile**[168] $\mathcal{A}e(S_i)$ l'accepteur fini non-déterministe $(A,E,\{e_0\},T,\{0,1\},\varphi)$ reconnaissant exactement S_i tel que:

$$\forall e \in E,\ \text{card}(H(e)) = 1 \text{, et } \text{card}(T(e)) = 1 \text{ si } e \neq e_0$$

Il est facile de prouver que l'accepteur préfixe arborescent $\mathcal{A}pref(S_i)$ est $\mathcal{A}e(S_i)/\mathcal{R}s1$, où $\mathcal{R}s1$ est la relation d'équivalence sur E telle que:

$$\forall e_1,e_2 \in E,\ \mathcal{R}s1(e_1,e_2) \Leftrightarrow H(e_1) = H(e_2)$$

Chaque état *e* de $\mathcal{A}pref(S_i)$ est étiqueté par H(e). L'accepteur canonique $\mathcal{A}can(L)$ est $\mathcal{A}pref(S_i)/\mathcal{R}$, avec $S_i \supseteq S_c$, où:

$\mathcal{R}(u_1,u_2) \Leftrightarrow \text{Tail}(L,u_1) = \text{Tail}(L,u_2)$
et u_1,u_2 sont les étiquettes de deux états e_1,e_2 de $\mathcal{A}pref(S_i)$.

On peut donc écrire:

$$\mathcal{R}(e_1,e_2) \Leftrightarrow \text{Tail}(L,H(e_1)) = \text{Tail}(L,H(e_2))$$

---

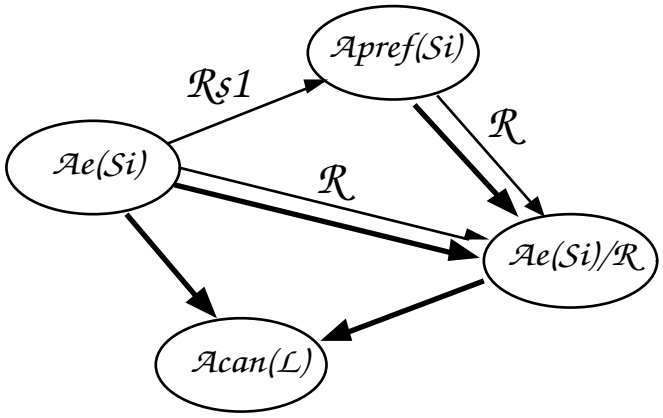
[167] Toute chaîne de A* possède au plus une factorisation dans V.

[168] Miclet (communication personnelle, 1990) utilise le terme “accepteur canonique maximal”.

Par ailleurs, il est évident que:

$$\forall\ e_1,e_2 \in E\ ,\ \mathcal{R}s1(e_1,e_2)\ \Rightarrow\ \mathcal{R}(e_1,e_2)$$

Par conséquent $\mathcal{A}pref(S_i)/\mathcal{R}$ est aussi $\mathcal{A}e(S_i)/\mathcal{R}$, où $\mathcal{R}$ est étendue à E. Pour S_i quelconque on peut représenter ainsi les accepteurs et les quotients:



où $\mathcal{A}e(S_i)/\mathcal{R}$ est isomorphe à un sous-accepteur de $\mathcal{A}can(L)$, d'après le théorème VI.1. On a indiqué en gras la relation: “le langage de l'accepteur X est inclus dans celui de l'accepteur Y”.

Appelons maintenant $\mathcal{R}'$ la relation d'équivalence sur les états de $\mathcal{A}e(S_i)$ telle que:

$$\forall\ e_1,e_2 \in E\ ,\ \mathcal{R}'(e_1,e_2)\ \Leftrightarrow$$
$$\mathcal{R}(e_1,e_2)\ \text{ et }\ Seg(H(e_1).T(e_1), H(e_1), T(e_1))\ \text{ et }\ Seg(H(e_2).T(e_2), H(e_2), T(e_2))\ .$$

Selon $\mathcal{R}'$, on ne peut fusionner deux états que si $\mathcal{R}$ est vérifiée et que chacun d'eux induit une segmentation acceptable sur la chaîne (unique) qui le traverse. Appelons $\mathcal{A}seg(S_i)$ l'accepteur quotient (en général non déterministe) $\mathcal{A}e(S_i)/\mathcal{R}'$. On vérifie facilement que, puisque la partition induite par $\mathcal{R}'$ affine celle induite par $\mathcal{R}$, , le langage $Lseg_i$ reconnu exactement par $\mathcal{A}seg(S_i)$ est inclus dans le langage $Lnseg_i$ reconnu exactement par $\mathcal{A}e(S_i)/\mathcal{R}$, par conséquent:

$$L \supseteq Lnseg_i \supseteq Lseg_i \supseteq S_i$$

et dans le cas où $S_i \supseteq S_c$,

$$L = Lnseg_i \supseteq Lseg_i \supseteq S_i$$

8. Construction sous contraintes de $\mathcal{A}_i$ (*constructing $\mathcal{A}_i$ under constraints*)

Soit $\mathcal{A}_i$ n'importe quel accepteur fini (non déterministe) à un seul état initial qui reconnaît exactement S_i . On peut toujours concevoir $\mathcal{A}_i$ comme le quotient $\mathcal{A}e(S_i)/\mathcal{R}s2$, où $\mathcal{R}s2$ est une relation d'équivalence convenablement choisie[169] sur les états de $\mathcal{A}e(S_i)$.

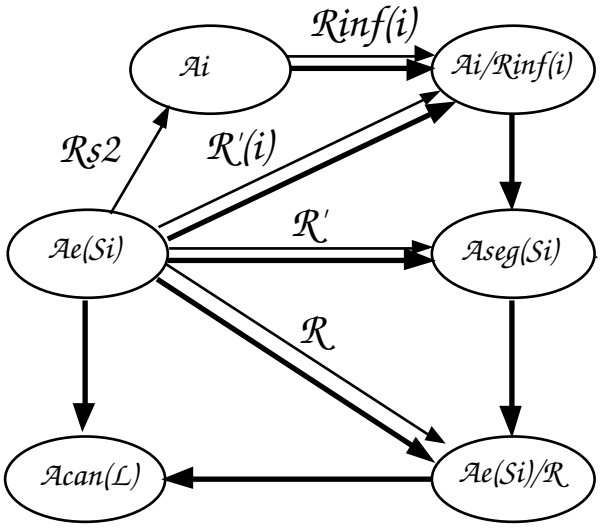
Faire une inférence inductive correcte sur $\mathcal{A}_i$ revient à trouver une relation d'équivalence $\mathcal{R}inf(i)$ telle que le langage exactement reconnu par $\mathcal{A}_i/\mathcal{R}inf(i)$ soit inclus dans L et que la segmentation induite par $\mathcal{A}_i/\mathcal{R}inf(i)$ soit acceptable. Si nous appelons $\mathcal{R}'(i)$ la relation d'équivalence sur E (l'ensemble des états de $\mathcal{A}e(S_i)$), telle que $\mathcal{A}_i/\mathcal{R}inf(i) = \mathcal{A}e(S_i)/\mathcal{R}'(i)$, il suffit d'avoir

$$\forall\ e_1,e_2 \in E\ ,\ \mathcal{R}'(i)(e_1,e_2) \Rightarrow \mathcal{R}'(e_1,e_2)$$

pour que le langage reconnu par $\mathcal{A}_i/\mathcal{R}inf(i)$ soit inclus dans $Lseg_i$, donc a fortiori dans L, et que la segmentation soit acceptable. Comme on a par ailleurs $\mathcal{R}s2(e_1,e_2) \Rightarrow \mathcal{R}'(i)(e_1,e_2)$, on doit choisir $\mathcal{R}s2$ puis $\mathcal{R}inf(i)$ de sorte que:

$$\forall\ e_1,e_2 \in E\ ,\quad \mathcal{R}s2(e_1,e_2) \Rightarrow \mathcal{R}'(e_1,e_2) \quad \text{et} \quad \mathcal{R}inf(i)(e_1,e_2) \Rightarrow \mathcal{R}'(e_1,e_2)$$

La première condition est réalisée en contraignant la construction de $\mathcal{A}_i$, mais pour la seconde il faudra disposer de contrexemples puisque la relation $\mathcal{R}$ n'est pas connue. Les accepteurs ainsi construits, les quotients, et la relation d'inclusion de langages (en gras) peuvent être représentés ainsi:



La contrainte sur la construction de $\mathcal{A}_i$ se traduit par une condition supplémentaire dans la relation d'équivalence $\mathcal{R}0$ qui sert à simplifier l'automate $\mathcal{A}'$. Nous allons établir cette condition.

---

[169] Nous appelons $\mathcal{R}s1$ et $\mathcal{R}s2$ des relations de “simplification” de $\mathcal{A}e(L_i)$.

Lorsqu'on cherche à construire $\mathcal{R}0$ pour simplifier $\mathcal{A}'$, on énumère les couples d'états (e_1,e_2), où e_2 est un état de $\mathcal{A}_{i-1}$, et pour $e = e_1$ puis pour $e = e_2$ on cherche à vérifier:

(1) ou bien que *e* est déjà un noeud de $\mathcal{A}_{i-1}$;
(2) ou encore que *toute* chaîne *x* dont le chemin $E_{\mathcal{A}}(x)$ contient *e* se segmente uv tel que:

$e \in f(e_0,u)$ où e_0 est un état initial de $\mathcal{A}'$, et Seg(x,u,v) .

Appelons $P(\mathcal{A}',e)$ cette propriété. Il est clair que:

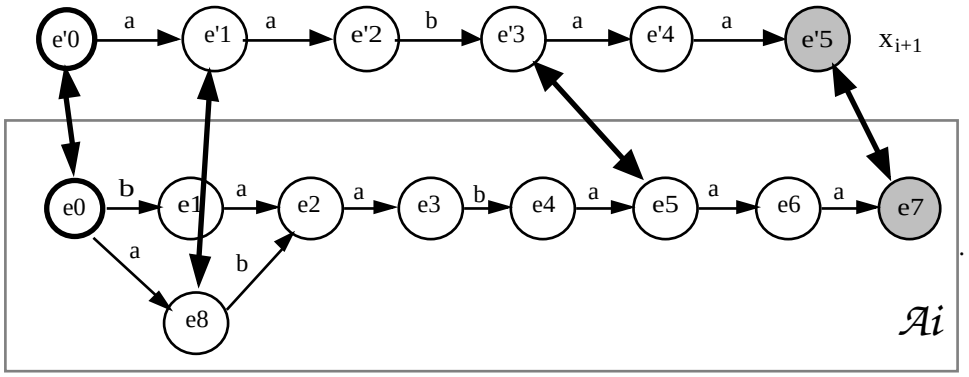
$$P(\mathcal{A}',e) \Rightarrow Seg(H(e).T(e), H(e), T(e)) .$$

Nous remplaçons donc, pour simplifier $\mathcal{A}'$, la relation $\mathcal{R}0$ par $\mathcal{R}'_0$ sur E, l'ensemble des états de $\mathcal{A}'$, telle que:

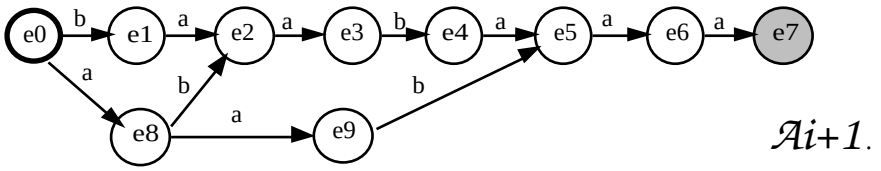
$\forall\ e_1,e_2 \in E$,
$\mathcal{R}'_0(e_1,e_2) \Leftrightarrow \mathcal{R}0(e_1,e_2)$ et $P(\mathcal{A}',e_1)$ et $P(\mathcal{A}',e_2)$.

Puisque $\mathcal{R}'_0(e_1,e_2) \Rightarrow \mathcal{R}0(e_1,e_2)$ on construit encore un accepteur qui reconnaît exactement $S_i \cup \{x_{i+1}\}$, mais avec une segmentation correcte. On peut montrer que la simplification de $\mathcal{A}'$ aboutit à un accepteur $\mathcal{A}_i = \mathcal{A}d(S_i)/\mathcal{R}s2$, où $\mathcal{R}s2$ respecte la condition: $\mathcal{R}s2(e_1,e_2) \Rightarrow \mathcal{R}(e_1,e_2)$.

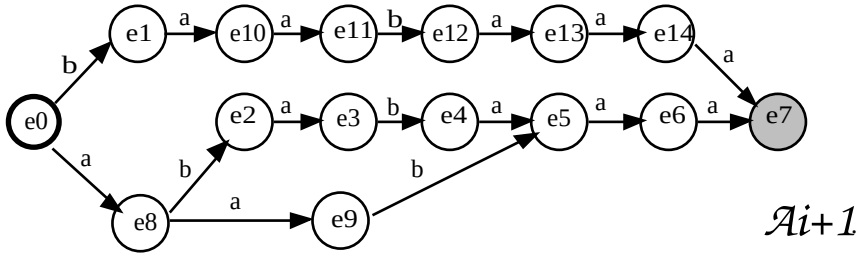
Dans la définition de la propriété P, nous avons souligné “toute” parce qu'il est souvent fastidieux de vérifier toutes les chaînes. Une approche réaliste consiste à vérifier $P(\mathcal{A}',e)$ pour un petit nombre de chaînes dont le chemin contient *e*, au risque de segmenter incorrectement d'autres chaînes. On peut accepter ce type d'erreur dans la mesure où elle n'introduit pas de mot nouveau incorrect. Elle peut du reste être corrigée lorsqu'on demande à l'accepteur de générer toutes ou une partie des chaînes qu'il reconnaît et qu'on fait apparaître leur segmentation. Supposons par exemple que S_i = {baabaaa,ababaaa} et que x_{i+1} = aabaa :



On a accepté $P(\mathcal{A}',e'_1)$ parce que l'informateur accepte de segmenter a/abaa, $P(\mathcal{A}',e'_3)$ … aab/aa, $P(\mathcal{A}',e_8)$ … a/babaaa, mais pour $P(\mathcal{A}',e_5)$ on a vérifié ababa/aa seulement. On obtient ainsi $\mathcal{A}_{i+1}$:



mais on s'aperçoit alors (ou plus tard) que baaba/aa est une segmentation inacceptable. Il n'est pas très difficile de corriger $\mathcal{A}_{i+1}$:



tout en gardant en mémoire l'information $P(\mathcal{A}_{i+1}, e_{10})$.

La construction sous contraintes de $\mathcal{A}_i$ introduit donc un dialogue entre la machine et l'informateur, dialogue qui porte sur des segmentations autorisées. Le système acquiert ainsi une connaissance de X. Il peut ensuite utiliser cette connaissance pour prendre seul une décision sur la propriété P: c'est pourquoi le système pose de moins en moins de questions à l'informateur.

Il est possible de réduire le questionnement à l'aide de contraintes supplémentaires spécifiques au domaine. Une contrainte que nous utilisons pour les langages rythmiques est qu'aucun mot n'est de longueur inférieure à 2, sauf peut-être le silence “-”.

Construire une table de vérité de $P(\mathcal{A}_i, e)$ revient à segmenter complètement toutes les chaînes de S_i. Cette opération n'est possible qu'à condition de connaître tout le vocabulaire. Si l'on ne connaît qu'une partie de l'ensemble X des séquences possibles de mots, on peut toutefois, pour n'importe quel $\mathcal{A}_i$, énumérer les états pour lesquels $P(\mathcal{A}_i, e)$ est plausible, et faire valider cette hypothèse par l'informateur. L'opération revient à examiner toute chaîne *w* de la partie connue de X, et si $w = uv$ avec $u \in X$ et $v \notin X$ on pose la question sur *v*. Inversement, si $u \notin X$ et $v \in X$, on pose la question sur *u*. A noter que l'informateur peut différer sa réponse: en pratique on construit une table de vérité de $P(\mathcal{A}_i, e)$ à trois valeurs: vrai, faux et inconnu.

A partir d'un certain moment l'utilisateur peut décider que le système connaît entièrement le vocabulaire V. La construction de $\mathcal{A}_i$ se poursuit alors sans dialogue puisque le système est jugé assez compétent pour calculer lui-même la propriété P.

Il existe d'autres contraintes que l'on peut imposer à la construction de $\mathcal{A}_i$, contraintes qui dépendent de propriétés du langage mais qui permettent par la suite de réaliser des inférences sans surgénéralisation. C'est le cas des langages k-réversibles[170]: tout

[170] Angluin 1982, p.750

échantillon d'un langage k-réversible L est reconnu exactement par un accepteur k-réversible, et d'autre part l'accepteur canonique de L est k-réversible. On peut donc limiter la recherche de $\mathcal{A}_i$ à l'ensemble des accepteurs k-réversibles. Pour cela, d'une part, $\mathcal{A}_i$ doit être déterministe, et d'autre part deux états e_1,e_2 ne peuvent vérifier $\mathcal{R}'0(e_1,e_2)$ que s'ils ont un k-leader commun.

Nous effectué la construction sous contraintes de l'accepteur reconnaissant exactement les dix exemples de variations rythmiques ci-dessus (en limitant la recherche aux mots de longueur supérieure à 1, sauf "-").[171] Nous donnons ci-dessous la segmentation obtenue et la partie de X correspondante:

1. trkt / dhati / dhadhatrkt / dhatidhagedheenagena
2. tidha / trkt / dhadhatrkt / dhatidhagedheenagena
3. dhatrkt / dhati / dhatrkt / dhatidhagedheenagena
4. dhatrkt / dhatrkt / dhage / dhatidhagedheenagena
5. dhatrkt / dhatrkt / dhadha / dhatidhagedheenagena
6. dhatrkt / dhati-dhati / dhatidhagedheenagena
7. ti- / dhati / dhadhatrkt / dhatidhagedheenagena
8. dhatidhatrkt / dhatrkt / dhatidhagedheenagena
9. trkt / trkt / dhadhatrkt / dhatidhagedheenagena
10. trktdhatrktdhagena / dhatidhagedheenagena

Mots reconnus:

trktdhatrktdhagena
tidha
dhatidhatrkt
ti-
dhati-dhati
dhagedheenagena
dhati
dhage
dhatrkt
dhadhatrkt
dhatidhagedheenagena
trkt
trktdhatidhagedheenagena
dhadha

Arrivé à ce stade on a cherché à établir la table de vérité de P($\mathcal{A}_i$,e) en posant des questions du type: "*couper* trktdhatrkt/dhagena?", ce qui a abouti à la segmentation:

1. trkt / dhati / dhadha / trkt / dhati / dhagedheenagena
2. tidha / trkt / dhadha / trkt / dhati / dhagedheenagena
3. dhatrkt / dhati / dhatrkt / dhati / dhagedheenagena
4. dhatrkt / dhatrkt / dhage / dhati / dhagedheenagena
5. dhatrkt / dhatrkt / dhadha / dhati / dhagedheenagena
6. dhatrkt / dhati-dhati / dhati / dhagedheenagena
7. ti- / dhati / dhadha / trkt / dhati / dhagedheenagena
8. dhatidhatrkt / dhatrkt / dhati / dhagedheenagena
9. trkt / trkt / dhadha / trkt / dhati / dhagedheenagena
10. trkt / dhatrkt / dhagena / dhati / dhagedheenagena

[171] Kippen & Bel 1989a, p.209

9. Généralisation de l'accepteur presque minimal (*generalizing the almost-minimal acceptor*)

Nous supposons maintenant connus l'accepteur presque minimal $\mathcal{A}_i$ (construit sous contraintes) qui reconnaît exactement S_i, et la propriété $P(\mathcal{A}_i,e)$. Soit E l'ensemble des états de $\mathcal{A}_i$. Nous choisissons $\mathcal{R}inf(i)$ telle que:

$$\forall\ e_1,e_2 \in E\ ,\ \mathcal{R}inf(i)(e_1,e_2) \Rightarrow P(\mathcal{A}_i,e_1)\ \text{ et } P(\mathcal{A}_i,e_2)\ ,$$

afin de restreindre l'espace de recherche des couples d'états qui satisfont $\mathcal{R}inf(i)$ aux couples d'états correspondant à des noeuds sur le graphe, ou marqués comme vérifiant P. Il est facile de montrer que dans ce cas:

$$\forall\ e_1,e_2 \in E\ ,\ \mathcal{R}inf(i)(e_1,e_2) \Rightarrow \text{Seg}(H(e_1).T(e_1), H(e_1), T(e_1))\ \text{ et }\ \text{Seg}(H(e_2).T(e_2), H(e_2), T(e_2))\ .$$

Si de plus on a $\mathcal{R}(e_1,e_2)$ alors on aura bien réalisé la condition $\mathcal{R}inf(i)(e_1,e_2) \Rightarrow \mathcal{R}(e_1,e_2)$. Pour vérifier $\mathcal{R}(e_1,e_2)$ il suffit de vérifier que l'ensemble des chaînes nouvelles reconnues par l'accepteur quotient $\mathcal{A}_i/\mathcal{R}inf(i)$ est inclus dans L. Le système génère donc cet ensemble et le soumet à l'informateur pour validation (**oracle**).

Une première coupure de l'espace de recherche de $\mathcal{R}inf(i)$ peut être imposée par des propriétés spécifiques du domaine, par exemple:

(1) Langage fini: on ne peut avoir $\mathcal{R}inf(i)(e_1,e_2)$ s'il existe $u \in A^*$ tel que $f(e_1,u) \supseteq \{e_2\}$, où *f* est la fonction de transition de $\mathcal{A}_i$. Dans le cas contraire, en effet, si [e] est la classe d'équivalence contenant e_1 et e_2 , on aurait $f'([e],u) \supseteq \{[e]\}$ où f' est la fonction de transition de $\mathcal{A}_i/\mathcal{R}inf(i)$, et l'accepteur reconnaîtrait des mots contenant un facteur u^k pour tout entier *k*.

(2) Dans le cas des langages rythmiques, les chaînes ont des longueurs identiques, ou plus généralement des longueurs prenant à un très petit nombre de valeurs connues. Dans ce cas,

$$\mathcal{R}inf(i)(e_1,e_2) \Rightarrow \forall u_1 \in H(e_1)\ ,\ \forall u_2 \in H(e_2)\ ,\ |u_1| = |u_2|.$$

La propriété (2) implique la propriété (1), et dans ce cas on a avantage à affecter, dès sa création, à chaque état e un entier qui représente |u| pour tout $u \in H(e)$.

Il peut exister une propriété $\mathcal{R}spec$ telle que: $\mathcal{R}spec(e_1,e_2) \Rightarrow \text{Tail}(L,H()) = \text{Tail}(L,H())$ et donc $\mathcal{R}(e_1,e_2)$. C'est le cas par exemple d'une propriété caractéristique des langages k-réversibles[172]:

$$\forall u_1,u_2,v,w \in A^* \text{ tels que } u_1vw \in L,\ u_2vw \in L \text{ et } |v| = k\ ,\ \text{Tail}(L,u_1v) = \text{Tail}(L,u_2v)$$

Par conséquent, si deux états e_1,e_2 d'un accepteur k-réversible sont tels que $T(e_1) \cap T(e_2) \neq \emptyset$ et qu'ils ont en commun un k-leader *v*, alors $\mathcal{R}(e_1,e_2)$ et dans ce cas $\mathcal{R}inf(i)(e_1,e_2)$ ne produit aucune surgénéralisation. La classe des langages k-réversibles

[172] Angluin 1982, p.750

est pour cette raison une classe de langages infinis identifiable à la limite à partir d'exemples seuls.

La condition de k-réversibilité entraîne d'autre part que l'accepteur quotient $\mathcal{A}_t/\mathcal{R}_{inf}(i)$ n'est pas ambigu. Dans le cas général, si l'on veut construire un accepteur non ambigu il suffit d'appliquer la règle:

> $\forall\ e_1,e_2 \in E$, s'il existe $u \in H(e_1)$ et $v \in T(e_2)$ tels que uv est reconnu par $\mathcal{A}_p$ alors on ne peut pas avoir $\mathcal{R}_{inf}(i)(e_1,e_2)$.

10. Heuristiques de généralisation (*generalisation heuristics*)

Des heuristiques générales permettent d'ordonner partiellement l'espace de recherche de $\mathcal{R}_{inf}(i)$. Supposons que pour deux états distincts e_1,e_2 on ait $P(\mathcal{A}_p e_1)$ et $P(\mathcal{A}_p e_2)$ et que par ailleurs ces états satisfassent les conditions spécifiques du domaine (langage fini, k-réversible, etc.). La relation $\mathcal{R}_{inf}(i)(e_1,e_2)$ est plausible dans les cas suivants, par ordre de plausibilité décroissante:

> (1) $T(e_1) \supseteq T(e_2)$ ou $H(e_1) \supseteq H(e_2)$;
> (2) $T(e_1) \cap T(e_2) \neq \emptyset$ ou $H(e_1) \cap H(e_2) \neq \emptyset$;
> (3) e_1 et e_2 possèdent un k-leader (ou un k-suiveur) commun, pour des valeurs décroissantes de *k*.

On peut ajouter ici des heuristiques dépendantes du domaine, par exemple celles basées sur des relations de ressemblance entre chaînes.

11. Extensions envisagées (*directions for future work*)

Une originalité de la méthode proposée est d'alterner les phases d'acquisition non inductive de connaissances (à partir d'exemples seuls) avec celles où le système peut généraliser ces connaissances (en produisant, le cas échéant, des contrexemples).[173]

La méthode traite de manière systématique le problème de la segmentation, en permettant, sous le contrôle d'un expert, la validation du lexique inféré. Ce contrôle devient de moins en moins nécessaire au fur et à mesure que le système acquiert des connaissances. Deux questions restent toutefois à résoudre en ce qui concerne la méthodologie et le modèle théorique:

1) Les experts sont-ils toujours en mesure de répondre aux questions posées par le système? Sous quelle forme devrait-on présenter les exemples à segmenter de manière à faire apparaître un contexte suffisamment précis pour que la décision soit fiable? Dans l'exemple cité, les décisions sont relativement faciles parce que la segmentation est liée à l'accentuation.
2) Comment pourrait-on traiter des réponses ambiguës ou incohérentes sur la segmentation?

---

[173] Cette approche est une modélisation idéalisée de la méthode d'enseignement dans le domaine étudié.

Les travaux actuels portent sur la construction de fonctions discriminantes d'un monoïde A* partitionné en (p+1) classes avec p > 1 (voir §2): il s'agit de construire, non plus un accepteur, mais un automate fini dont la fonction de sortie retourne le numéro de la classe.[174] Le numéro de classe peut désigner celui d'une grammaire — dans ce cas on traite simultanément plusieurs *qa'ida*s. Il peut aussi désigner un ensemble de valeurs de vraisemblance pour les exemples, depuis 0 ("incorrecte") jusqu'à la valeur maxi ("très correcte").

On s'intéresse enfin à l'hypothèse de k-réversibilité, toujours vérifiée pour un langage fini (avec une valeur suffisante de *k*). Le fait qu'on ait affaire à des langages finis permet d'adapter l'algorithme d'Angluin[175] en éliminant le *backtracking* lorsque la valeur courante de *k* est contredite par un contrexemple, autrement dit, de construire simultanément l'ensemble des généralisations du langage pour des valeurs décroissantes de *k*.

[174] C'est dans ce but que nous avons jugé utile d'introduire le formalisme de la fonction de sortie, au lieu d'associer à tout automate l'ensemble de ses états acceptables.

[175] 1982

VII. Temps symbolique, objets temporels/atemporels, synchronisation (*symbolic time, time/out-time objects, synchronization*)[176]

Pour Balaban[177], la musique est avant tout une structure hiérarchique (éventuellement sous plusieurs aspects) organisée sur une échelle du temps. Balaban examine de manière critique différents langages de description de la musique en distinguant les représentations déclaratives[178], qui décrivent la musique comme une structure de données (en vue de la formalisation de partitions), et les représentations procédurales[179], où l'aspect temporel est partiellement défini par les structures de données et par l'exécution du programme. Dans toutes ces approches les structures de données atomiques (hauteur, durée, registre, dynamique, etc.) sont signifiantes, mais les structures de plus haut niveau sont artificielles et manquent de flexibilité:[180]

> In SCORE the music is necessarily decomposed into 'instruments' which are collections of arrays describing, each, a line of notes and their properties. PLA decomposes the music into 'voices' which are similar to SCORE's instruments, and into 'sections' which are collections of voices. Both languages enforce a rigid polyphonic view of all music, and use artificial formal constructs — collections of arrays of musical properties — to describe the music. Gourlay's language is designed for music printing, and therefore all higher level constructs correspond to partitioning of a printed score by measures.[…] The language is not appropriate for standardizing music description since the notion of hierarchy in music does not, necessarily, coincide with the measures partitioning, but rather with musical objects like parts, sections, phrases, motifs, themes, etc.[181]

On s'intéresse, dans les chapitres VII-VIII-IX, aux outils conceptuels permettant de formaliser les relations entre la description symbolique de la musique (les structures de surface du niveau syntaxique) et les caractéristiques sonologiques des "objets musicaux".

1. Sonèmes, objets musicaux, objets sonores (*sonemes, musical objects, sound-objects*)

En faisant un parallèle avec la linguistique computationnelle, on peut s'interroger sur la possibilité d'introduire une **composante sonologique** dans une grammaire formelle décrivant un idiome musical.

---

176 L'essentiel de ce chapitre a été réédité en version anglaise (Bel 1991a) et donnera lieu à publication (Bel 1991c).

177 1990

178 Erickson 1977; Byrd 1977; Smith 1973; Droman 1983; Maxwell & Ornstein 1983; Hamel 1984; Yavelow 1986; Gourlay 1986. (Tous cités par Balaban)

179 Smoliar 1971; Smith 1976; Schottstaedt 1983; Cointe et Rodet 1983. (Tous cités par Balaban)

180 Balaban 1990.

181 *Ibid.*

Honing[182] affirme qu'il existe un large consensus sur l'utilisation d'éléments **discrets** (notes, événements sonores ou objets) comme primitives des représentations de la musique. Dans les approches inspirées de la linguistique et orientées vers la modélisation de partitions, on considère par exemple les notes comme des "phonèmes" de la musique. En psychologie de la musique, on se contente d'admettre l'existence d'un ensemble d'"éléments acoustiques" de base.[183] Toutefois, l'expérimentation en psychoacoustique montre qu'il est très difficile de définir ces éléments d'un point de vue strictement perceptuel.[184]

1.1 Sonèmes et objets musicaux (*sonemes and musical objects*)

L'informatique permet de représenter le son aussi bien comme un phénomène **continu** que **discontinu**. Dans le premier cas, il peut s'agir d'une description de processus de synthèse (l'évolution d'un nombre fini de paramètres dans un espace de représentation adéquat) ou encore de modèles physiques dont l'évolution est jugée intéressante.[185] Dans le second cas, le compositeur se munit, soit de manière générale, soit pour une œuvre musicale particulière, d'un ensemble d'objets que l'on peut appeler — par analogie avec la phonétique — des **sonèmes**.[186]

> Un sonème est une classe d'événements acoustiques équivalents (ses **variantes acoustiques**).

Une taxonomie de sonèmes (les "objets musicaux") a été proposée par Schaeffer[187], fondateur de la musique concrète. On pourrait l'utiliser pour formaliser une caractérisation systématique d'objets à l'aide de "traits" représentés par des variables dichotomiques, par exemple *continu/discontinu*, etc.[188-189]

En limitant les manipulations des objets musicaux à des opérations de collage, la musique concrète ne répondait pas entièrement aux attentes de certains compositeurs, qui ont préféré rechercher une totale liberté de manipulation dans la synthèse numérique.[190] Une approche sonologique aurait plutôt consisté à classer les objets d'après leur **fonction** dans un système de communication:

---

[182] 1990

[183] Deutsch 1982. Cité par Honing 1990.

[184] Voir par exemple McAdams & Bregman 1985.

[185] Borin et al. 1990, p.234

[186] Il est difficile de mettre en évidence l'intentionalité du compositeur: le choix d'une représentation discrète plutôt que continue peut résulter aussi bien d'une axiologie compositionnelle, que de contraintes imposées par un système d'écriture, ou par un environnement de tâches — par exemple, l'utilisation de matériaux échantillonnés, d'un séquenceur, etc.

[187] 1966

[188] Ce travail a été entrepris par Marco Ligabue (CNUCE de Florence) en s'inspirant de la phonologie de Jakobson (1963). Voir Chiarucci 1972 pour une approche similaire.

[189] Il ne faut toutefois pas confondre la caractérisation des objets (par un ensemble fini de traits) avec l'ensemble des paramètres qui permettraient de les reconstituer de manière satisfaisante par une opération de synthèse. (Voir par exemple Risset 1989a, ou Kronland-Martinet & Grossmann 1991) La reconstitution sonore d'un instrument est d'ailleurs d'un intérêt discutable dans une situation de simulation: l'évaluation, par un expert humain, des propositions de la machine risque de porter plus sur la qualité de la reconstitution du matériau sonore que sur le contenu musical.

[190] Risset 1989b, p.67

> […] not only do individuals and groups give different verbal meanings to music; they also conceive its structures in ways that do not permit us to regard musical parameters as objective acoustical facts. In music, thirds, fourths, fifths, and even octaves, are social facts, whose syntactical behaviour can differ as much as that of *si, see*, and *sea, beau, bow*, and *bo*, or *buy, bye, by* and *bai.*[191]

Même dans un domaine aussi restreint que celui de la musique tonale, une telle classification doit faire appel à une théorie élaborée, comme par exemple celle de Lerdahl et Jackendoff[192] (voir chapitre IV §1).

A l'opposé de cette théorie,[193] une approche empiriste de la performance est celle de Katayose & Inokuchi[194], qui consiste à identifier et quantifier les paramètres caractéristiques d'une interprétation musicale “virtuose” dans une (très petite[195]) partie des informations sur le geste musical.

Dans tous les cas cités, les recherches s'insèrent dans un système musical déjà formalisé[196] pour lequel existe à la fois un large corpus d'œuvres et un consensus sur certaines règles d'interprétation.

1.2 Objets sonores (*sound-objects*)

Dans un environnement d'informatique musicale, élaborer la composante sonologique d'une grammaire musicale impose en premier lieu de définir une correspondance entre les sonèmes et les symboles terminaux de la grammaire. Ce qui tient lieu de règles d'articulation peut être formulé, comme dans la phonologie générative de Chomsky[197], à l'aide de règles de réécriture.[198] Il est préférable toutefois, dans un souci de généralité, de considérer toute variante acoustique d'un sonème, non comme un échantillon sonore figé, mais comme le résultat d'un processus déclenché et contrôlé par des **messages**, i.e. des vecteurs de paramètres positionnés sur l'axe du temps.

On peut ainsi parler d'**objet sonore** pour désigner une structure de données qui comprend la description du processus (que nous appelons “**prototype d'objet sonore**”) et un certain nombre de paramètres, propriétés et procédures qui permettent d'**instancier** les variantes acoustiques du sonème associé.[199] (Voir chapitre IX)

---

[191] Blacking 1984, p.364

[192] Jackendoff & Lerdahl 1982, pp.92-ff; Lerdahl & Jackendoff 1983.

[193] L'objet — contestable — des travaux de Lerdahl et Jackendoff est de définir un modèle de la **compétence** universelle dans le domaine de la tonalité, en référence au postulat d'innéisme de Chomsky. Voir à ce sujet la critique de Piaget (Piattelli-Palmarini 1979).

[194] 1990

[195] On se limite aux estimations de l'intensité d'attaque des notes et aux durées du maintien de la pression sur un clavier. Une étude plus détaillée du geste musical a été menée par l'équipe de l'ACROE à Grenoble. Des travaux similaires en vue d'établir des règles d'interprétation en jazz (différentes conceptions du *swing*) sont en cours au CNUCE de Florence.

[196] Pour Lerdahl et Jackendoff il ne s'agit que de monodie. Katayose & Inokuchi travaillent sur des pièces polyphoniques de musique classique.

[197] Chomsky & Halle 1973.

[198] On peut désigner comme **morphème** tout regroupement significatif (pas nécessairement séquentiel) “minimal” de sonèmes. Voir par exemple la notion de “vocabulaire” dans un langage de percussion, chapitre VI §7.

[199] Nous avons volontairement introduit ici la terminologie de la **représentation orientée objet** en informatique, en lieu et place de celle des “objets musicaux”.

Un objet sonore ne produit pas nécessairement un son: il produit des messages qui sont interprétés par un générateur de sons. Certains messages peuvent ne servir qu'à modifier les objets sonores suivants.

1.3 Contraintes soniques (*sonic constraints*)

Une représentation discrète est souvent associée à une stratégie de composition **descendante** ou **guidée par les buts** (*top-down, goal-driven*) qui consiste à "instancier" une structure hiérarchique prédéfinie ou calculée à l'aide de règles de construction (une "grammaire"). Dans les représentations continues, par contre, il semble que l'on privilègerait plutôt une stratégie **ascendante** ou **guidée par les données** (*bottom-up, data-driven*) dans laquelle les structures musicales "émergent" du continuum sonore lors de l'écoute.[200]

Le système sonologique décrit dans cette étude privilégie l'approche descendante au niveau de la structure syntaxique des propositions, et l'approche ascendante au niveau de l'instanciation sonore de ces propositions.

De la même manière que la composante sonologique d'une grammaire fait le lien entre les représentations syntaxique et acoustique, cette composante peut à son tour se scinder en deux niveaux: une **matrice sonologique** qui décrit des catégories d'objets musicaux et leurs relations avec des segments de la structure de surface, et une **matrice sonique** qui décrit les propriétés des objets indépendantes des sources sonores (propriétés **acousmatiques**[201]) et de leur fonction syntaxique (propriétés **asyntaxiques**).[202] Dans l'approche que nous préconisons, la matrice sonologique est implicitement décrite par les règles de réécriture, l'agencement final des sonèmes s'effectuant au niveau de la représentation sur le temps symbolique (voir infra et chapitre VIII). La matrice sonique, par contre, est l'ensemble des paramètres et procédures d'instanciation des objets sonores. L'instanciation est donc un problème de **résolution de contraintes**, les unes "descendant" de la structure et les autres "remontant" des objets (voir chapitre IX).

2. Le temps musical (*musical time*)

Alors qu'un énoncé linguistique est une séquence de phonèmes, une structure musicale discrète fait intervenir de multiples relations (implicites ou explicites) d'antériorité et de simultanéité entre les objets sonores. Il est donc indispensable, quelle que soit la composante sonologique d'une grammaire musicale, de disposer d'outils conceptuels permettant de formaliser ces relations temporelles.

Parmi les systèmes de représentation de la musique, on peut distinguer:[203]

— ceux qui ne contiennent **aucune représentation du temps**, par exemple les systèmes en temps réel;

---

[200] Il est légitime de s'intéresser en priorité, dans ce cas, aux phénomènes psychoacoustiques.

[201] Terme créé par Pythagore pour désigner une forme d'enseignement où le maître est caché de ses élèves, et repris par le compositeur Pierre Schaeffer pour désigner une écoute qui s'intéresse au contenu, et non à l'origine des sons. (Schaeffer 1966, p.91)

[202] Laske 1972a, p.30-3.

[203] Cette classification est inspirée de Honing (1990).

— ceux qui attribuent des **marques temporelles** aux objets sonores: points ou durées, sur un temps absolu, relatif, ou partiellement défini par des informations fournies en temps réel;[204]

— ceux qui contiennent un encodage explicite des **relations temporelles**: logiques temporelles[205], représentations orientées objet[206], types de données abstraits (*abstract data types*) en langage fonctionnel ou en logique des prédicats[207], etc.

Une limitation de nombreux modèles est l'absence de prise en compte de **propriétés** du temps musical et des objets qui pourraient conditionner leur positionnement temporel en fonction du contexte. Par conséquent, les systèmes (autres que ceux basés sur le temps réel) souffrent d'une rigidité qui les confine souvent à l'exécution de maquettes sonores. Par exemple, à toute séquence de symboles on fait correspondre une interprétation (unique) des objets sonores correspondants sans tuilage possible des intervalles supports temporels contigus. On reproduit ainsi, dans un modèle présumé général, des contraintes aussi limitatives que celles liées à la notation tonale et au temps métronomique.

Nous proposons dans le cadre de cette étude une approche nouvelle qui consiste à distinguer deux niveaux: celui de la **représentation symbolique** sur un temps gradué, et celui de la **représentation sonétique** sur le temps physique. La mise en temps des objets sonores fait intervenir à la fois la représentation symbolique (les **dates symboliques** des objets), une correspondance arbitraire entre temps symbolique et temps physique (**structure du temps**), des **contraintes** liées à des propriétés attachées aux objets, et, le cas échéant, des décisions du compositeur quant aux solutions envisagées. (Voir chapitre IX)

Les relations de simultanéité entre objets ne sont pas nécessairement explicitées dès le départ au niveau symbolique: la relation est explicite, non sur des *objets*, mais sur des *séquences* d'objets. Le système est en mesure de déduire les relations entre objets en faisant une hypothèse de moindre complexité lorsque les informations sont incomplètes. (Voir §8-9 et chapitre VIII)

Notre approche a été moins guidée par la recherche d'un formalisme méta-musical (transculturel) que par celle de solutions efficaces[208] à certains problèmes de traitement du temps communs à la notation rythmique instrumentale et à la musique électroacoustique. Nous présentons en premier lieu l'environnement informatique dans lequel s'est effectuée cette implémentation.

---

[204] Voir par exemple les méthodes de synchronisation de Dannenberg 1984; Vercoe 1984.

[205] Balaban & Neil 1989; Balaban 1990.

[206] Diener 1988; Camurri & al. 1990.

[207] Smaill & Wiggins 1990.

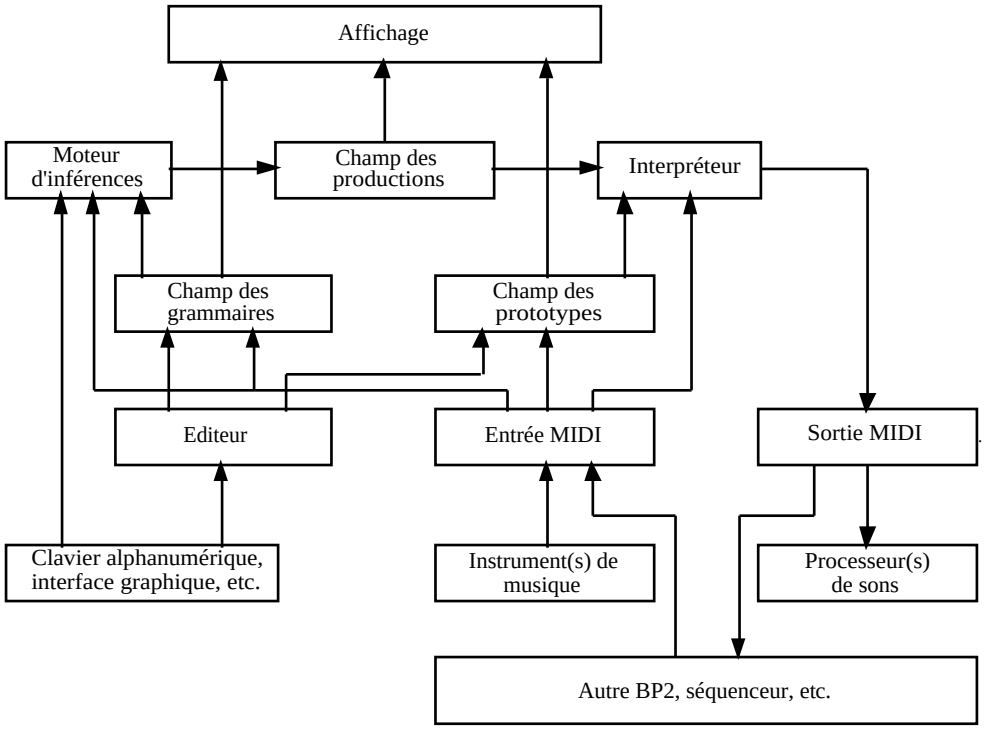
[208] L'efficacité consiste ici à rechercher des algorithmes de faible complexité qui peuvent être implémentés avec des langages de programmation de bas niveau.

3. *Bol Processor* BP2 — l'environnement (*BP2 environment*)

Dans cette version du *Bol Processor*, l'accent a été mis sur la réduction de la durée du **cycle de vie compositionnel** (*compositional life cycle*[209]), en permettant l'écoute directe d'ébauches musicales. Pour cela, le système a été relié à un processeur de sons en temps réel.[210]

Les travaux se sont orientés vers la recherche de procédures permettant divers modes de contrôle des réalisations, depuis le mode “automatique” qui laisse à la machine l'initiative des décisions, jusqu'au mode “pas-à-pas” qui permet l'exploration manuelle d'un ensemble de solutions.

Un schéma synoptique du *Bol Processor* BP2 est le suivant:



Trois champs sont utilisés pour mettre en mémoire des **grammaires**, des **productions** grammaticales et des **prototypes d'objets sonores** définis à l'aide d'un instrument de musique ou d'un clavier alphanumérique. Les productions sont des représentations symboliques (chaînes de caractères ou graphèmes) de structures musicales. Elles sont interprétées puis transmises au(x) processeur(s) de sons sous forme de messages en temps réel.

L'interprétation d'une production tient compte des propriétés des objets (au niveau sonique: continuité, déformabilité, etc., voir §1.3), qui ne sont pas explicitées par la grammaire. La résolution de ces contraintes permet de transformer une structure

---

[209] Laske 1989a, p.47

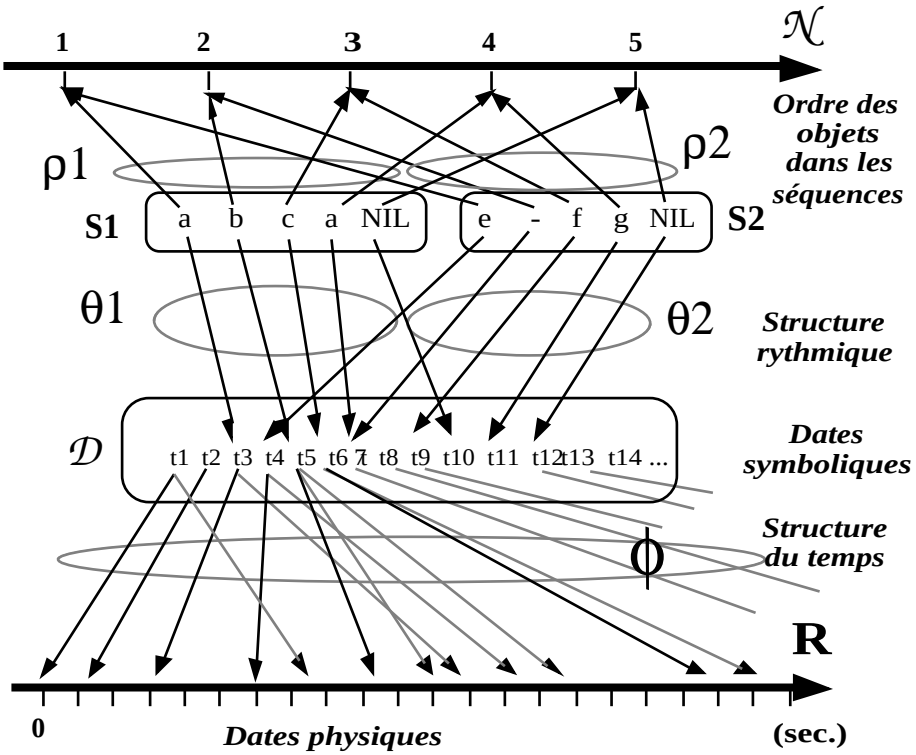
[210] Le système SYTER (SYnthèse en TEmps Réel), conçu par J.F. Allouis à l'INA-GRM. On peut utiliser n'importe quel processeur muni d'une interface MIDI.

temporelle “symbolique” (construite sur un ensemble arbitraire totalement et strictement ordonné) en une liste de dates et de paramètres d'exécution des objets. (Voir chapitre IX) L'algorithme de résolution fournit un ensemble fini (parfois très grand) de solutions que le compositeur est libre d'explorer lorsqu'il ne souhaite pas se contenter de la première solution.

Le schéma du BP2 indique aussi des possibilités de contrôle extérieur, à la fois du moteur d'inférence, des grammaires et du module d'interprétation. Ces contrôles sont essentiellement destinés à la composition improvisationnelle. Ils sont provoqués, soit par des interventions manuelles (clavier de contrôle, contrôleurs MIDI), ou encore par des messages en provenance d'une autre machine dialoguant avec le BP2.[211]

4. Temps symbolique et objets temporels (*symbolic time and time-objects*)

La notion de **temps symbolique** se rapproche de celle de *basic time* de Jaffe[212], ou encore du *virtual time* dans l'environnement de programmation musicale *Formula*[213]. Nous introduisons la terminologie à partir d'un exemple utilisant la notion intuitive de “séquence”, terme que nous définirons ultérieurement (chapitre VIII §1). La figure ci-dessous représente une structure formée de deux séquences notées symboliquement “abca” et “e-fg”, où “a”, “b”, “c”, “e”, “f”, “g” et “-” sont des **étiquettes d'objets sonores:**



211 On peut ainsi mettre en interaction plusieurs grammaires/automates pour faire de la “composition distribuée” en temps réel.

212 1985

213 Anderson et Kuivila 1989, pp.11-23.

L'application ρ (**indexage**) sert à définir la relation d'ordre strict des objets dans chaque séquence. Nous considérons par ailleurs $\mathcal{D}$ = {t1,t2,…}, un ensemble totalement et strictement ordonné de **dates symboliques**, et θ une application *injective* de S dans $\mathcal{D}$ que nous appelons la **structure rythmique** de la séquence. Nous appelons **objet temporel** le couple (x,θ(x)), où "x" représente l'étiquette d'un objet sonore et θ(x) sa date symbolique. Par convention, $\theta.\rho^{-1}$ est une fonction monotone croissante.

L'étiquette "-" désigne ici un silence, objet qui ne bénéficie d'aucun statut particulier. Les silences doivent être distingués des **objets vides** que nous étiquetons "_".[214] Un objet vide est la prolongation de l'objet, vide ou non, qui le précède.

Les deux séquences de l'exemple peuvent être représentées simultanément dans un tableau dont les lignes correspondent aux séquences et les colonnes aux dates symboliques:

1	2	3	4	5	6	7	8	9	10	11	12	13	14
_	_	a	_	b	c	a	_	_	NIL	_	_	_	_
_	_	_	e	_	-	_	_	f	_	g	NIL	_	_

Toute case du tableau qui ne contient pas une étiquette d'objet non vide contient le symbole "_".

Nous appelons **structure du temps**[215] l'application Φ qui associe à toute date symbolique une date physique. L'ensemble image de Φ est une partie de R, où les dates sont représentées avec une unité de temps convenable. Φ n'est pas nécessairement une fonction monotone croissante,[216] mais nous envisageons uniquement ce cas. Lorsque la monotonie est vérifiée, soit

$$t_i < t_j \iff \Phi(t_i) < \Phi(t_j)$$

on peut construire une distance Δ sur $\mathcal{D}$:

$$\Delta(t_i,t_j) = | \Phi(t_j) - \Phi(t_i) |$$

Puisque

$$\forall i,j,k,\ \Delta(t_i,t_j) + \Delta(t_j,t_k) \geq \Delta(t_i,t_k)\ ,$$

$(\mathcal{D},\Delta)$ est un espace métrique. Cet espace est, de plus, euclidien si la propriété

---

214 Dans la notation tonale numérique chinoise, les silences sont notés avec le chiffre "0" et les objets vides avec le symbole "-". Dans la notation indienne tonale de V.N. Bhatkhande, on utilise "S" pour le silence et "-" pour l'objet vide. Enfin, dans celle des percussions indiennes, on confond silence et objet vide.

215 Xenakis distingue les structures hors-temps, la structure du temps et la structure temporelle (1963, pp.190-1,200). La notion de structure du temps que nous introduisons ici correspondrait à celle de structure temporelle.

216 Jaffe (1985, p.40) ne s'intéresse qu'aux fonctions strictement monotones. Par contre il règle le problème de la polyphonie en utilisant une fonction Φ distincte pour chaque voix (*op.cit.* pp.44-ff).

$$j - i = l - k \Rightarrow \Phi(t_j) - \Phi(t_i) = \Phi(t_l) - \Phi(t_k)$$

est vérifiée. Cette situation correspond au **temps métronomique**.

5. Durée symbolique d'un objet temporel (*symbolic duration of a time-object*)

Dans le tableau précédent, on pourrait appeler durée symbolique d'un objet dont l'étiquette figure dans la colonne *i* la valeur de (j-i), où *j* est l'index de la colonne de l'objet non vide suivant sur la même ligne. Dans ce but on a placé un objet non vide “NIL” en fin de chaque séquence. Avec cette convention, les séquences d'objets temporels peuvent être représentées par des listes de couples (étiquette,durée symbolique):

(a,2) (b,1) (c,1) (a,3) (e,2) (-,3) (f,2) (g,1)

ce qui peut s'écrire plus simplement:

a _ b c a _ _ e _ -_ _ f _ g

Cette définition de la durée symbolique correspond au cas où l'ensemble des dates symboliques $\mathcal{D}$ est une partie de l'ensemble des entiers naturels N. Une autre option, que nous avons discutée en détail par ailleurs[217], consiste à prendre pour $\mathcal{D}$ une partie de l'ensemble des nombres rationnels Q. Dans ce cas, en effet, on peut insérer une infinité de dates intermédiaires entre deux dates distinctes données.

Durée symbolique: définition

> La **durée symbolique** du *n*-ième objet temporel[218] s_n dans une séquence S est la différence $\theta(s_{n+p})-\theta(s_n)$, où s_{n+p} est le prochain objet non vide consécutif à s_n dans la séquence S.

Remarque: on a $p > 1$ si s_{n+1} est un objet vide.

6. Objets atemporels (*out-time objects*)

Le formalisme introduit précédemment ne permet de manipuler que des objets de durée symbolique strictement positive. Cette approche n'est pas satisfaisante pour un grand nombre d'applications musicales. Il est difficile, par exemple dans le cas de la notation classique tonale, de caractériser la durée d'un objet aussi “fugitif” qu'une appogiature brève, même au prix d'une quantification très fine des durées.

En fait, les structures musicales qui manipulent de tels objets peuvent être vues comme intermédiaires entre l'homophonie et la polyphonie. Les modes d'exécution des ornements (appogiature, *grupetto*, mordant, etc.) sont dans ce cas fortement liés au

---

[217] Bel 1990e, pp.118-21.

[218] On utilise une définition différente pour la durée symbolique des objets sonores (voir chapitre IX §2.1).

traitement de la "matière" sonore. Dans une représentation informatique destinée à l'exécution mécanique, deux approches sont possibles:

1) On rattache l'ornement à l'objet sonore qu'il modifie (en général, l'objet suivant dans la séquence), au risque de multiplier les définitions d'objets;

2) On utilise une représentation symbolique particulière pour certains ornements.

C'est l'approche (2) qui justifie la création d'un nouveau type d'objets, les **objets atemporels**. Un objet atemporel a une durée symbolique nulle. Si "c" désigne un objet temporel et <<a>>, <<b>>, … des objets atemporels, la chaîne

<<a>> c

représente une structure sonore dans laquelle <<a>> est déclenché en même temps que "c". On peut écrire:

<<a>> <<b>> = <<b>> <<a>>

dans la mesure où les dates symboliques de ces deux objets atemporels sont identiques.

Un objet atemporel peut représenter un objet sonore (i.e. une séquence de messages qui produisent des sons), mais dans ce cas les messages sont envoyés simultanément (voir chapitre IX §1). La durée perçue de l'objet sonore représenté par un objet atemporel n'est toutefois pas nécessairement nulle; on peut imaginer par exemple un son percussif (cordophone *pizzicato*, membranophone, etc.).

Une autre application typique des objets atemporels est l'envoi de messages "non musicaux" pour la communication de paramètres d'exécution ou de messages de synchronisation entre machines.

Un objet atemporel est représenté par un couple (<<x>>, $\theta(x)$) dans lequel "x" est l'étiquette d'un objet sonore et $\theta(x)$ sa date symbolique.

7. Temps lisse et temps strié (*smooth and streaked time*)

Les notions de temps **lisse** et de temps **strié** (*smooth/streaked time*) ont été introduites par Boulez[219] pour caractériser deux situations typiques de l'interprétation musicale. Ces notions peuvent être ramenées à des propriétés de la structure Φ du temps.

Si l'application Φ est connue à l'avance, les interprètes (humains ou mécaniques) doivent ajuster les durées des objets afin de satisfaire les contraintes de "voisinage" (voir infra). Cette situation est celle du **temps strié**: "la pulsation du temps *strié* sera régulière ou irrégulière, mais systématique".[220] Par pulsation "régulière" on désigne ici une structure du temps qui se rapproche, au moins localement, de celle du temps métronomique.

Au temps strié Boulez oppose le **temps lisse**, que l'on peut associer à une structure du temps déterminée au moment de l'exécution.

---

[219] 1963

[220] Boulez 1963, p.104. Le qualificatif "systématique" pourrait s'appliquer, dans le cas d'une musique interprétée par une machine, à toute structure perçue comme "non arbitraire". Une telle définition, aussi simple qu'elle puisse paraître, met l'accent sur l'importance des mécanismes de perception.

Si la distinction entre temps lisse et temps strié, pour une musique interprétée par des humains ou (et) sous la direction d'un humain, se situe au niveau perceptif, c'est au niveau des algorithmes de mise en temps qu'il faut l'envisager dans le cas de musiques interprétées par des machines. Une structure de temps **strié** est une liste croissante de dates physiques sur lesquelles les objets doivent être positionnés: à chaque objet correspond une **strie de référence**. Une structure du temps **lisse** peut être définie comme une liste de dates indéterminées, chaque objet étant alors considéré comme "relocalisable".

8. Problème de la synchronisation (*the synchronization problem*)

Nous posons ici de manière abstraite le problème de la synchronisation d'événements. Au §9 sont présentées les conditions qui définissent une classe d'univers d'événements formés de séquences, ainsi qu'un ensemble de conventions qui permettent de résoudre le problème dans ce cas. L'application de cette représentation aux structures d'objets temporels, et l'algorithme de résolution des contraintes correspondant, font l'objet du chapitre VIII.

8.1 Univers d'événements (*event universe*)

Nous appelons **univers d'événements** un ensemble E structuré par trois[221] relations:

1) une relation d'ordre strict appelée "antériorité" et notée '<';

2) une relation d'équivalence appelée "simultanéité" et notée '=';

3) une relation d'ordre strict appelée "séquentialité" et notée '&';

et remplissant les **conditions de cohérence** suivantes:

$\forall (e_1,e_2,e_3) \in E^3$,

$e_1 \,\&\, e_2 \Rightarrow e_1 < e_2$;

$e_1 \,\&\, e_2$ et $e_2 = e_3 \Rightarrow e_1 \,\&\, e_3$;

$e_1 < e_2 \Rightarrow$ on n'a pas $e_1 = e_2$;

$e_1 < e_2$ et $e_2 = e_3 \Rightarrow e_1 < e_3$.

Remarque: on peut prouver de même: $e_1 < e_2$ et $e_1 = e_3 \Rightarrow e_3 < e_2$.

"a < b" se lit "a précède b". "a & b" se lit "a puis b".

On considère le graphe bicoloré non orienté G tel que:

$$e_1 < e_2 \quad \text{ou} \quad e_1 = e_2 \Rightarrow G(e_1,e_2)$$

[221] On peut aussi n'introduire que deux relations, celle d'antériorité étant un ordre réflexif et antisymétrique. La simultanéité de deux événements est alors leur antériorité réciproque.

8.2 Problème de la synchronisation (*the synchronization problem*)

En général, le graphe G n'est pas complet, ce qui revient à dire qu'il existe des couples d'événements qui ne sont pas comparables par '<' ni par '='.

Résoudre le **problème de la synchronisation** sur E revient à inférer assez de relations d'antériorité et de simultanéité pour que les fermetures transitives de '<' et de '=' produisent un graphe G complet. Nous appelons **univers complet** un univers d'événements tel que G est complet.

On remarque que les relations de séquentialité "&" ne servent ici qu'à déduire des relations d'antériorité. La séquentialité n'est qu'une donnée supplémentaire qu'on pourra utiliser lorsque les événements sont des intervalles temporels.

8.3 Le temps dans un univers complet d'événements (*time in a complete event universe*)

Lorsque le graphe G est complet on peut définir un temps gradué sur E. On construit le quotient de E par la relation d'équivalence "=". Les classes obtenues sont strictement ordonnées par "<". La date de chaque classe est son rang dans cet ordre.

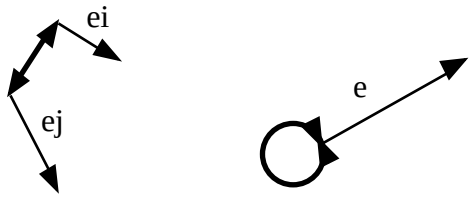
Inversement, étant donné un ensemble de points temporels sur un temps gradué, on peut toujours construire un graphe complet G à partir des relations "<" et "=" sur les dates de ces points.

La représentation d'un univers complet d'événements par un ensemble de points temporels sur un temps gradué est donc isomorphe de celle par les relations "=" et "<". Par contre elle ne permet pas de représenter la relation "&" ni de décrire un univers incomplet.

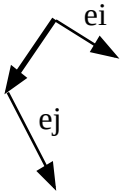
8.4 Représentation graphique (*graphic representation*)

1) Chaque événement est représenté par un arc orienté:

2) Les relations $e_i = e_j$ et e = e (réflexivité) sont représentées:

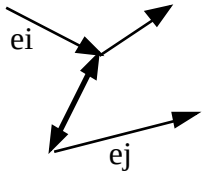


3) La relation $e_i < e_j$ est représentée:

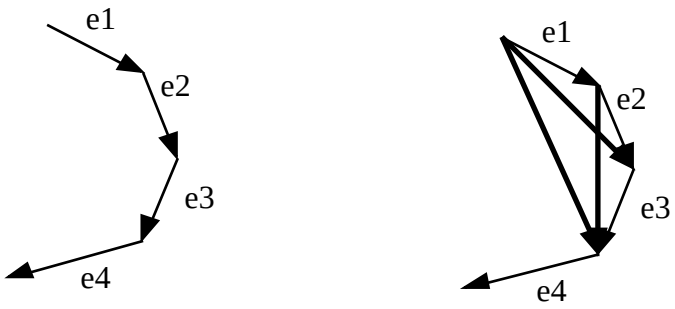


4) La relation e_i & e_j est représentée:

ou de manière équivalente:



5) Dans tout univers d'événements, on peut ne représenter qu'un sous ensemble des relations, choisi de sorte que le graphe G puisse être obtenu à partir des fermetures transitives de ces relations. Par exemple, une séquence est représentée par un graphe unilinéaire et permet toujours de déduire un graphe complet:



9. Synchronisation de séquences (*synchronizing sequences*)

On s'intéresse ici à une classe d'univers d'événements formés de séquences, pour laquelle existe une représentation parenthésée simple à partir de laquelle on peut résoudre de manière efficace le problème de la synchronisation.

On pose comme première hypothèse que E est un ensemble fini partitionné en *k* séquences:

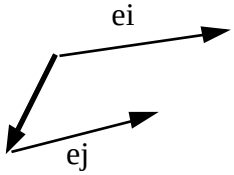
$$E = S_1 \cup \ldots \cup S_k$$

chaque séquence étant totalement ordonnée par '&'.

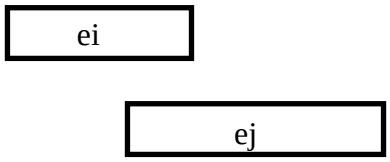
Remarque

Un univers complet dans lequel les événements sont des intervalles temporels (de durées connues) peut toujours être partitionné en séquences. On peut en effet écrire une relation de séquencement entre deux événements quelconques à condition de créer de nouveaux événements qui ne servent qu'à la synchronisation.

Soient par exemple deux événements non séquentiels:



dont les intervalles supports temporels sont dans la relation $\beta 4(e_i , e_j)$:

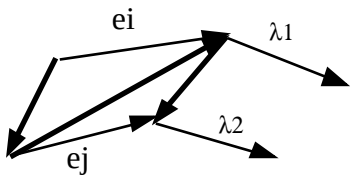


(Voir chapitre IX §5)

On crée deux événements $\lambda 1$ et $\lambda 2$ tels que:

$$e_i \ \& \ l1 \ , \quad e_j \ \& \ l2 \ , \quad e_j < l1 \ , \quad l1 < l2 \ ,$$

ce qui donne la représentation:



qui contient toute l'information sur le séquencement. Les événements e_i et e_j appartiennent maintenant à des séquences. ■

On note chaque séquence S_i comme une chaîne:

$$S_i = e_{i1} \ldots e_{ip} \quad \text{telle que} \quad e_{ij} < e_{ik} \iff j < k$$

On note '<<' la relation d'ordre strict sur $\{S_1, \ldots, S_k\}$ telle que:

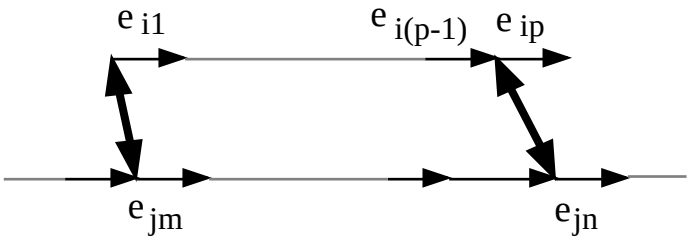
$S_i << S_j \iff$

$\forall\, e_{ip} \in S_i, \forall\, e_{jq} \in S_j,\ e_{ip} < e_{jq}$

On note par ailleurs '$\angle$' la relation d'ordre large sur $\{S_1, \ldots, S_k\}$ telle que:

$S_i \angle S_j$ avec $S_i = e_{i1} \ldots e_{ip}$ et $S_j = e_{j1} \ldots e_{jq} \iff$

$\exists\, m \in [1,q] \mid e_{i1} = e_{jm}$ et $\exists\, n \in [1,q] \mid e_{ip} = e_{jn}$



Remarque: l'événement e_{ip} en fin de séquence S_i ne sert qu'à la synchronisation de la séquence. Nous utiliserons plus tard une notation particulière pour cet événement.

9.1 Propriétés imposées aux univers incomplets d'événements (*properties imposed on an incomplete event universe*)

L'ensemble des séquences $\{S_1, \ldots, S_k\}$ est tel que:

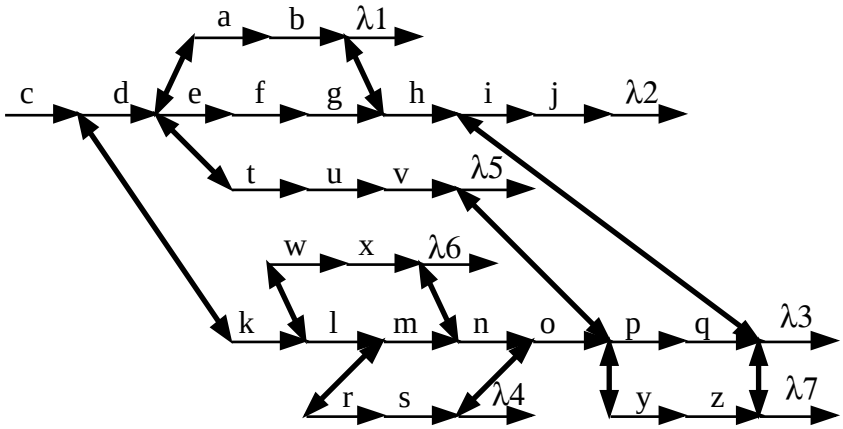
1) l'ordre '$\angle$' est total;

2) $S_i \angle S_l$ et $S_j \angle S_l$ =>

$S_i << S_j$ ou $S_j << S_i$

ou ($S_i \angle S_j$ et $S_j \angle S_i$) .

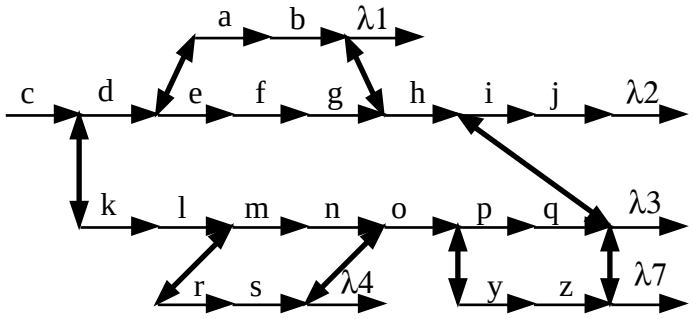
Ces propriétés sont faciles à illustrer graphiquement. Elles ne sont pas vérifiées dans l'univers suivant, bien qu'il soit partitionné en séquences:



En effet, l'ordre '$\angle$' n'est pas total puisqu'aucun couple ordonné par '$\angle$' ne contient la séquence "tuv…". De plus, on devrait avoir la relation '<<' entre "wx…" et "rs", puisqu'on a:

"wx…" $\angle$ "klmnop…" et "rs…" $\angle$ "klmnop…"

Les propriétés sont par contre vérifiées dans l'univers suivant:



que l'on peut représenter par une expression parenthésée:

c { d { e f g, a b } h, k l { m n, r s } o { p q, y z } } i j

On remarque que les événements $\lambda 1$, $\lambda 2$, … ne figurent pas dans l'expression. Leur rôle est donc uniquement de "marquer" les fins de séquences.[222]

D'autres expressions équivalentes peuvent être écrites en utilisant la propriété:

$\forall$ A, B $\in$ E^*, {A, B} représente la même structure que {B, A}.

Théorème VII.1

Si l'on peut associer à chaque événement e_i une date symbolique t_i (dans un ensemble strictement et totalement ordonné $\mathcal{D}$) et si l'univers E est partitionné en séquences, alors il possède les propriétés définies ci-dessus.

Preuve

On a vu qu'un univers de points temporels était nécessairement complet.

Soient deux séquences arbitraires:

$$S_i = e_{i1} \ldots e_{ip} \quad \text{et} \quad S_j = e_{j1} \ldots e_{jq}$$

Si l'on a $e_{i1} < e_{j1}$, alors on peut remplacer S_j par:

$$\lambda_j \, e_{j1} \ldots e_{jq} \quad \text{tel que} \quad e_{i1} = \lambda_j \, .$$

(On procède de manière symétrique si $e_{j1} < e_{i1}$.)[223]

De manière similaire, si $e_{ip} < e_{jq}$ on peut remplacer S_i par:

$$e_{j1} \ldots e_{jq} \, \lambda_i \quad \text{tel que} \quad e_{iq} = \lambda_i \, .$$

(De manière symétrique si $e_{jq} < e_{ip}$.)

On a maintenant $S_i \angle S_j$ et $S_j \angle S_i$. En transformant ainsi chaque paire de séquences on obtient ainsi un ordre total "$\angle$" qui respecte les propriétés. ■

Remarque: cette construction est triviale, mais en général il existe d'autres représentations plus pertinentes. On peut par exemple chercher à minimiser le nombre d'événements λ créés.

9.2 Notion de "tempo" (*notion of "tempo"*)

On décide maintenant d'associer à tout élément e_i d'un univers d'événements E un nombre rationnel strictement positif $V(e_i)$ que l'on appelle son **tempo**. On convient qu'il est possible d'inférer des relations '<' et '=' entre événements dans le cas suivant:

Règle d'inférence

Soient deux sous-séquences $\sigma_i \subset S_i$ et $\sigma_j \subset S_j$ telles que:

$$\sigma_i = e_i \ldots e_{i+p} \, , \quad \sigma_j = e_j \ldots e_{j+p} \, , \quad e_i = e_j$$

Les propositions suivantes sont vraies:

---

[222] Voir les objets 'NIL' en fin de séquence dans les représentations en tableau de structures polymétriques, §4 ou chapitre IX §2.

[223] En musique, les λ sont par exemple des silences.

$$\sum_{l=i}^{i+p-1} \frac{1}{V(e_l)} = \sum_{l=j}^{j+q-1} \frac{1}{V(e_l)} \Rightarrow e_{i+p} = e_{j+q}$$

$$\sum_{l=i}^{i+p-1} \frac{1}{V(e_l)} < \sum_{l=j}^{j+q-1} \frac{1}{V(e_l)} \Rightarrow e_{i+p} < e_{j+q}$$

Remarque

Les conditions de cohérence (voir §8.1) ne sont pas toujours respectées pour des valeurs arbitraires de tempos.

Théorème VII.2

Si tous les tempos des événements d'un univers d'événements E possédant les propriétés définies au §9.1 sont connus, alors le problème de synchronisation est résolu.

Preuve

L'univers étant partitionné en séquences, à chaque séquence est associé un sous-graphe complet de G. Soient deux séquences S_i et S_j telles que $S_i \angle S_j$.

Posons:

$S_i = e_{i1} \ldots e_{ip}$, $S_j = e_{j1} \ldots e_{jq}$, $m \in [1,q]$, $n \in [1,q]$, $e_{i1} = e_{jm}$, $e_{ip} = e_{jn}$.

En posant $\sigma_i = e_{i1}$ et $\sigma_j = e_{jm}$ on peut inférer la relation entre e_{i2} et $e_{j(m+1)}$. Puis on pose $\sigma_i = e_{i1}\, e_{i2}$ et $\sigma_j = e_{jm}\, e_{j(m+1)}$, ainsi de suite. Par ailleurs, on prouve facilement que

$$\forall\, l \in [1, m[\ ,\ ejl < ei1$$

$$\text{et}\ \ \forall\, l \in]n, q]\ ,\ eip < ejl\ .$$

Donc tous les éléments de S_i sont comparables aux éléments de S_j .

Si $S_i \angle S_l$ et $S_j \angle S_l$ alors $S_i << S_j$ ou bien $S_j << S_i$. Dans les deux cas tous les éléments de S_i sont comparables aux éléments de S_j .

Enfin, si $S_i \angle S_l$ et $S_j \angle S_m$, sachant que la restriction de G aux couples de $S_l \cup S_m$ est un graphe complet, alors on peut compléter G pour $S_i \cup S_l$ puis pour $S_m \cup S_j$, et enfin, en utilisant les règles de transitivité, pour $S_i \cup S_j$.■

9.3 Détermination des tempos (*tempo assignment*)

On considère un univers d'événements E possédant les propriétés définies au §9.1. Les règles proposées ici seront illustrées au chapitre VIII dans le cas des structures polymétriques.

9.3.1 Marque explicite

On peut indiquer le tempo d'un événement e_i en le faisant précéder, dans la séquence, d'une marque explicite de tempo, que nous notons '1/n'. Ainsi, si l'on écrit:

$$1/n\ e_i$$

alors

$$V(e_i) = n$$

On peut montrer qu'en l'absence de marque explicite de tempo, les règles 9.3.2 à 9.3.5 respectent la cohérence de l'univers d'événement.

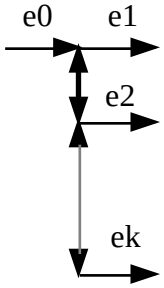
9.3.2 Tempo par défaut

Au début de la séquence maximale pour la relation '$\angle$', sauf indication contraire explicite, le tempo vaut 1.

9.3.3 Propagation après divergence

Si $e_0\ e_1$ est une séquence, et si $e_1 = e_2 = \ldots = e_k$, alors

$$\exists\ i \in [1,k] \quad \text{tel que}$$
$$V(e_i) = V(e_0) \quad \text{ou} \quad V(e_i) \text{ est imposé explicitement.}$$



Lorsque plusieurs séquences sont synchronisées, et qu'aucune ne débute par une marque explicite de tempo, on en choisit donc une au moins qui débute avec le tempo du dernier événement précédant la divergence.

Dans les formules polymétriques (chapitre VIII) la séquence choisie est le premier argument de l'expression parenthésée (voir §2.1, convention 6).

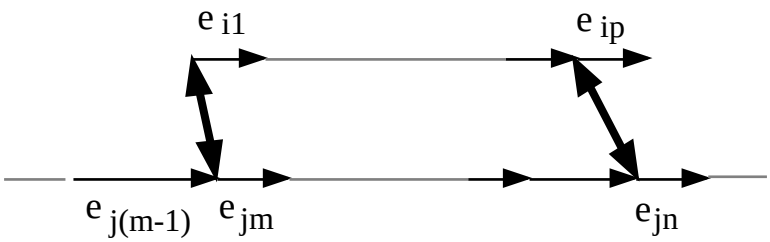
9.3.4 Propagation après convergence

Dans le cas où $S_i \angle S_j$, avec

$S_i = e_{i1} \ldots e_{ip}$, $S_j = e_{j1} \ldots e_{jq}$, $m \in]1,q]$, $n \in [1,q]$, $e_{i1} = e_{jm}$, $e_{ip} = e_{jn}$,

on a

$$V(e_{j(m-1)}) = V(e_{jn})$$



9.3.5 Propagation dans une séquence

Si $e_i\ e_j$ est une séquence, et s'il n'existe pas *k* tel que $e_j = e_k$, alors

$$V(e_j) = V(e_i)$$

Remarque: e_j n'est pas précédé d'une marque explicite de tempo.

Les règles du §9.3 permettent de déterminer certaines valeurs de tempo, et par conséquent d'inférer des relations ‘<’ et ‘=’ entre événements de E. Pour résoudre le problème de la synchronisation, il est par ailleurs nécessaire de choisir, pour les tempos non déterminés, des valeurs qui respectent la cohérence de E. Le chapitre VIII traite ce cas en considérant des objets temporels ou atemporels, ordonnés par un ensemble de dates symboliques (**structure polymétrique**). Le problème se ramène alors au calcul des dates inconnues, ou, ce qui revient au même, des durées symboliques des objets dans les séquences.

Le chapitre IX s'intéresse à l'**instanciation** (la “mise en temps”) des structures polymétriques prenant en compte des descriptions prototypiques des objets sonores et de leurs propriétés **métriques** et **topologiques**. Cette opération tient compte de contraintes de “voisinage” des objets dans les séquences. L'ensemble solution du système de contraintes est en général infini, et l'algorithme proposé permet de trouver, soit la première, soit toutes les solutions d'une partie finie de cet ensemble (solutions **canoniques**).

VIII. Synchronisation de séquences d'objets temporels/atemporels (*time-object/out-time object sequence synchronization*)[224]

Au chapitre VII §8 s'est posé le problème général de la synchronisation d'un univers d'événements. Nous proposons ici un algorithme de résolution dans le cas où les événements sont des objets temporels/atemporels. La synchronisation se ramène alors à un problème de calcul de durées (symboliques). Nous appelons **formule polymétrique** une expression parenthésée représentant un ensemble de séquences d'objets temporels (**structure polymétrique**).

La notation parenthésée des structures polymétriques est une extension à la **polyphonie** d'un système de notation initialement destiné aux séquences rythmiques.[225] On parle aussi de **polyrythmie** lorsque ces formes produisent un effet d'interférence entre plusieurs phénomènes périodiques.[226]

Le formalisme présenté dans ce chapitre est compatible avec le modèle théorique de processus concurrentiels (**traces**) proposé en 1977 par Mazurkiewicz[227] et appliqué à la musique par Chemillier[228]. (Voir annexe 7)

Un **algorithme d'interprétation** permettant de passer d'une formule polymétrique simple (notation *prescriptive*) à une formule **complète** (notation *descriptive*, voir §2) est présenté au §5. Cet algorithme vérifie aussi la cohérence de l'univers d'événements structuré par les relations de simultanéité et d'antériorité (voir chapitre VII §8.1). Un exemple d'application à la notation tonale est traité en annexe 4.

Il est important d'utiliser un minimum d'informations explicites pour interpréter les représentations symboliques. On peut dans ce cas utiliser la même représentation dans plusieurs contextes de réécriture. (Voir exemple §2.2) En l'absence d'information explicite, l'algorithme attribue aux séquences des durées symboliques par défaut; ces attributions sont effectuées de sorte que l'interprétation soit la plus “simple” possible, au sens du nombre de symboles de la formule complète. (Voir exemple §3)

Les logiciels du type “séquenceur” imposent une limite fixe de précision des intervalles temporels, et par conséquent une approximation pour certains rapports de durées. Un avantage de la méthode présentée ici est qu'elle permet, pour toute structure d'objets temporels, de calculer la résolution temporelle la plus grossière respectant rigoureusement

---

224 Une première version de ce chapitre a été publiée dans *Interface* (Bel 1990e). Les idées essentielles ont été rééditées dans un rapport en version anglaise (Bel 1991a) dont une version simplifiée (Bel 1991c) sera publiée.

225 Ce système de notation est à son tour une adaptation de la notation musicale utilisée par certains percussionnistes (Kippen 1988, pp.123-7).

226 Voir par exemple les interférences et la polyphonie de Schillinger, Ames 1989, pp.9,30.

227 1984a-b

228 1987

les rapports de durées. Ce paramètre sert à déterminer la taille minimale du tableau des phases (voir chapitre IX §2.1) utilisé pour la mise en temps des objets sonores.

1. Représentation de séquences (*representing sequences*)

La notation parenthésée des formules polymétriques est une extension de celle des séquences. Nous décrivons en premier les règles syntaxiques permettant de noter les séquences d'objets temporels.

Soit un ensemble de dates symboliques $\mathcal{D}$, i.e. un intervalle $[0,t_N]$ de Q^+, l'ensemble des nombres rationnels positifs ou nuls.

Séquence: définition

> Une **séquence** est un couple (S,θ), où $S = s_1 \ldots s_n$ désigne une chaîne sur E* (un ensemble d'étiquettes d'objets temporels ou atemporels), et θ une application injective de S dans $\mathcal{D}$ (sa structure rythmique).

Nous avons proposé de noter toute séquence comme une liste de couples (s_i,d_i), où d_i est la durée symbolique de l'objet étiqueté par s_i, par exemple:

$$S = (a,1)\ (b,1)\ (c,2)\ (d,2/3)\ (e,1/3)\ .$$

où “a”, “b”, “c”, “d” et “e” sont les étiquettes des objets. Dans le cas où la durée symbolique est nulle on a affaire à un objet atemporel (voir chapitre VII §4-5).

Nous adoptons les conventions suivantes:

> 1. Le dénominateur des fractions peut précéder le couple.

Ainsi, (d,2/3) est noté

$$/3\ (d,2)$$

et la séquence S s'écrit:

$$S = (a,1)\ (b,1)\ (c,2)\ /3\ (d,2)\ /3\ (e,1)$$
$$= /1\ (a,1)\ /1\ (b,1)\ /1\ (c,2)\ /3\ (d,2)\ /3\ (e,1)$$

Nous appelons **marque explicite de tempo** l'expression “/3”. On dit informellement “la vitesse 3”.

> 2. Les marques de tempo ne sont pas nécessairement répétées.

La séquence peut donc s'écrire:

$$S = (a,1)\ (b,1)\ (c,2)\ /3\ (d,2)\ (e,1)$$

> 3. Tout objet vide “prolonge” l'objet qui le précède.

On peut donc écrire:

$$S = (a,1)\ (b,1)\ (c,1)\ (\ \_\ ,1)\ /3\ (d,1)\ (\ \_\ ,1)\ (e,1)$$

> 4. On peut supprimer les parenthèses et l'indication de durée pour les objets de durée symbolique 1.

On aboutit donc à la notation[229]:

S = a b c _ /3 d _ e

Il est intéressant par ailleurs de pouvoir supprimer les parenthèses pour un objet atemporel (chapitre VII §6). Il suffit pour cela de noter différemment l'étiquette de cet objet. Nous avons adopté la notation <<a>> pour désigner un objet atemporel (correspondant au même objet sonore que l'objet temporel *a*). Ainsi, on peut écrire:

(a,1) (f,0) (b,1) (c,1) (g,0) (_ ,1) /3 (d,1) (_ ,1) (e,1)
= a <<f>> b c <<g>> _ /3 d _ e .

Il est clair, d'après la convention 4, qu'en l'absence d'indication contraire toute séquence débute au tempo 1.

> 5. Un silence d'une unité peut se noter “1” et, le cas échéant, se combiner avec la marque explicite de tempo qui le précède pour former un silence fractionnaire.

Ainsi par exemple,

(a,1) (b,1) (- ,1/3) (- ,1/3) (c,1) (d,1)
= a b /3 - /3 - /1 c d
= a b 1/3 1/3 /1 c d
= a b 2/3 /1 c d

formule pour laquelle nous admettons la notation:

a b 2/3 c d .

On peut aussi écrire ainsi des silences de durée fractionnaire supérieure à 1:

/5 a b c 43 3/4 d e f

qu'il faut interpréter

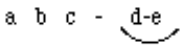
(a,1/5) (b,1/5) (c,1/5) (-,43/5) (-,3/20) (d,1/5) (e,1/5) (f,1/5)
= (a,1/5) (b,1/5) (c,1/5) (-,172/20) (-,3/20) (d,1/5) (e,1/5) (f,1/5)
= (a,1/5) (b,1/5) (c,1/5) (-,175/20) (d,1/5) (e,1/5) (f,1/5)

puisque deux silences consécutifs équivalent à un silence de la somme de leurs durées.

La notation proposée ici est donc très souple en ce qui concerne les formules rythmiques. En comparaison, la notation classique occidentale ne fait appel, pour coder

---

[229] Cette notation est proche de celle utilisée par les percussionnistes de l'Inde. Une différence est que l'on confond, en percussion, le symbole de l'objet vide et celui du silence. D'autre part, dans les notations modernes on indique souvent le tempo en regroupant les objets dont la somme des durées est l'unité, par exemple:



les durées, qu'à des fractions dont les dénominateurs sont des puissances de 2, sauf localement dans les triolets, etc.

Il va de soi que lorsque deux marques explicites de tempo se succèdent la première peut être supprimée, par exemple:

a b c /3 /2 d e = a b c /2 d e

2. Interprétation de formules polymétriques (*interpreting a polymetric structure*)

(Voir annexe 7 pour une définition à partir de l'algèbre des traces.)

Nous appelons **formule polymétrique** toute expression de la forme

{A1,…,Ak}

dans laquelle Ai est une étiquette d'objet temporel, une séquence d'étiquettes d'objets atemporels suivie d'une étiquette d'objet temporel, ou encore une séquence de formules polymétriques.

La formule est **complète** si et seulement si

$\forall i,j \in [1,k]$, $|Ai| = |Aj|$

où |Ai| dénote le nombre de symboles de la séquence Ai.

Interpréter une formule polymétrique consiste à construire une structure polymétrique complète associant aux mêmes objets les dates (et donc les durées) déduites de la formule. L'interprétation est impossible lorsque les durées symboliques de deux arguments de la formule sont inégales. On dit dans ce cas que la formule (i.e. l'univers d'événements) est **incohérente**. Par exemple:

{/2 a b c d, /3 e f g h i j} = /6 {a _ _ b _ _ c _ _ d _ _ , e _ f _ g _ h _ i _ j _ }

dans laquelle chaque séquence a une durée symbolique totale de 12/6 = 2, est cohérente. Par contre,

{/3 a b, /2 c d} = /6 {a _ b _ , c _ _ d _ _ }

est incohérente car il ne peut exister de bijection entre les objets des deux séquences.

2.1 Indétermination des durées (*undetermined durations*)

Nous avons vu (convention 4) que le tempo d'une séquence était 1 en l'absence d'indication contraire. Cette propriété n'est pas vraie lorsque la séquence est précédée d'un symbole “{” ou “,”. Il y a donc une indétermination sur les durées des séquences, ce qui permet d'écrire, par exemple

{ a b, c d e } = {a _ _ b _ _ , c _ d _ e _ }
= {a _ _ _ _ _ _ b _ _ _ _ _ , c _ _ _ d _ _ _ e _ _ _ }
= etc…

ce qui revient à écrire

{ a b, c d e } = /x {a _ _ b _ _ , c _ d _ e _ }

où *x* est un entier désignant le tempo de la structure polymétrique.

Par contre, l'indétermination est levée si un argument au moins de la formule contient une marque explicite de tempo, par exemple

{/2 ab {/3 a b c, d e}, f g h i}
= {/2 ab {/3 a b c, /2 d e}, f g h i}
= {/2 ab /6 {a _ b _ c _ , d _ _ e _ _ }, f g h i}
= {/6 a _ _ b _ _ {a _ b _ c _ , d _ _ e _ _ }, f g h i}
= {/6 a _ _ b _ _ {a _ b _ c _ , d _ _ e _ _ }, /6 f _ _ g _ _ h _ _ i _ _}
= /6 {a _ _ b _ _ {a _ b _ c _ , d _ _ e _ _ }, f _ _ g _ _ h _ _ i _ _} ,

le calcul s'effectuant en propageant les contraintes d'égalité des durées symboliques des arguments de chaque formule.

Lorsque les durées sont indéterminées nous utilisons une nouvelle convention pour lever l'ambiguïté du tempo:

> 6. Par défaut, le tempo d'une structure polymétrique est déterminé par celui de son premier argument pris isolément.[230]

Reprenons l'exemple:

{ a b, c d e } = /x {a _ _ b _ _ , c _ d _ e _ }

dans lequel le premier argument, pris isolément, est:

a b = /1 a b .

La formule s'interprète donc:

{/1 a b, c d e } = /3 {a _ _ b _ _ , c _ d _ e _ } , donc x = 3

[230] Le fait de prendre un argument comme référence s'impose en raison de simplifications du type:

{ a b c } = a b c = /1 a b c

Dans une version précédente du BP2 (Bel 1990), l'argument de référence était celui qui contenait le plus grand nombre de symboles. Cette convention présentait l'avantage de ne pas introduire de dissymétrie dans la notion de structure polymétrique. Toutefois elle ne permettait pas de représenter simplement des structures de durée (relative) déterminée comme par exemple:

{ - , a b c d}

de même durée que

{ - , a , b, c, d}

(On peut donner comme exemple un arpège et un accord de durées identiques.)

En utilisant un silence (ou un objet vide) comme premier argument on peut écrire n'importe quelle séquence sans marque explicite de tempo, c'est à dire uniquement avec des durées symboliques relatives:

a b c _ /3 d _ e
= a b c _ /3 { - - -, d _ e}
= a b c _ /3 {3, d _ e}
= a b c _ {3/3, d _ e }
= a b c _ {1, d _ e}
(= a b c _ { - , d _ e })
(= a b c _ { _ , d _ e})

2.2 Réécriture des formules polymétriques (*rewriting polymetric expressions*)

L'avantage de la notation précédente est évident lorsque cette formule est utilisée dans une grammaire du type:

S —> /2 A /3 A
A —> a b c _ { _ , d _ e}

dont l'unique dérivation terminale est:

/2 a b c _ { _ , d _ e} /3 a b c _ { _ , d _ e}
= /6 a _ _ b _ _ c _ _ _ _ _ { _ _ _ , d _ e} a _ b _ c _ _ _ { _ _ , d _ e}
= /6 a _ _ b _ _ c _ _ _ _ _ { _ _ _ , d _ e} a _ b _ c _ _ _ /6 { _ _ , d _ e}
= /6 a _ _ b _ _ c _ _ _ _ _ { _ _ _ , d _ e} a _ b _ c _ _ _ /18 { _ _ _ _ _ , d _ _ _ e _ }
= /6 a _ _ b _ _ c _ _ _ _ _ { _ _ _ , d _ e} a _ b _ c _ _ _ /9 { _ _ _ , d _ e}
= /6 a _ _ b _ _ c _ _ _ _ _ d _ e a _ b _ c _ _ _ /9 d _ e .

3. Silences indéterminés (*undetermined rests*)

Pour qu'une formule polymétrique soit cohérente il faut et il suffit que les durées symboliques de ses arguments soient égales. Il est possible, dans un argument de durée connue, de placer un silence de durée indéterminée, noté "…", par exemple:

{/3 a … b, /2 c d} = {/3 a - b, /2 c d}
{a {b c, d e f}, /2 … g h i j } = {/1 a {/1 b c, d e f}, /2 - - g h i j }

Dans le premier exemple, la durée de la structure est fixée par le second argument "/2 cd", de durée 1. Si *x* est la durée du silence indéterminé, on a pour le premier argument

$$1 = \text{durée(a)} + x + \text{durée(b)} = 1/3 + x + 1/3$$

soit x = 1/3. Dans le deuxième exemple, la durée du premier argument est 3. On a donc

$$3 = x + \text{durée(g)} + \text{durée(h)} + \text{durée(i)} + \text{durée(j)}$$

chaque durée étant fixée à 1/2 a cause de la marque explicite de tempo "/2" qui précède la séquence. On obtient donc x = 1 = 2/2. Dans ces deux exemples le tempo de l'argument qui contient un silence indéterminé est fixé par une marque explicite "/3" ou "/2".

Il est clair que la durée d'un silence indéterminé est égale à

$$\frac{pgap[a]}{qgap[a]} = \frac{pmax}{qmax} - \frac{p[a]}{q[a]}$$

où pmax/qmax est la durée de la structure (i.e. celle de l'argument pris comme référence), *a* le rang de l'argument contenant le silence indéterminé, et p[a]/q[a] la somme des durées des termes définis dans cet argument. Dans le cas où cette différence est négative le système retourne un message d'erreur: "*Pas assez de temps pour insérer un silence*".

Considérons maintenant le cas où l'argument de rang *a,* qui contient un silence indéterminé, ne contient pas de marque explicite de tempo. Nous supposons que la durée de la formule pmax/qmax est déjà connue. La valeur de p[a]/q[a] est remplacée par p[a]/(m.q[a]), où *m* est le tempo indéterminé de l'argument de rang *a.* On a donc:

$$\frac{pgap[a]}{qgap[a]} = \frac{pmax}{qmax} - \frac{p[a]}{m.q[a]}$$

La valeur minimale de *m* qui garantisse un résultat positif ou nul est:

$$m = \text{Partie entière de } \left(\frac{p[a].qmax}{q[a].pmax}\right)$$

Nous considérons que cette valeur est celle qui donne la solution la plus "évidente" du problème, les autres solutions restant accessibles à condition que les informations fournies soient suffisantes.

4. Représentations minimale et dilatée d'une formule polymétrique (*minimal and stretched representations of a polymetric expression*)

4.1 Simplification d'une séquence (*simplifying a sequence*)

Simplifier une séquence revient à utiliser le nombre minimum de symboles pour la représenter. La réduction se fait en supprimant des objets vides là où les fractions représentant les durées sont réductibles, par exemple:

/6 a _ _ b _ _ c _ d _ = /2 a b /3 c d

/5 a _ _ b _ _ c _ _ _ _ d _ _ _ = /5 a _ _ b _ _ /1 c /5 d _ _ _

4.2 Changement d'échelle (*scale adjustment*)

Dans le deuxième exemple précédent on ne peut pas supprimer tous les objets vides parce que les fractions 3/5 et 4/5 qui indiquent les durées symboliques de "a", "b" et respectivement de "c" ne peuvent pas s'écrire 1/x avec *x* entier. Si l'on applique à l'échelle des dates symboliques une homothétie de centre 0 et de rapport *s*, par exemple s = 24, tous les tempi sont multipliés par *s* et l'on obtient:

/120 a _ _ b _ _ /24 c /120 d _ _ _
= /40 a b /24 c /30 d

Nous appelons **minimale** ce type de représentation. Si l'on avait choisi s = 12 de sorte que les tempi fussent premiers entre eux, on aurait obtenu une représentation unique de la séquence (représentation **canonique**): /20 a b /12 c /15 d.

Appelons *Prod* le plus petit commun multiple (ppcm, *LCM*) de 40, 24 et 30. On a Prod = 120. On obtient facilement la formule initiale en divisant 120 par les tempi. Ainsi, pour la sous-séquence “ab” le nombre d'objets vides à ajouter est (120 / 40) - 1 = 2. Cette opération très simple permet d'écrire la séquence sans marque de tempo:

/40 a b /24 c /30 d
s'écrit: a _ _ b _ _ c _ _ _ _ d _ _ _

La première formulation (ou toute représentation minimale) correspond à la représentation interne dans le BP2. La seconde (représentation **dilatée**) est celle qui apparaît à l'écran.

Par rapport à la formule initiale, la formule dilatée multiplie par 5 les durées puisque la marque de tempo “/5” n'apparaît pas. Nous appelons **facteur d'échelle** ce coefficient. Il est facile de vérifier que, dans le cas général, le facteur d'échelle est:

Ratio = Prod / s

4.3 Formule polymétrique (*polymetric expression*)

Les opérations qui s'appliquent aux séquences (changement d'échelle du temps symbolique et dilatation des durées) s'appliquent de la même manière aux formules polymétriques. Il suffit que le facteur de changement d'échelle *s* soit identique pour tous les arguments de la formule. Si $s_1,\ldots, s_k$ sont les changements d'échelles utilisés pour les représentations minimales des arguments A1,…, Ak de la formule, on prendra pour *s* n'importe quel multiple commun de $s_1,\ldots, s_k$.

Lors de l'évaluation d'une formule polymétrique on cherche à minimaliser la représentation; par conséquent on opère autant de changements d'échelle que nécessaire pour ne pas ajouter d'objets vides. On obtient donc une expression minimale lorsque les arguments de la formule et des sous-formules sont sous forme minimale. Il est facile ensuite de passer à la représentation dilatée pour obtenir la formule complète.

5. Algorithme d'interprétation (*interpretation algorithm*)

Etant donnée une formule polymétrique quelconque[231], l'algorithme doit fournir cinq résultats:

(1) Une valeur succès/échec qui indique si la formule est cohérente;

Et, en cas de succès:

(2) La formule complète (ou sa représentation minimale)

(3) La durée symbolique P/Q de la formule

(4) La valeur du changement d'échelle *s* et le facteur d'échelle *Ratio*

(5) Une valeur *fixtempo* (vrai/faux) indiquant si la formule contient une marque explicite de tempo.

L'algorithme est donné en annexe 3. Les étapes les plus importantes sont décrites ci-dessous. Un exemple de calcul est ensuite proposé.

[231] Dans le cas d'une séquence A on écrit { A } .

Appelons *k* le nombre d'arguments de la formule ($k \geq 1$), et p[a] / q[a] la durée symbolique de l'argument de rang *a* (pour $1 \leq a \leq k$). Appelons fixtempo[a] une variable booléenne qui indique si le champ de rang *a* contient ou non une marque explicite de tempo.

5.1 Analyse (*analysis*)

Lorsque l'on évalue une formule polymétrique, on a en entrée une facteur de changement d'échelle *oldscale* qui est celui de la séquence qui précède la formule (ou 1 si cette séquence est vide).

On commence à analyser l'argument de rang *a* (de gauche à droite) avec un facteur de changement d'échelle initial:

scale[a] <— oldscale

Le tempo par défaut est:

tempo <— oldscale

Lorsque l'on rencontre une marque explicite de tempo “/x”, le tempo devient:

tempo <— oldscale . x

On en profite pour affecter “vrai” aux variables *fixtempo* et fixtempo[a].

Si l'on rencontre une formule polymétrique F, son évaluation retourne un facteur de changement d'échelle *s*. Ce changement d'échelle affecte la partie de l'argument déjà lue. Le facteur de changement d'échelle de la séquence devient

scale[a] <— s . scale[a]

et le tempo:

tempo <— s . tempo

Pour prendre en compte ce nouveau changement d'échelle, il faudrait multiplier par *s* toutes les marques de tempo déjà lues dans l'argument. Il est préférable d'insérer une marque de changement d'échelle “\s” avant la copie de F, pour effectuer ces corrections en une seule fois (voir **Corrections d'échelle** dans l'algorithme).

A noter que si l'argument de rang *a* contient une ou plusieurs formules polymétriques, son facteur de changement d'échelle scale[a] après analyse reste un multiple de *oldscale*.

5.2 Résolution (*resolution*)

Soit a0 la valeur minimale de *a* telle que fixtempo[a] est vrai et que l'argument de rang *a* ne contienne pas de silence indéterminé (i.e. vargap[a] = faux). Par défaut, a0 est la plus petite valeur de *a* telle que vargap[a] = faux.

La durée symbolique de la formule est:

$$\frac{p[a0]}{q[a0]} = \frac{P}{Q}$$

Pour tout $a \leq k$ appelons r[a] un entier qui réalise la condition:

$$[1] \qquad \forall\, a \in (1,k),\quad \frac{p[a]}{r[a]\,.\,q[a]} = \frac{p[a0]}{r[a0]\,.\,q[a0]} = \frac{P}{r[a0]\,.\,Q}$$

On cherche un ensemble d'entiers positifs {r[1],…, r[k]} aussi petits que possible satisfaisant la relation ci-dessus. Appelons L le PPCM (plus petit commun multiple, *Lowest Common Multiple*) des p[a], et posons:

$$pp[a] = \frac{L}{p[a]}$$

En divisant par L les deux membres de l'équation [1] on obtient:

$$[2] \qquad \forall a \in (1, k),\quad r[a] \,.\, q[a] \,.\, pp[a] \;=\; r[a0] \,.\, q[a0] \,.\, pp[a0]$$

Calculons maintenant M, le PPCM des (q[a] . pp[a]), et posons:

$$qq[a] = \frac{M}{q[a] \,.\, pp[a]}$$

En divisant par M les deux membres de l'équation [2] on obtient:

$$[3] \qquad \forall\ a \in (1,k),\quad \frac{r[a]}{qq[a]} = \frac{r[a0]}{qq[a0]}$$

Les plus petites valeurs entières de r[a] satisfaisant cette équation sont évidemment:

$$[4] \qquad \forall a \in (1, k),\quad r[a] = qq[a]$$

sachant que:

$$qq[a0] = r[a0] = \frac{M}{q[a0] \,.\, pp[a0]} = \frac{M \,.\, L}{p[a0] \,.\, q\,[a0]}$$

Informellement, on a multiplié la durée symbolique p[a]/q[a] de l'argument de rang *a* de la formule par un rapport r[a0]/r[a] de manière à garantir la simultanéité du début et de la fin de la séquence qu'il décrit avec ceux des autres séquences de la formule. La durée totale de la formule est inchangée, mais on doit lui appliquer maintenant un facteur de changement d'échelle *s* multiple de r[a0].

5.3 Silences indéterminés (*undetermined rests*)

On n'a pas pris en compte jusqu'ici les arguments tels que vargap[a] = vrai. On calcule maintenant, pour chacun d'eux, la valeur:

$$\frac{pgap[a]}{qgap[a]} = \frac{pmax}{qmax} - \frac{p[a]}{q[a]}$$

Si ce nombre est négatif et qu'on a par ailleurs fixtempo[a] = vrai, le message "*Erreur: pas assez de temps pour insérer un silence*" est retourné. Dans le cas contraire, on détermine le tempo de l'argument de rang *a*:

$$m \;=\; \text{Partie entière de } \left(\frac{p[a].qmax}{q[a].pmax}\right)$$

puis on calcule

$$\frac{pgap[a]}{qgap[a]} = \frac{pmax}{qmax} - \frac{p[a]}{m.q[a]}$$

Dans tous les cas, si aucun message d'erreur n'a été retourné, on relance l'analyse. Lorsqu'on rencontre un silence indéterminé dans l'argument de rang *a* on le remplace par le silence fractionnaire pgap[a]/qgap[a].

5.4 Calcul de *s* (*calculating* s)

Le problème reste de trouver une valeur minimale de *s*, le changement d'échelle de la formule, puis de réécrire les arguments de la formule en respectant les variations de durée. Pour que la formule reste minimale on ne doit modifier les durées qu'en jouant sur les valeurs des marques de tempo et sur l'échelle.

Soit "/x" une marque de tempo de l'argument de rang *a.* La valeur de *x* a pu être modifiée pendant l'analyse de cet argument (s'il contient une ou plusieurs formules polymétriques). Dans ce cas, scale[a] > oldscale. On a toutefois nécessairement

$$x \cdot \frac{\text{oldscale}}{\text{scale[a]}}$$

entier. Ce nombre est en effet la valeur de la marque de tempo avant analyse.

Multiplier la durée de l'argument de rang *a* par r[a0]/r[a] revient à multiplier la valeur de la marque de tempo initiale par r[a]/r[a0], sous réserve que le résultat soit entier. Si nous appliquons maintenant le changement d'échelle *s* de la formule, la valeur définitive de la marque de tempo doit être:

$$x' = s \cdot \frac{x}{\text{scale[a]}} \cdot \text{oldscale} \cdot \frac{r[a]}{r[a0]}$$

La valeur de *s* correcte est donc celle qui garantit un résultat entier pour x' dans toutes les marques de tempo de tous les arguments. Puisque s/r[a0] est entier, on peut ramener ce problème à celui de la recherche de ss = s / r[a0].

On a donc:

$$x' = ss \cdot \frac{\text{oldscale.}\ r[a] \cdot x}{\text{scale[a]}}$$

Pour que ce résultat soit entier, la valeur minimale de ss est donc calculée en prenant une valeur initiale ss = 1 et en appliquant la formule:

$$ss = \text{PPCM}\ (ss, \frac{\text{scale[a]}}{\text{PGCD}\ (\text{oldscale.r[a].x},\ \text{scale[a]})})$$

pour toutes les marques "/x" apparaissant dans l'argument de rang *a,* et pour $1 \leq a \leq k$. "PGCD" est le plus grand commun diviseur.

On en déduit:

$$s = ss \cdot r[a0]$$

L'algorithme se termine par les corrections d'échelle et la copie dans B[] de la formule avec les nouvelles marques de tempo.

6. Exemple de traitement (*example of processing*)

Considérons la formule:

{i {a b, c d e}, j k}

où “a”, “b”, “c”, “d”, “e”, “j” et “k” sont des symboles terminaux. Cette formule ne contient aucune marque explicite de tempo, elle sera donc évaluée comme:

/1 {i {a b, c d e}, j k}

La procédure POLY() est appelée (récursivement) chaque fois qu'on rencontre le séparateur “{”. Les deux arguments de la formule, au premier niveau d'appel, sont:

argument 1: i {a b, c d e}

argument 2: j k

La valeur (par défaut) de oldscale est 1. On commence par analyser l'argument 1, avec tempo = 1 et scale[1] = 1. Lorsqu'on rencontre “{” la procédure POLY() est appelée au deuxième niveau et les arguments de la formule sont:

argument 1: a b

argument 2: c d e

On a encore oldscale = 1 et on analyse chaque argument de rang *a* avec tempo = 1 et scale[a] = 1. On obtient:

$$\frac{p[1]}{q[1]} = \frac{2}{1} \qquad \text{et} \qquad \frac{p[2]}{q[2]} = \frac{3}{1}$$

Puisqu'aucun argument ne contient de marque explicite de tempo, on a a0 = 1 et par conséquent P/Q = p[1]/q[1] = 2.

On calcule L = PPCM des p[a] = PPCM (2,3) = 6.

On a donc pp[1] = L/p[1] = 3 et pp[2] = L/p[2] = 2.

On calcule maintenant M = PPCM des q[a].pp[a]. M = PPCM (3,2) = 6.

On en déduit les r[a] = M / (q[a].pp[a]):

$$r[1] = \frac{6}{1 \times 3} = 2 \quad \text{et} \quad r[2] = \frac{6}{1 \times 2} = 3$$

On pose ensuite ss = 1 et on applique, pour toutes les marques de tempo dans les deux arguments:

$$ss = PPCM\,(ss, \frac{scale[a]}{PGCD\,(oldscale.r[a].x,\ scale[a])})$$

Puisque scale[1] = scale[2] = 1, on a nécessairement:

$$ss = 1 \qquad \text{et} \qquad s = r[a0] = r[1] = 2$$

Dans la formule {a b, c d e} on avait des marques implicites de tempo

{/x a b, /x c d e}

avec x = 1 par défaut. Ces valeurs sont maintenant corrigées avec la formule

$$x' = s \cdot \frac{x}{scale[a]} . oldscale \cdot \frac{r[a]}{r[a0]}$$

ce qui donne:

{/2 a b, /3 c d e}

(Intuitivement on comprend que “ab” doit être exécuté à la vitesse 2 et “abc” à la vitesse 3 pour qu'il y ait synchronisme.)

On sort de la procédure POLY() (au 2ème niveau) avec P/Q = 2/1 et s = 2. Puisque s > 1, au niveau 1 dans l'analyse du premier argument on a maintenant scale[1] = 2 et tempo = 2. On insère une marque de changement d'échelle “\2” avant de copier la formule précédemment déterminée:

/1 {/1 i \2 {/2 a b, /3 c d e} …

puis on indique explicitement le nouveau tempo:

/1 {/1 i \2 {/2 a b, /3 c d e} /2…

L'analyse du premier argument au niveau 1 se termine avec:

scale[1] = 2 et p[1]/q[1] = 3/1

L'analyse du deuxième argument “jk” donne évidemment:

scale[2] = 1 et p[2]/q[2] = 2/1

On a de nouveau a0 = 1 et donc P/Q = 3/1. On obtient maintenant:

L = 6, pp[1] = 2, pp[2] = 3, M = 6, r[1] = 3 et r[2] = 2

On effectue enfin la correction d'échelle du premier argument:

/1 i \2 {/2 a b, /3 c d e} /2

ce qui donne:

/2 i {/2 a b, /3 c d e} /2

On pose de nouveau ss = 1, et on applique, pour toutes les marques de tempo du premier argument:

$$ss = PPCM\,(ss, \frac{scale[1]}{PGCD\,(oldscale.r[1].x,\ scale[1])}) = PPCM\,(ss, \frac{2}{PGCD\,(x,\ 2)})$$

ce qui donne ss = 2. On fait de même pour l'argument 2:

$$ss = PPCM\,(ss, \frac{scale[2]}{PGCD\,(oldscale.r[2].x,\ scale[2])}) = PPCM\,(ss, \frac{1}{PGCD\,(x,\ 1)})$$

ce qui donne encore ss = 2.

On en déduit:

s = ss.r[a0] = ss.r[1] = 2 x 3 = 6

On copie enfin la formule:

{/2 i {/2 a b, /3 c d e} /2, /1 j k}

dans B[] en appliquant les corrections de tempo:

$$x' = s \cdot \frac{x}{scale[a]} . oldscale . \frac{r[a]}{r[a0]}$$

ce qui donne, pour l'argument 1,

$$x' = 6/2 = 3$$

et pour l'argument 2:

$$x' = 6 \times 2/3 = 4$$

et par conséquent pour la formule finale:

{/6 i {/6 a b, /9 c d e} /6, /4 j k}

qu'on peut écrire plus simplement:

{/6 i {/6 a b, /9 c d e}, /4 j k}

On sort alors de la procédure POLY() au niveau 1 avec s = 6 et P/Q = 3/1. On calcule alors *Prod*, le PPCM des tempi

Prod = PPCM (6,9,4) = 36

et de même:

Ratio = Prod / s = 36/6 = 6

La formule dilatée est obtenue en divisant *Prod* par chaque tempo, ce qui donne la durée symbolique de chaque terminal:

{i _ _ _ _ _ {a _ _ _ _ _b _ _ _ _ _ , c _ _ _ d _ _ _ e _ _ _ }, j _ _ _ _ _ _ _ _ k _ _ _ _ _ _ _ _ }

Le tableau correspondant est:

```
┌──────────────────────────────────────┐
│ i _ _ _ _ _ a _ _ _ _ _ b _ _ _ _ _  │
│ _ _ _ _ _ _ c _ _ _ d _ _ _ e _ _ _  │
│ j _ _ _ _ _ _ _ _ k _ _ _ _ _ _ _ _  │
└──────────────────────────────────────┘
```

7. Traitement des objets atemporels (*processing out-time objects*)

Dans ce qui précède on suppose que la formule polymétrique puisse contenir des objets atemporels (voir définition chapitre VII §6), sous condition que toute séquence d'objets atemporels (i.e. de durée symbolique nulle) se termine par un objet temporel (i.e. de durée symbolique non nulle).

Il est facile de voir que l'algorithme précédent s'applique dans la mesure où il suffit de traiter une chaîne représentant une séquence d'objets atemporels comme un préfixe de la chaîne représentant l'objet temporel qui lui succède immédiatement.

Prenons un exemple semblable au précédent:

{i {a <<h>> b, c d e}, j <<f>> <<g>> k}

= {/6 i {/6 a <<h>> b, /9 c d e}, /4 j <<f>> <<g>> k}

= {i_ _ _ _ _ {a_ _ _ _ _<<h>> b_ _ _ _ _ ,c_ _ _ d_ _ _ e_ _ _}, j_ _ _ _ _ _ _ _ <<f>> <<g>> k_ _ _ _ _ _ _ _ }

ce qui donne le tableau suivant:

i	_	_	_	_	_	a	_	_	_	_	_	b	_	_	_	_	_
_	_	_	_	_	_	_	_	_	_	_	_	<<h>>					
_	_	_	_	_	_	c	_	_	_	d	_	_	_	e	_	_	_
j	_	_	_	_	_	_	_	_	k	_	_	_	_	_	_	_	_
_	_	_	_	_	_	_	_	_	<<f>>								
_	_	_	_	_	_	_	_	_	<<g>>								

8. Conclusion

Un exemple d'application à la notation tonale de l'algorithme de résolution de formules polymétriques est proposé en annexe 4. L'exemple montre que l'on peut faire une économie d'écriture sur des formules rythmiques complexes, en comparaison avec la représentation utlisée dans la plupart des séquenceurs. Dans un grand nombre de cas, une représentation incomplète est suffisante dans la mesure où l'algorithme recherche la solution la plus simple compatible avec les données.

Toutefois, l'intérêt majeur de cette méthode est de simplifier la représentation de “familles” de structures, dans la mesure où chaque partie constitutive de la structure peut être définie incomplètement, les durées symboliques étant ensuite ajustées par l'algorithme en fonction des contraintes “horizontales” (le tempo dans une séquence) et “verticales” (la synchronisation de séquences).

IX. Mise en temps d'objets sonores (*time-setting of sound-objects*)[232]

On s'intéresse dans ce chapitre à la **mise en temps** de structures polymétriques. Chaque structure est une juxtaposition de séquences représentées par des chaînes de symboles. Ces symboles sont les étiquettes d'objets sonores. On a désigné par **objet temporel** le couple (symbole, date symbolique). A chaque symbole correspond un (et un seul) **prototype d'objet sonore** unique. Ce prototype comprend une séquence de **messages**[233] qui déclenchent (ou modifient)[234] un processus dans un système de production sonore, et une liste de **propriétés métriques et topologiques**. Un **objet sonore** est une **instance** de l'objet temporel de même étiquette, dans laquelle une liste de dates physiques est associée à la liste de messages de son prototype.[235] Une structure polymétrique est **instanciée** lorsque tous ses objets temporels sont instanciés.

L'instanciation d'un objet temporel est fonction:

— de la structure Φ du temps (voir chapitre VII §4);

— de sa durée symbolique dans la structure polymétrique et du facteur d'échelle de la formule (voir chapitre VII §4 et chapitre VIII §4.2);

— de la durée physique du prototype d'objet sonore correspondant;

— de propriétés “métriques” permettant de calculer sa position et sa “dilatation/contraction” (voir §3);

— de propriétés “topologiques” qui définissent les “contraintes de voisinage” entre objets d'une même séquence (voir §5).

Si l'on ne tient pas compte des propriétés topologiques, on peut faire correspondre à toute structure polymétrique une instanciation unique. Les propriétés topologiques introduisent un système de contraintes dont l'ensemble des solutions est en général vide ou infini. (Des procédures interactives, non décrites ici, permettent de relâcher graduellement les contraintes lorsque l'ensemble des solutions est vide.) Les ensembles solutions infinis contiennent une partie finie “intéressante” (les solutions **canoniques**)

---

232 Ce chapitre a été réédité dans un rapport en version anglaise (Bel 1991a) dont une version simplifiée (Bel 1991c) sera publiée.

233 La notion de message fait implicitement référence à un réseau qui comprend un ordinateur, un instrument de musique et un ou plusieurs synthétiseurs (*expanders*) reliés par une liaison de données (par exemple le standard de communications MIDI — *Musical Instrument Digital Interface*). Toutefois le problème dépasse le cadre de l'informatique musicale dans la mesure où l'on peut remplacer tout message par un geste sur un instrument.

234 A chaque message correspond une action élémentaire sur un instrument (réel ou virtuel): enfoncement d'une touche, modification de la position d'une manette, etc.

235 Un message associé à une action élémentaire peut déclencher un processus complexe dans le terminal de synthèse, processus qui comporte un nombre virtuellement illimité d'étapes. On passe ainsi d'une structure finie discrète à un phénomène (perçu comme) continu.

qui correspond à un ensemble fini de modifications de position satisfaisant les contraintes. Nous présentons un algorithme qui permet, soit de trouver une solution canonique particulière, soit encore d'explorer l'ensemble des solutions canoniques pour choisir la plus satisfaisante.[236]

Le problème de la mise en temps est abordé dans deux situations typiques: temps **lisse** et temps **strié** (*smooth/streaked time*) (voir chapitre VII §6). Une distinction supplémentaire est introduite entre **temps lisse mesuré** et **non mesuré**. Une extension des concepts de "lisse" et "strié" aux objets sonores eux-mêmes, inspirée d'une proposition de Duthen et Stroppa[237], est proposée.

Il est clair que, même si la méthode proposée ici permet de s'affranchir de la rigidité du temps métronomique, elle ne permet pas pour autant de modéliser des "règles d'interprétation" du type de celles qui visent à caractériser l'exécution du *rubato* en musique classique,[238] ou encore la notion de *swing* en jazz. Les programmes de bas niveau théorique masquent en général cette rigidité en introduisant des dispersions sur les points ou les modes d'attaque (exemple: la procédure "*humanize*" des séquenceurs). Or les musiciens savent que ces subtilités d'exécution ne sont pas totalement arbitraires mais qu'elles obéissent à des règles que l'on ne sait formuler que très partiellement. Notre approche consiste à déterminer ces paramètres de date et de durée à partir de contraintes imposées par les objets eux-mêmes et leur contexte.

1. Prototypes d'objets sonores (*sound-object prototypes*)

Soit un ensemble fini $A = \{A_1,\ldots,A_N\}$ de messages (ou, de manière équivalente, d'actions élémentaires) et soit $\mathcal{P}(A)$ l'ensemble des parties de A. Nous appelons **prototype d'objet sonore** une application

$$Ep: \quad R \cup \{nil\} \longrightarrow \mathcal{P}(A)$$

(où R est l'ensemble des nombres réels) telle que $Ep(nil) = \emptyset$ et $Ep(t) \neq \emptyset$ pour un ensemble fini de valeurs de *t*. L'utilité de la valeur "nil" est montrée plus loin (voir *infra*: troncature en début). Nous appelons **bornes** du prototype les nombres *tmin* et *tmax* tels que:

$$Ep(tmin) \neq \emptyset , \quad Ep(tmax) \neq \emptyset , \text{ et } \forall t \notin [tmin,tmax], \; Ep(t) = \emptyset .$$

Informellement, Ep(tmin) contient le premier message de l'objet et Ep(tmax) son dernier message.

Nous supposons connu un ensemble fini de prototypes d'objets sonores $\{Ep_j : j = 0,\ldots,jmax\}$. Par convention, Ep_0 désigne le prototype d'objet vide, que nous avons noté "_".[239] A chaque prototype Ep_j (pour $j \geq 1$) est associé un ensemble de paramètres et de propriétés que nous précisons plus loin (§4-5-6).

---

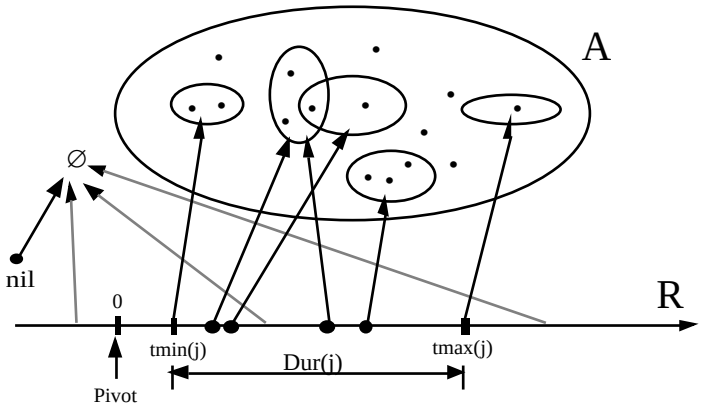
[236] Le premier mode d'utilisation correspond à des situations d'improvisation alors que le second est plus adapté au travail de composition.

[237] 1990

[238] Voir par exemple Katayose et al. 1990.

[239] E_{p0} ne doit pas être confondu avec les objets de durée nulle, pour lesquels tmin = tmax.

La figure ci-dessous représente un prototype d'objet sonore et l'application Ep_j correspondante.



La **durée d'un prototype d'objet sonore** est:

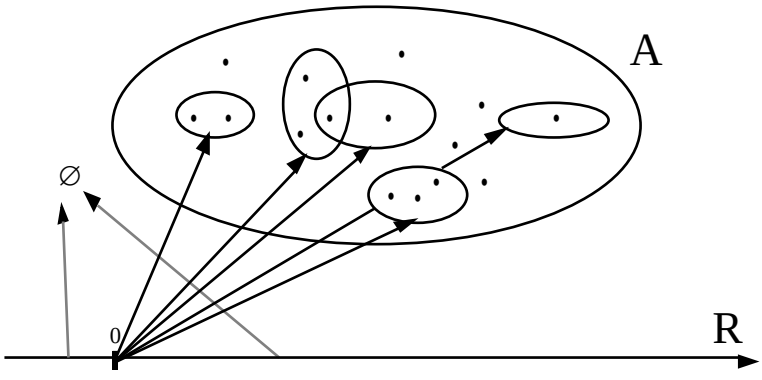
$$Dur(j) = tmax(j) - tmin(j) .$$

A tout prototype d'objet sonore Ep_j on peut associer un **prototype d'objet atemporel** $E'p_j$ défini par:

$$E'p_j(0) = \bigcup_{t \in [tmin,tmax]} Ep_j(t)$$

$$E'p_j(t) = \emptyset \quad \forall t \neq 0$$

Informellement, l'objet atemporel est un objet “aplati” concentrant tous ses messages à la date zéro. Au prototype d'objet sonore représenté ci-dessus on peut associer le prototype d'objet atemporel suivant:



2. Codage des structures polymétriques (*encoding polymetric structures*)

On a vu précédemment que toute formule polymétrique complète pouvait se représenter comme un tableau à deux dimensions. Par exemple, la formule

/3 ab { cde, ab } cd

est interprétée

/3 ab { /3 cde, /2 ab } /3 cd

soit, en représentation dilatée

/6 a _ b _ { c_ d _ e _ , a _ _ b _ _ } c _ d _

et sous forme de tableau (avec un facteur d'échelle Ratio = 6)

a	_	b	_	a	_	_	b	_	_	c	_	d	_	NIL
_	_	_	_	c	_	d	_	e	_	NIL	_	_	_	_

où NIL désigne la fin de chaque séquence (Voir algorithme au chapitre VIII).

2.1 Cas des objets temporels (*case of time-objects*)

Les symboles “a”, “b”, …, dans l'expression précédente, désignent des **objets sonores** E_k, où *k* est un index arbitraire ($k \geq -1$). Le symbole “_” indique un **objet sonore vide** unique E_0 (i.e. la prolongation de l'objet non vide qui le précède dans la séquence). “NIL” est l'objet fictif E_{-1} qui sert à marquer la fin de chaque séquence. Les objets sonores non vides ($k > 0$) sont des **instances** des prototypes d'objets définis précédemment.

Désignons par *i* le rang de l'objet E_k dans la séquence (en tenant compte des objets vides). Le rang de la deuxième instance de “b” dans la séquence de la première ligne est par exemple 8. Notons de même *inext* le rang du prochain objet non vide. Pour le même objet “b” de rang 8 on trouve: inext = 11. Nous appelons **durée symbolique** de l'objet sonore E_k l'expression:[240]

$$d(k) = \frac{\text{inext} - i}{\text{Ratio}}$$

Pour calculer les dates des messages des objets sonores dans chaque séquence, on dispose d'une suite (croissante[241]) de dates {T(i): i = 1,…,imax} telle que:

$$\forall i \in [1,\text{imax}], \quad T(i) = \Phi.\theta.\rho^{-1}(i) \quad \text{(temps strié)}$$

ou T(i) = 0 dans le cas où Φ est inconnue (temps lisse)

Les fonctions θ et ρ sont respectivement la structure rythmique et l'indexage des objets dans la séquence.

---

[240] Comparer cette définition avec celle de la durée symbolique d'un objet temporel, chapitre VII §5.

[241] dans le cas seulement où Φ est croissante (hypothèse de Jaffe, 1985).

Nous appelons T(i) la **position de référence** des objets de rang *i* dans la structure.[242] Cette position est l'abscisse de la **strie de référence** de l'objet sur l'axe du temps physique. Connaissant la représentation symbolique de la structure (formule polymétrique complète), les positions de références T(i) et certaines propriétés des objets, on est en mesure de calculer divers **paramètres d'exécution** (voir infra) qui permettent de déterminer la date de chaque message. Ces paramètres d'exécution sont regroupés dans une **table des instances**, qui contient par ailleurs un pointeur permettant d'identifier l'objet. La table des instances de l'exemple proposé peut s'écrire:

Table des instances

k	1	2	3	4	5	6	7	8	9
j = Obj(k)	1	2	1	2	3	4	3	4	5
d(k)	1/3	1/3	1/2	1/2	1/3	1/3	1/3	1/3	1/3
T(k), etc…	…	…	…						

où sont indiquées les durées symboliques d(k) des objets. *k* est l'index de l'instance E_k, et j = Obj(k) est un pointeur permettant d'identifier le prototype de l'objet dans la **table des symboles**:

Table des symboles

j	1	2	3	4	5
symbole	a	b	c	d	e

Pour coder la structure polymétrique complète il suffit donc de disposer les index des objets E_k dans un **tableau des phases** Seq(nseq,i):

Tableau des phases *Seq(nseq,i)*

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
nseq = 1	1	0	2	0	3	0	0	4	0	0	5	0	6	0	-1
nseq = 2	0	0	0	0	7	0	8	0	9	0	-1	0	0	0	0

Le tableau d'entiers {Seq(nseq,i): nseq = 1,nmax; i = 1,…,imax} décrit une **instance de structure polymétrique** dont E_k est le k-ième **objet sonore**, avec k = Seq(nseq,i).

[242] L'ordre des objets sonores est un ordre *strict* sur leurs dates symboliques (sauf pour les objets de même rang). Les coïncidences sont respectées dans la mesure où la structure du temps est identique pour toutes les séquences. On peut illustrer cette contrainte par l'image d'exécutants qui interprèteraient chacun une séquence sous la direction d'un chef d'orchestre. Le décalage local d'un objet correspond au jeu *rubato*, tandis qu'un décalage global des stries correspond à une rupture de tempo ordonnée par le chef d'orchestre.

L'objet E_k (pour $k \geq 0$) est une **instance** du prototype d'objet sonore Ep_j avec j = Obj(k). Les instances sont calculables si et seulement si:

$$\forall i \in [1,imax], \forall nseq \in [1,nmax],$$
$$Obj(k) \in [0,jmax] \text{ avec } k = Seq(nseq,i).$$

Par ailleurs, pour tout objet vide on a

$$k = Seq(nseq,i) = 0$$

sachant que:

$$Obj(0) = 0, \text{ et } Ep_0(t) = \varnothing \quad \forall t \in R$$

2.2 Cas des objets atemporels (*out-time objects*)

Soient <<f>> et <<g>> deux objets atemporels (voir chapitre VII §6) et soit la formule

/3 ab { c <<f>> de, a <<g>> <<f>> b } cd

qui est interprétée

/3 ab { /3 c <<f>> de, /2 a <<g>> <<f>> b } /3 cd

soit, en représentation dilatée:

/6 a _ b _ { c_ <<f>> d _ e _ , a _ _ <<g>> <<f>> b _ _ } c _ d _

La durée symbolique d(k) des objets atemporels est donc nulle.

Les tables deviennent par exemple:

Table des symboles

j	1	2	3	4	5	6	7
symbole	a	b	c	d	e	f	g

Table des instances

k	1	2	3	4	5	6	7	8	9	10	11	12
j = Obj(k)	1	2	1	2	3	4	3	4	5	6	7	6
d(k)	1/3	1/3	1/2	1/2	1/3	1/3	1/3	1/3	1/3	0	0	0
T(k), etc…	…	…	…									

Tableau des phases *Seq(nseq,i)*

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
nseq = 1	1	0	2	0	3	0	0	4	0	0	5	0	6	0	-1
nseq = 2	0	0	0	0	7	0	8	0	9	0	-1	0	0	0	0
nseq = 3	0	0	0	0	0	0	**10**	**11**	-1	0	0	0	0	0	0
nseq = 4	0	0	0	0	0	0	0	**12**	-1	0	0	0	0	0	0

Remarque

Dans une implémentation informatique il est important de minimiser le nombre de lignes du tableau Seq(). Ce problème n'est pas traité ici.

3. Paramètres d'exécution (*performance parameters*)

Chaque prototype d'objet sonore Ep_j (pour $j \neq 0$ avec $j = Obj(k)$) est susceptible, lors de son instanciation dans une séquence, de subir une **dilatation** (ou contraction) $\alpha(k)$ (***time-scale ratio***) ainsi qu'un **décalage local** $\delta(k)$ (***local drift***) par rapport à sa position de référence T(i). A ce décalage peut s'ajouter un **décalage global** $\Delta(i)$ (***global drift***) toujours positif qui affecte tous les objets de rang supérieur ou égal à *i* dans la séquence. Nous appelons ces nombres les **paramètres d'exécution** de l'objet.

Nous appelons **mise en temps** de la séquence d'objets d'indice *nseq* la fonction

$$f: \quad R \longrightarrow \mathcal{P}(A)$$

telle que

$\forall t \in R,$

$$f(t) = \left(\bigcup_{\substack{i = 1, imax \\ nseq = 1, nmax}} \left[Ep_j(LocalTime_k(t)) \right] \right) \cup \left(\bigcup_{\substack{i = 1, imax \\ nseq = 1, nmax}} \left[E'p_j(LocalTime'_k(t)) \right] \right)$$

avec:

$j = Obj(k)$ et $k = Seq(nseq,i)$.

Le premier membre de f(t) représente l'union des **objets sonores** (i.e. E_k avec $d(k) > 0$), pour lesquels le **temps local** est

$$LocalTime_k(t) = \frac{t - T(i) - \Delta(i) - \delta(k)}{\alpha(k)}$$

sachant qu'on a toujours $\alpha(k) \neq 0$ pour des objets sonores.

Le temps local est ici une fonction affine du temps absolu *t.* Nous proposons plus loin (§6.1) une définition plus générale de *f* et de $\text{LocalTime}_k()$ tenant compte des déformations des objets.

Le second membre de f(t) représente l'union des **objets atemporels** (i.e. E_k avec d(k) = 0), pour lesquels le temps local est:

$$\text{LocalTime}'_k(t) = t - T(i) - \Delta(i) - \delta(k) .$$

Rappelons que tous les messages d'un objet atemporel E_k sont “concentrés” au temps local zéro et seront donc envoyés à la date:

$$t = T(i) + \Delta(i) + \delta(k) .$$

Mettre en temps (ou **instancier**) la structure polymétrique revient à déterminer les coefficients $\alpha(k)$, $\delta(k)$, $\Delta(i)$ ou (et), si nécessaire, les fonctions $\text{LocalTime}_k(t)$ pour i = 1,…,imax et pour nseq = 1,…,nmax.

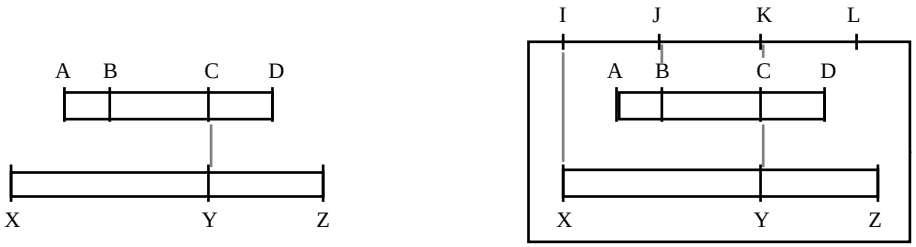
4. Propriétés métriques des objets non vides (*metrical properties of non-empty objects*)

4.1 Pivot temporel (*time pivot*)

Un modèle trivial de séquence consiste à placer “bout à bout” les intervalles supports temporels des objets. Dans ce cas, la structure rythmique de la séquence dépend des durées physiques (tmax-tmin) des objets. Une formulation plus intéressante a été proposée par Duthen et Stroppa[243]. Les auteurs partent de l'observation que tout objet sonore comporte un ou plusieurs points

> musicalement ou perceptuellement plus pertinents que d'autres (par exemple, le temps d'attaque d'une note percussive, divers accents, un climax d'une figure…).

Chaque point particulier est appelé un **pivot temporel** de l'objet sonore. Lorsqu'on construit une hiérarchie d'objets sonores on met en correspondance certains pivots selon des règles définies à l'avance. L'objet de niveau supérieur possède des pivots déduits de ses composants, par coïncidence, interpolation ou tout autre mécanisme d'inférence, par exemple:



où A, B, C, D, X, Y et Z sont les pivots de deux objets, avec une règle mettant en coïncidence C et Y, et d'autres règles assignant les pivots I, J, K, L à l'objet composite.

---

[243] 1990. Cette méthode, mise en œuvre manuellement lors du mixage de *Traiettoria*, est en cours d'implémentation à l'Institut de Recherche et Coordination Acoustique-Musique (IRCAM, Paris).

L'intérêt de la méthode de synchronisation de pivots est indubitable mais son domaine d'application est limité à la construction *ascendante* de structures (musicales). A l'opposé, la génération de structures par des grammaires formelles obéit à une stratégie *descendante,* dont l'instanciation des objets sonores constitue la dernière étape. Dans ce cas, la synchronisation est gérée au niveau du temps symbolique et non au niveau des pivots.

Nous associons à tout objet sonore non vide un pivot unique défini informellement ainsi: *en temps strié, le pivot doit être placé à la position de référence T(i) de l'objet.* Un objet sonore muni d'un tel pivot est appelé un **objet strié**. Certains objets (les **objets lisses**) n'ont pas de pivot.

Un objet peut être déplacé par rapport à sa position de référence s'il possède une propriété que nous notons “Reloc” (voir infra). Lorsque l'objet est lisse on peut toujours assigner un pivot arbitraire (par exemple au début de l'intervalle support temporel) et déclarer la propriété Reloc.

Les propriétés ci-dessous (liste non limitative) sont celles qui apparaissent les plus pertinentes pour l'agencement d'objets striés.

Nous désignons par Dur(j) la **durée du prototype d'objet sonore** Ep_j (voir §1).

Pivot en début (PivBeg), fin (PivEnd)

Le pivot coïncide avec le premier (resp. dernier) message. On a donc:

Cas PivBeg: tmin(j) = 0 et tmax(j) = Dur(j)
Cas PivEnd: tmin(j) = -Dur(j) et tmax(j) = 0

Pivot en début (PivBegOn), fin (PivEndOff) type ON/OFF

Parmi les messages du prototype d'objet sonore Ep_j, certains sont des déclenchements (type ON) et d'autres des fins (type OFF) de processus. Il est fréquent qu'on ait à localiser un objet sur le premier message de type ON (propriété PivBegOn), ou encore le dernier de type OFF (propriété PivEndOff).

Si t1(j) et t2(j) sont les dates du premier (resp. dernier) message de type ON (resp. OFF) par rapport au premier message du prototype, on a:

Cas PivBegOn: tmin(j) = -t1(j) et tmax(j) = Dur(j) - t1(j)
Cas PivEndOff: tmin(j) = -t2(j) et tmax(j) = Dur(j) - t2(j)

Pivot centré (PivCent)

Le pivot coïncide avec le milieu de l'intervalle support temporel du prototype d'objet sonore Ep_j, donc:

$$tmin(j) = -tmax(j) = -Dur(j)/2$$

Pivot centré type ON/OFF (PivCentOnOff)

Le pivot coïncide avec le milieu de l'intervalle support temporel de la partie de l'objet qui débute sur la première action de type ON et se termine sur la dernière de type OFF. On a donc:

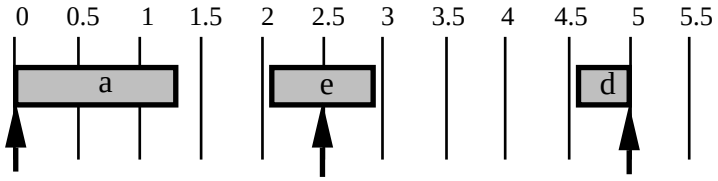
$$tmin(j) = -\frac{t1(j)+t2(j)}{2} \qquad tmax(j) = D(j) - \frac{t1(j)+t2(j)}{2}$$

Pivot spécifié (PivSpec)

Le pivot est à une date t0 quelconque par rapport au premier message de l'objet. On a dans ce cas:

$$tmin(j) = - t0 \text{ et } tmax(j) = Dur(j) - t0$$

Dans les exemples ci-dessous nous représentons les intervalles supports temporels des objets non vides par des rectangles sur une échelle de temps graduée. Les largeurs et positions verticales des rectangles sont sans signification. Le pivot de chaque objet est indiqué par une flèche. La figure ci-dessous représente, sur une structure du temps métronomique, trois objets “a”, “e”, “d” de propriétés PivBeg, PivCent et PivEnd, et de durées 1.3s, 0.8s et 0.4s respectivement:



4.2 Dilatation / contraction (*time-scale ratio*)

Les propriétés suivantes permettent de déterminer les bornes acceptables de $\alpha(k)$.[244]

Elasticité (FixScale, OkRescale, OkExpand, OkCompress)

$\alpha(k)$ arbitraire: élastique (OkRescale)
$\alpha(k) = 1$: figé (FixScale)
$\alpha(k) \geq 1$: expansible (OkExpand)
$\alpha(k) \leq 1$: contractable (OkCompress)

4.3 Décalage

Ces propriétés permettent de déterminer $\delta(k)$ et $\Delta(i)$.

Objet décalable (Reloc)

Un objet est **décalable** si l'on peut (pour respecter d'autres contraintes) retarder ou avancer son déclenchement, i.e. $\delta(k) \neq 0$. Dans la terminologie musicale classique occidentale, un objet de propriété Reloc peut faire partie d'une séquence interprétée *rubato.*

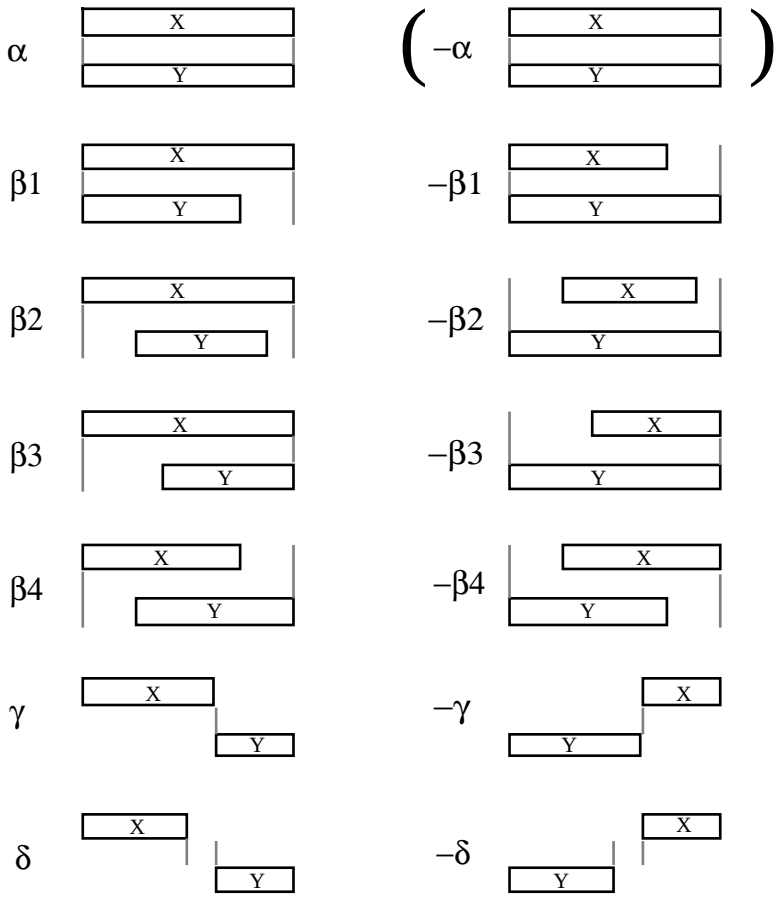
Rupture de tempo (BrkTempo)

Il y a **rupture de tempo** sur un objet E_k (avec k = Seq(nseq,i) et k > 0) lorsque les stries postérieures à sa position de référence peuvent être retardées, i.e. $\Delta(ii) > \Delta(i)$ pour tout ii > i. Un objet de propriété BrkTempo peut produire un effet de *point d'orgue.*

[244] Il est clair que certains objets sonores ne peuvent être dilatés ou contractés sans être “dénaturés”. Plus on se rapproche de phénomènes continus et plus on modifie, par dilatation ou contraction, des paramètres “hors temps” tels que hauteur, timbre, grain, etc.

5. Propriétés topologiques des objets non vides (*topological properties of non-empty objects*)

Les positions relatives des intervalles supports temporels de deux objets non vides X et Y dans une séquence correspondent à quatorze configurations topologiques représentées graphiquement ci-dessous:



Les dispositions α et -α étant équivalentes, on est ramené à treize configurations distinctes. Les symboles α, β1, β2, etc., sont empruntés à Vecchione[245]. Une représentation similaire a été proposée par Allen & Kautz[246], Oppo[247] et Van Benthem[248].

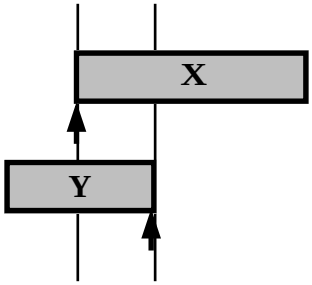
[245] 1984, pp.150-ff. Vecchione utilise ces symboles pour des opérateurs, et non de simples prédicats temporels. Voir Bel 1989a, pp.124-9.

[246] 1985

[247] 1984

[248] 1983, pp.58-79.

Il semblerait en première analyse que l'énoncé “X précède Y” impose une configuration α, $\beta1$, $\beta2$, $\beta3$, $\beta4$, γ ou δ. Cette restriction n'est vraie que lorsque les objets sont interprétés en temps lisse. Dans le cas du temps strié, le pivot peut avoir une position quelconque par rapport à l'intervalle support temporel, par conséquent il faut aussi envisager $-\beta1$, $-\beta2$, $-\beta3$, $-\beta4$, comme par exemple dans la séquence “XY” ci-dessous



où les objets X et Y possèdent les propriétés PivBeg et PivEnd respectivement. Autrement dit, *la notion de séquence est fondée sur une relation d'ordre strict sur les dates symboliques des objets, et non sur la topologie de leurs intervalles supports temporels.*

Nous appelons **séquence stricte**[249] tout agencement d'intervalles temporels X_1, …, X_{nmax} tel que:

$$\forall n \in [1,nmax-1], \quad \gamma(X_n,X_{n+1})$$

Pour définir les propriétés topologiques des objets non vides nous regroupons les treize configurations en trois classes:

Recouvrement: $\{\alpha, -\alpha, \beta1, -\beta1, \beta2, -\beta2, \beta3, -\beta3, \beta4, -\beta4\}$

Continuité: $\{\gamma, -\gamma\}$

Discontinuité: $\{\delta, -\delta\}$

Recouvrement en début (OverBeg), fin (OverEnd)

Un objet qui possède la propriété de **recouvrement en début** peut débuter avant que le(s) précédent(s) dans la séquence ne soi(en)t terminé(s). Symétriquement, la propriété de **recouvrement en fin** permet de terminer un objet après avoir déclenché le(s) suivant(s).

Formellement,

non OverBeg(X_i) => $\forall j \in [1,iprec]$, $\gamma(X_j,X_i)$ ou $\delta(X_j,X_i)$

non OverEnd(X_i) => $\forall j \in [inext,imax]$, $\gamma(X_i,X_j)$ ou $\delta(X_i,X_j)$

où *iprec* est l'indice de l'objet précédent non vide, et *inext* celui de l'objet suivant non vide.

[249] Ne pas confondre les index des intervalles supports temporels avec ceux des objets placés sur ces intervalles: l'ordre de la séquence stricte X_1, …, X_{nmax} n'est pas nécessairement celui de la séquence des objets correspondants.

Continuité au début (ContBeg), en fin (ContEnd)

Un objet est **continu au début** (resp. **en fin**) si son intervalle support temporel doit être contigu à celui d'un objet précédent (resp. suivant) dans la séquence.

Formellement,

$ContBeg(X_i) \Rightarrow \exists j \in [1,iprec],\ non\ \delta(X_j,X_i)$

$ContEnd(X_i) \Rightarrow \exists j \in [inext,imax],\ non\ \delta(X_i,X_j)$

où *iprec* est l'indice de l'objet précédent non vide, et *inext* celui de l'objet suivant non vide.

6. Déformations d'objets non vides (*stretching non-empty objects*)

Des propriétés peuvent être définies arbitrairement pour modifier la nature ou les dates des messages des objets en fonction du contexte. Nous nous intéressons ici à une transformation qui permet de résoudre les contraintes induites par les propriétés précédentes sans modifier l'échelle du temps de l'objet (le paramètre $\alpha(k)$).

6.1 Troncature en début (TruncBeg) ou en fin (TruncEnd) (*Truncating the beginning [TruncBeg] or the end [TruncEnd]*)

Cette propriété permet de supprimer la partie commençante ou(et) la partie finissante de l'objet. La suppression est opérée de la manière suivante:

Soient t1(k) et t2(k) les dates réelles auxquelles doit commencer et finir l'objet non vide E_k. Ces dates correspondent, soit aux “extrémités normales” de l'objet, soit à celles de l'objet tronqué. La structure f(t) est maintenant définie ainsi:

$\forall t \in R,$

$$f(t) = \left(\bigcup_{\substack{i = 1,imax \\ nseq = 1,nmax}} \left[Ep_j(LocalTime_k(t)) \right] \right) \cup \left(\bigcup_{\substack{i = 1,imax \\ nseq = 1,nmax}} \left[E'p_j(LocalTime'_k(t)) \right] \right)$$

avec j = Obj(k) et k = Seq(nseq,i)

sachant que:

(1) $\forall t \in [t1(k),t2(k)],$

$$LocalTime_k(t) = \frac{t - T(i) - \Delta(i) - \delta(k)}{\alpha(k)}$$

avec j = Obj(k) et k = Seq(nseq,i) ;

(2) $\forall t < t1(k),$

(2.1) $LocalTime_k(t) = nil$ si $Ep_j(LocalTime_k(t))$ est de type ON ;

(2.2) $LocalTime_k(t) = LocalTime_k(t1(k))$ sinon ;

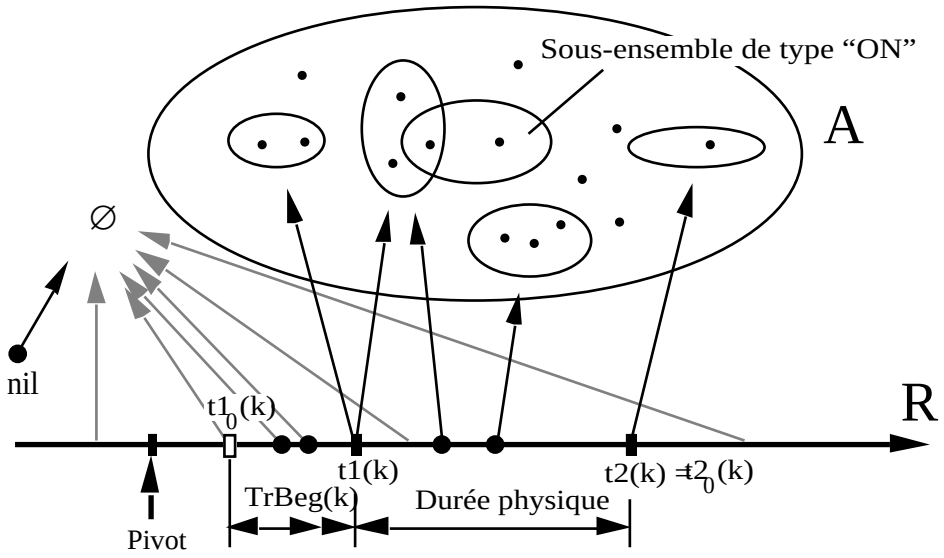
(3) $\forall t > t2(k),$

(3.1) $LocalTime_k(t) = nil$ si $Ep_j(LocalTime_k(t))$ est de type ON ;

(3.2) $LocalTime_k(t) = LocalTime_k(t2(k))$ sinon.

Informellement, tronquer un objet en début (resp. en fin) revient à supprimer tous les messages de type ON dans la partie tronquée, et à envoyer tous les autres messages de la partie tronquée à la date t1(k) (resp.t2(k)).

La figure ci-dessous représente un objet sonore tronqué en début:



Pour savoir si l'objet est tronqué en début, on calcule la date physique $t1_0(k)$ de son déclenchement “normal”: le temps local de l'objet est alors tmin(j) et on a

$$tmin(j) = \frac{t1_0(k) - T(i) - D(i) - d(k)}{\alpha(k)}$$

ce qui donne:

$$t1_0(k) = \alpha(k).tmin(j) + T(i) + \Delta(i) + \delta(k))$$

On note $TrBeg(k) = t1(k) - t1_0(k)$. L'objet est tronqué si $TrBeg(k) > 0$.

On procède de manière similaire pour la troncature en fin. Si $t2_0(k)$ est la date de fin de l'objet non tronqué, on calcule $TrEnd(k) = t2_0(k) - t2(k)$. L'objet est tronqué si $TrEnd(k) > 0$.

7. Calcul du coefficient de dilatation/contraction (*calculating the time-scale ratio*)

La dilatation ou contraction $\alpha(k)$ d'un objet E_k dépend de sa durée symbolique, de sa nature (lisse ou striée) ainsi que de la structure du temps (lisse ou strié). Si Dur(j) désigne la durée du prototype d'objet sonore Ep_j (avec j = Obj(k) et k = Seq(nseq,i)), la durée (physique) de l'objet non vide E_k est $\alpha(k).Dur(j)$.

En temps strié on dispose des dates des stries de référence T(i) ainsi que, le cas échéant, des valeurs de décalage global des stries $\Delta(i)$. Pour les objets striés on dispose d'autre part de la période de référence Tref(j) du prototype Ep_j.

Dans tous les cas, il faut trouver une expression de $\alpha(k)$ qui tienne compte des propriétés métriques de l'objet et de la nature du temps (lisse ou strié). On vérifie ensuite que la valeur de $\alpha(k)$ pour un objet est compatible avec ses propriétés d'élasticité. Dans le cas contraire on prend $\alpha(k) = 1$.

Des exemples de calcul sont proposés en annexe 5.

7.1 Calcul de $\alpha(k)$ en temps lisse (*computing $\alpha(k)$ in smooth time*)

En temps lisse il peut exister une *horloge* destinée, non à positionner les objets sonores, mais à ajuster leurs durées. Nous convenons dans ce cas de parler de **temps lisse mesuré**. Appelons *Tclock* la période de l'horloge.

Lorsque l'horloge est absente (**temps lisse non mesuré**), nous prenons:

$$\alpha(k) = d(k) \ .$$

La table ci-dessous donne les formules utilisées pour le calcul de $\alpha(k)$ en temps lisse mesuré:

Type d'objet	Tref(j)	Dur(j)	$\alpha(k)$
strié	> 0	= 0	d(k) . Tclock / Tref(j)
strié	> 0	> 0	d(k) . Tclock / Tref(j)
lisse	= 0	> 0	d(k) . Tclock / Dur(j)
lisse	= 0	= 0	d(k)

Pour les silences (les objets non vides notés conventionnellement "-"), Tref(j) et Dur(j) sont indéterminés. Le calcul de $\alpha(k)$ n'est donc pas possible. Soit *iprec* le rang de l'objet non vide précédent et *kprec* son indice. Sa durée symbolique est d(kprec) et sa durée physique t2(kprec) - t1(kprec). Nous appelons **période locale** P le rapport:

$$P = \frac{t2(kprec) - t1(kprec)}{d(kprec)}$$

La durée physique du silence d'indice *k* est donc P. d(k).

Un message d'erreur est envoyé si une séquence d'objets en temps lisse débute par un silence.

7.2 Calcul de $\alpha(k)$ en temps strié (*computing $\alpha(k)$ in streaked time*)

7.2.1 Objets striés (*streaked objects*)

La valeur proposée de $\alpha(k)$ est:

$$\alpha(k) = \frac{T(inext) - T(i) + \Delta(inext) - \Delta(i)}{Tref(j)}, \qquad \text{avec } j = Event(k) \text{ et } k = Seq(ligne, i)$$

$$\text{si } d(k) > 0,$$

ou

$$\alpha(k) = 0 \quad \text{si } d(k) = 0 \quad \text{(Cas des objets atemporels).}$$

Intuitivement, il s'agit d'ajuster le tempo du métronome de référence à celui de l'exécution de l'objet dans la séquence.

7.2.2 Objets lisses (*smooth objects*)

Pour calculer $\alpha(k)$ on remplace Tref(j) par Dur(j). On obtient:

$$\alpha(k) = \frac{T(inext) - T(i) + \Delta(inext) - \Delta(i)}{Dur(j)}$$

si $d(k) > 0$,

ou

$\alpha(k) = 0$ si $d(k) = 0$ (Cas des objets atemporels).

La durée de l'objet lisse E_k est alors

$$\alpha(k).Dur(j) = T(inext) - T(i) + \Delta(inext) - \Delta(i)$$

pour un objet sonore, et

$$\alpha(k).Dur(j) = 0$$

pour un objet atemporel.

8. Procédure de mise en temps (*time-setting procedure*)

Une procédure permettant l'instanciation des objets temporels dans une structure polymétrique, tenant compte des propriétés topologiques des objets, est proposée ici. On montre que la procédure est algorithmique et que sa complexité en fonction du nombre d'objets est polynomiale.

La procédure complète est donnée en annexe 6. Des exemples de mise en temps de séquences et de formules polymétriques sont proposés dans les annexes 5.3 et 5.4 respectivement.

8.1 Données (*data*)

La structure polymétrique est décrite par le tableau Seq(nseq,i), où *i* désigne le rang de l'objet sonore E_k dans la séquence *nseq*, ainsi que le tableau Obj(k). Pour tout k = Seq(nseq,i), le pointeur j = Obj(k) désigne le prototype d'objet sonore Ep_j.

La structure du temps est donnée par l'ensemble des positions de référence T(i). Dans le cas où le temps est lisse, on pose initialement T(i) = 0 pour tout *i*.

8.2 Boucle principale (*main loop*)

Procédure Mise_en_temps

```
Variables globales:
  NMAX = nombre maximal de séquences
  IMAX = longueur maximale d'une séquence
  KMAX = nombre maximum d'objets dans la structure
  Seq[NMAX][IMAX],Δ[IMAX],T[IMAX],
  Obj[KMAX],t1[KMAX],t2[KMAX],t"1[KMAX],t"2[KMAX],TrBeg[KMAX],TrEnd[KMAX]
    , α[KMAX],δ[KMAX]

Entrée:
  Seq[][],Obj[],T[]

Sortie:
  T[],t1[],t2[],TrBeg[],TrEnd[],α[],δ[]

Début
  Pour (i = 1; i ≤ IMAX)
  |    Δ[i] <— 0;
  |    i <— i + 1;
  Finpour
  Calcul_de_alpha;

Reprise:
  Pour (nseq = 1; nseq ≤ NMAX)
  |    Placer(nseq); /* Calcul de t1[], t2[] */
  |    Si(Positionner(nseq,nature_temps) = échec)
  |    alors
  |    |    Afficher "Mise en temps impossible";
  |    |    /* Ici on peut aussi relancer l'algorithme en relâchant
  |    |       toutes les contraintes ContBeg, OverBeg ou OverEnd. */
  |    |    Stop;
  |    Finsi
  |    Si((nature_temps = lisse) et (nseq = 1))
  |    alors
  |    |    Interpolation_des_stries;
  |    Finsi
  |    Si(nseq = 1)
  |    alors
  |    |    Pour (i = 1; i ≤ IMAX)
  |    |    |    T[i] <— T[i] + Δ[i];
  |    |    |    Δ[i] <— 0;
  |    |    |    i <— i + 1;
  |    |    Finpour
  |    sinon
  |    |    BTflag <— faux;
  |    |    Pour (i = 1; i ≤ IMAX)
  |    |    |    T[i] <— T[i] + Δ[i];
  |    |    |    Si (Δ[i] ≠ 0)
  |    |    |    alors
  |    |    |    |    BTflag <— vrai;
  |    |    |    Finsi
  |    |    |    i <— i + 1;
  |    |    Finpour
  |    |    Si (BTflag) aller à Reprise;
  |    Finsi
  |    nseq <— nseq + 1;
  Finpour
Fin
```

Les décalages globaux Δ(i) sont initialisés à 0, et les valeurs de α(k) sont calculées pour tous les objets (voir formules au §8).

On effectue la mise en temps dans l'ordre des séquences de la structure polymétrique: nseq = 1, NMAX. Pour chaque séquence on calcule d'abord les coordonnées “idéales” t1(k) et t2(k) de l'objet de rang *i* avec k = Seq(nseq,i), à l'aide de la procédure **Placer()**:

```
Procédure Placer(nseq)

Début
  Pour (i = 1, i ≤ IMAX)
  |    k <– Seq[nseq][i];
  |    t1[k] <– α[k].tmin[j] + T[i];
  |    t2[k] <– α[k].tmax[j] + T[i];
  |    i <– i + 1;
  Finpour
Fin
```

On appelle ensuite la fonction **Positionner()**, qui sera décrite plus loin. En cas de succès, il faut effectuer deux opérations avant de passer à la séquence suivante:

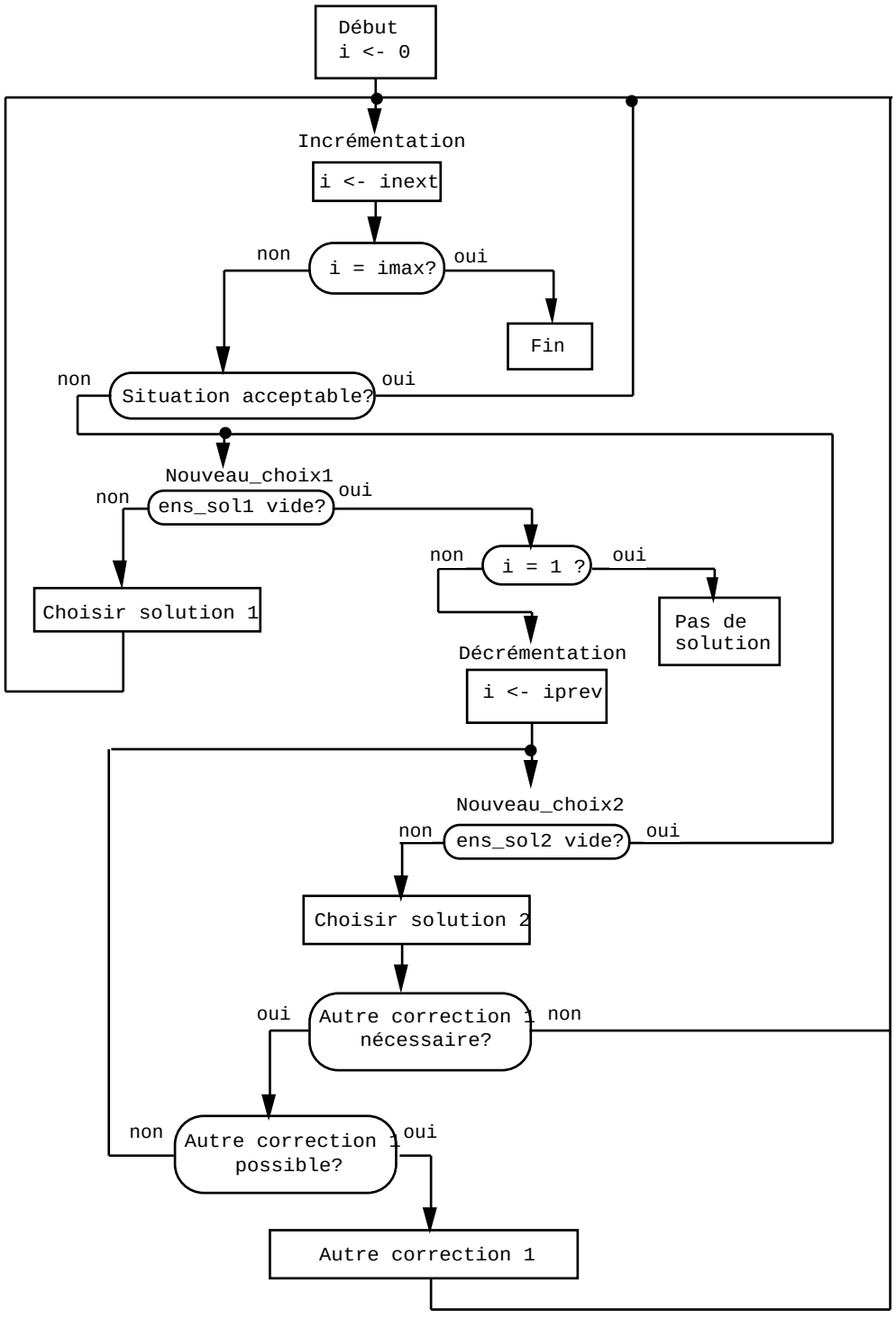
(1) Si le temps est lisse et que la séquence positionnée est la première, il est nécessaire de calculer les valeurs de Δ(i) pour les objets vides de rang *i*. Ce calcul se fait par interpolation (voir exemple en annexe 5.3.1).

(2) On met à jour les valeurs de T(i):

$$T(i) \leftarrow T(i) + \Delta(i)$$
$$\Delta(i) \leftarrow 0$$

Après le positionnement de toute séquence autre que la première, si $\Delta(i) \neq 0$ pour une valeur de *i*, une correction de type {Break tempo} a été sélectionnée (un “point d'orgue”). Cette correction entraîne un décalage global des stries qu'il faut prendre en compte pour toutes les séquences. La procédure de mise en temps est donc relancée avec les nouvelles valeurs de T(i), i.e. la nouvelle structure du temps.

8.3 Fonction Positionner(): organigramme (*"Positionner()" function: flowchart*)



8.4 Variables manipulées par la fonction Positionner() (*Variables used by the "Positionner()" function*)

```
Fonction Positionner(nseq,nature_temps)

Variables locales:
   t'1[IMAX],t'2[IMAX],δ1[IMAX],δ2[IMAX],
   shift1[IMAX],shift2[IMAX],Ts[IMAX],dΔ0[IMAX],dΔ1[IMAX],dΔ2[IMAX],
   ContEndPrev[IMAX],OverEndPrev[IMAX],Bt[IMAX]

Début
   Ts[0] <— t'2[0] <— -∞ ;
   dΔ0[0] <— dΔ1[0] <— dΔ2[0] <— δ1[0] <— δ2[0]<— 0;
   ContEndPrev[0] <— Bt[0] <— faux;
   OverEndPrev[0] <— vrai;
   i <— k <— iprev <— 0;
   backtrack <— faux;
   aller à Incrémentation;
```

Lorsque l'on positionne les objets sonores on est amené à modifier temporairement les valeurs de t1(k) et t2(k). Puisqu'on ne prend en compte, dans la fonction, que les objets d'une seule séquence, les valeurs temporaires de t1(k) et t2(k) sont conservées dans les tableaux t'1(i), t''1(i) et t'2(i), t''2(i) respectivement, avec k = Seq(i,nseq). Ces valeurs sont ensuite copiées dans t1(k) et t2(k) lorsque la configuration calculée par la fonction **Positionner()** a été acceptée (voir **Incrémentation**).

De la même manière, on utilise δ1(i) et δ2(i) pour les décalages locaux temporaires, et la valeur finale est: δ(k) = δ1(i) + δ2(i).

dΔ0(i) est la variation (temporaire) du décalage global Δ(i) de la strie de rang *i* lorsqu'on prend en compte les décalages globaux (temporaires) des stries précédentes.

dΔ1(i) et dΔ2(i) sont les variations (temporaires) du décalage global Δ(i) de la strie de rang *i* causées par les corrections de position de l'objet E_k, avec k = Seq(nseq,i).

Dans ce qui précède, on utilise l'indice 1 pour les variables qui sont affectées par la correction "à gauche" de l'objet (contraintes sur t1(k)), et 2 pour la correction "à droite". On utilise ces mêmes indices pour numéroter les ensembles solutions des deux types de correction.

t'1(i) est la nouvelle valeur de t1(k) lorsqu'on prend en compte la correction 1 (voir infra) et les décalages globaux. En général,

$$t'1(i) \leftarrow\!\!- t1(k) + \Delta(i) + d\Delta0(i) + shift1(i)$$

où shift1(i) représente la valeur de correction. t'2(i) est la nouvelle valeur de t2(k) dans les mêmes conditions.

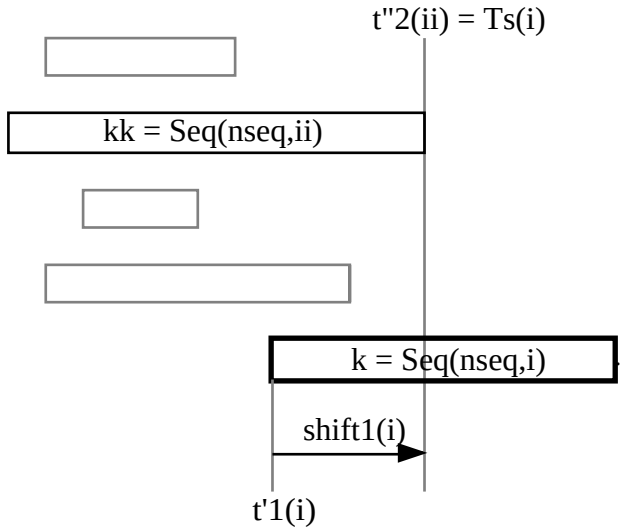
t''2(i) est la nouvelle valeur de t2(k) lorsqu'on prend en compte la correction 2 et la correction 1 (voir infra). En général,

$$t''2(i) \leftarrow\!\!- t'2(i) + shift2(i)$$

où shift2(i) représente la valeur de correction. t''1(i) est la nouvelle valeur de t1(k) dans les mêmes conditions.

TrBeg(k) et TrEnd(k) ont été définis au §6.1.

Pour tout objet E_k de rang *i*, on note Ts(i) la borne supérieure de t"2(ii) pour ii < i, où t"2(ii) est la valeur temporaire de t2(kk) pour un objet E_{kk} de la même séquence que E_k:



Pour positionner l'objet E_k il faut tenir compte des propriétés topologiques ContEnd(jj), OverEnd(jj) et BrkTempo(jj) de l'objet E_{kk}, avec jj = Obj(kk). On pose donc:

ContEndPrev(i) = ContEnd(jj) Continuité en fin de l'objet E_{kk}

OverEndPrev(i) = OverEnd(jj) Recouvrement de la fin de l'objet E_{kk}

Bt(i) = BrkTempo(jj) Possibilité de décalage global après l'objet E_{kk} (“rupture de tempo”).

8.5 Incrémentation (*incrementation*)

```
Incrémentation:
  dΔ0[i+1] <— dΔ0[i] + dΔ1[i] + dΔ2[i];
  /* Recherche du rang du prochain objet non vide */
  Pour (inext = i+1; i ≤ IMAX)
  |    dΔ0[inext] <— dΔ0[i+1];
  |    dΔ1[inext] <— dΔ2[inext] <— δ1[inext] <— δ2[inext] <— 0;
  |    Si (Seq[nseq][inext] > 0) interrompre Pour;
  |    inext <— inext + 1;
  Finpour
  /* Mise à jour de Ts */
  Si (Ts[i] < t"2[i])
  alors
  |    Ts[inext] <— t"2[i];
  |    ContEndPrev[inext] <— ContEnd[j]; OverEndPrev[inext] <—
  |      OverEnd[j]; Bt[inext] <— BrkTempo[j];
  sinon
  |    Ts[inext] <— Ts[i];
  Finsi
  iprev <— i;
  i <— inext;
  k <— Seq[nseq][i];
  Si (k = -1)   /* Fin de séquence “NIL” */
  alors
  |    Si (Solution_acceptée)
  |    alors
```

```
|    |    Pour (i = 1; i ≤ IMAX)
|    |    |    Δ[i] <— Δ[i] + dΔ0[i]+ dΔ1[i]+ dΔ2[i];
|    |    |    k <— Seq[nseq][i];
|    |    |    Si (k > 0)
|    |    |    alors
|    |    |    |    t1[k] <— t"1[i];
|    |    |    |    t2[k] <— t"2[i];
|    |    |    |    δ[k] <— δ1[i] + δ2[i];
|    |    |    Finsi
|    |    |    i <— i + 1;
|    |    Finpour
|    |    Retour(succès);
|    sinon
|    |    backtrack <— vrai;
|    |    aller à Décrémentation;
|    Finsi
Finsi
j <— Obj[k];
TrBeg[k] <— TrEnd[k] <— 0;
t"1[i] <— t'1[i] <— t1[k] + Δ[i] + dΔ0[i];
t"2[i] <— t'2[i] <— t2[k] + Δ[i] + dΔ0[i];
shift2[i] <— 0;
ens_sol2[i] <— vide;

shift1[i] <— Ts[i] - t"1[i];
Si (Situation(i,nseq,shift1[i],nature_temps,t"1[i],t"2[i]) =
   acceptable)
alors
|    aller à Incrémentation;
Finsi
ens_sol1[i] <—
   Choix_possibles(i,nseq,t'1[i],t'2[i],shift1[i],Ts[i],nature_temps,1)
   ;
Si (ens_sol1[i] = vide)
alors
|    Tsm <— Ts[i];
|    shift2[i] <— - shift1[i];
|    aller à Décrémentation;
Finsi
Aller à Nouveau_choix1;
```

En partant de l'objet fictif de rang 0 dans la séquence, ou d'un objet non vide de rang *i*, on cherche le rang *inext* du prochain objet non vide de la séquence. Si l'on parvient en fin de séquence (i.e. Seq(nseq,i) = -1), on valide les corrections (après, le cas échéant, un dialogue avec l'utilisateur), ou bien on revient en arrière (*backtracking*) pour rechercher la prochaine solution.

Pendant la recherche de *inext* on reporte sur les stries suivantes les décalages globaux des stries précédentes:

$$d\Delta 0(inext) \leftarrow\!\!- d\Delta 0(i) + d\Delta 1(i) + d\Delta 2(i)$$

et on initialise les décalages locaux $\delta 1(inext)$ et $\delta 2(inext)$. On affecte enfin à *i* la valeur *inext* trouvée.

Connaissant Ts(i) et t'1(i) on calcule:

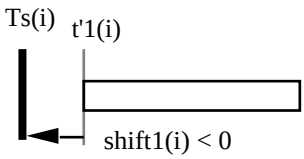
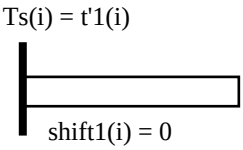
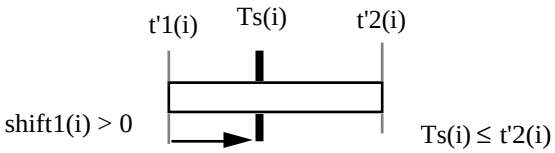
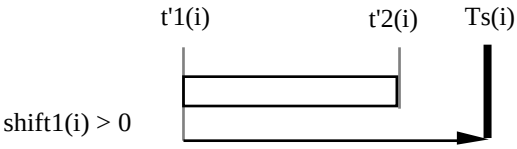
$$shift1(i) \leftarrow Ts(i) - t'1(i)$$

puis on appelle la fonction **Situation()**:

```
Fonction Situation(i,nseq,shift,nature_temps,t1,t2)

Début
  Si (i = 1) Retour(acceptable);
  k <— Seq[nseq][i];
  j <— Obj[k];
  Si ((nature_temps = lisse) et (nseq = 1)) Retour(inacceptable);
  Si ((shift < 0) et (non ContBeg[j]) et (non ContEndPrev[i]))
     Retour(acceptable);                                          /* 1 */
  Si (shift = 0)  Retour(acceptable);                             /* 2 */
  Si (shift > 0)
  alors
  |    Si ((Ts ≤ t2) et OverBeg[j] et OverEndPrev[i])
  |       Retour(acceptable);  /* 3 */
  |    Si ((Ts > t2) et OverBeg[j] et OverEnd[j] et OverEndPrev[i] et
  |       (non ContEnd[j])) Retour(acceptable);
  |                                                               /* 4 */
  Finsi
  Retour(inacceptable);
Fin
```

8.6 Situations

En temps strié ou pour nseq > 1, on trouve 4 situations possibles correspondant à diverses configurations de l'objet E_k par rapport à l'objet E_{kk}. Ces situations sont énumérées ci-dessous, ainsi que les contraintes sur les propriétés qui les rendent acceptables:

Nr	Configuration	Condition
1		(not ContBeg(j)) et (not ContEndPrev(i))
2		aucune
3		OverBeg(j) et OverEndPrev(i)
4		OverBeg(j) et OverEnd(j) et (not ContEnd(j)) et OverEndPrev(i)

Il est facile de voir qu'en temps lisse avec nseq = 1, puisque T(i) = 0, on aboutit toujours aux situations 3 ou 4, situations qui sont inacceptables puisque le recouvrement des objets de la première séquence n'est pas autorisé dans ce cas.

Situations: topologie des intervalles supports temporels

Nous avons introduit au §5 la notation de Vecchione pour les configurations topologiques d'intervalles. On peut facilement établir la liste des configurations possibles des intervalles supports temporels de E_{kk} et E_k. Notons $\Omega(E_{kk},E_k)$ le prédicat temporel qui décrit la configuration. Les valeurs possibles de Ω dans les 4 situations sont données ci-dessous:

Situation	Valeurs de Ω
1	δ
2	γ
3	$\beta 4, -\beta 1, -\beta 2$

4	$\alpha, -\alpha, \beta 1, \beta 2, \beta 3, -\beta 3, -\gamma, -\delta$

Nous appelons **correction canonique 1** la plus petite modification (en valeur absolue) de t'1(i) qui conduit à une situation acceptable.

Théorème IX.1

La correction canonique 1 est:

t'1(i) <— t'1(i) + shift1(i)

Preuve

Trivial pour la situation 2. Dans tous les autres cas, shift1(i) doit changer de signe ou s'annuler. Il est facile de voir que la plus petite modification est celle qui conduit à la situation numéro 2. ■

Voir la fonction **Situation(i,nseq,shift,t1,t2)** dans l'algorithme.

8.7 Ensemble solution 1 (*Solution set 1*)

On appelle la fonction

Choix_possibles(i,nseq,shift,t1,t2,Ts,nature_temps,numéro)

avec numéro = 1:

```
Fonction
  Choix_possibles(i,nseq,shift,t1,t2,Ts,nature_temps,numéro)

Début
  sol <— vide;
  k <— Seq[nseq][i];
  j <— Obj[k];
  Si ((nature_temps = lisse) et (nseq = 1))
  alors
  |    Retour({Break tempo});
  Finsi
  Si (Reloc[j] ou j = 1)
  alors
  |    sol <— sol ∪ {Shift object};
  Finsi
  Si ((shift > 0) et Bt[i])
  alors
  |    sol <— sol ∪ {Break tempo};
  Finsi
  Si ((numéro = 1) et (shift > 0) et TruncBeg[j] et (t2 > Ts))
  alors
  |    sol <— sol ∪ {Truncate beginning};
  Finsi
  Si ((numéro = 2) et (shift < 0) et TruncEnd[j] et (t1 < (Ts + shift)))
  alors
  |    sol <— sol ∪ {Truncate end};
  Finsi
  Retour(sol);
Fin
```

Les seules opérations possibles pour modifier t'1(i) sont le décalage local, le décalage global et la troncature du début de l'objet E_k. Les conditions suivantes sont faciles à établir:

Décalage local

Il suffit d'avoir Reloc(j).

Décalage global

Il n'est possible que si shift1(i) > 0, avec la condition Bt(i), ou en temps lisse pour nseq = 1.

Troncature de début

Elle peut s'appliquer uniquement dans la situation 3 si la propriété TruncBeg(j) est vraie.

8.8 Correction 1

Elle s'effectue en prenant une solution dans l'ensemble solution 1. Ce choix peut être laissé à l'utilisateur, ou énumératif si l'ensemble solution est arbitrairement ordonné.[250] L'ordre des “préférences”, dans ce dernier cas, peut être figé dans le programme ou modifié en cours d'exécution.

```
Nouveau_choix1:
  Si (ens_sol1[i] = vide)
  alors
  |    aller à Décrémentation;
  Finsi
  backtrack <— faux;
  dΔ1[i] <— 0;
  δ1[i] <— 0;
  sol1 <— Prochain_choix(ens_sol1[i]);
  ens_sol1[i] <— ens_sol1[i] \ {sol1};
  t'1[i] <— t1[k] + Δ[i] + dΔ0[i] + shift1[i];
  t'2[i] <— t2[k] + Δ[i] + dΔ0[i];
  shift2[i] <— 0;
  ens_sol2[i] <— vide;
  Si (sol1 = Truncate beginning)
  alors
  |    TrBeg[k] <— shift1[i];
  sinon
  |    TrBeg[k] <— 0;
  |    t'2[i] <— t'2[i] + shift1[i];
  |    Si (sol1 = Shift object) δ1[i] <— shift1[i];
  |    Si (sol1 = Break tempo) dΔ1[i] <— shift1[i];
  Finsi
  t"1[i] <— t'1[i];
  t"2[i] <— t'2[i];
  aller à Incrémentation;
```

La procédure **Prochain_choix(ens_sol1(i))**, qui dépend essentiellement de l'implémentation, n'est pas décrite ici.

[250] La solution {Break tempo} est toujours placée en dernier, car elle impose, lorsque nseq > 1, de relancer la procédure de positionnement (voir “Reprise” dans la boucle principale, §8.2).

Théorème IX.2

Si l'ensemble solution ens_sol1(i) est non vide pour toute valeur de *i*, la procédure Positionner() retourne une solution sans décrémentation.

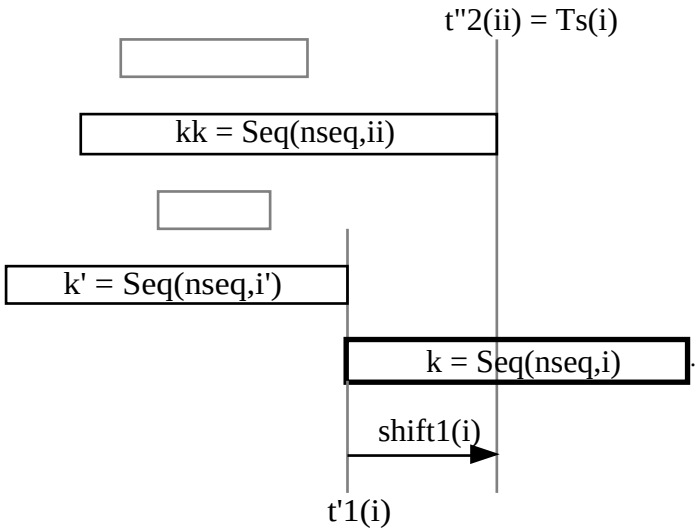
Preuve

Il suffit de suivre pas à pas les instructions de l'algorithme: ou bien la situation est acceptable et l'on incrémente *i*, ou bien elle n'est pas acceptable et l'on effecte la correction sur t'1(i) et t'2(i) pour ensuite incrémenter *i*.

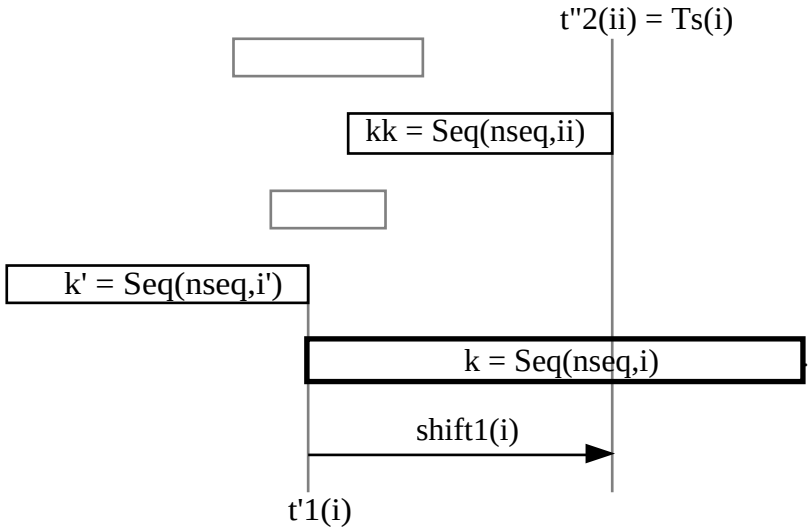
Il reste à prouver que la solution ainsi trouvée est correcte, autrement dit, informellement, que la correction 1 ne modifie pas les situations sur les objets précédant E_k dans la séquence. Nous établissons la preuve par récurrence. Pour i = 1 aucun objet ne précède E_k et la situation est toujours acceptable. Supposons maintenant que la configuration des objets précédant E_k (de rang *i*) soit acceptable. Si la situation de E_k est acceptable on peut incrémenter *i*. Dans le cas contraire on effectue la correction 1. Deux cas sont à distinguer:

(1) shift1(i) < 0. La correction 1 diminue t'1(i). La configuration topologique $\Omega(E_{kk},E_k)$ prend la valeur γ (voir tableau §5). D'après la définition de Ts(i), il est clair que la configuration topologique $\Omega(E_{k'},E_k)$ pour tout $k' \neq kk$ tel que k' = Seq(nseq,i') avec i' < i avait la valeur δ. Il est clair par ailleurs que, Ts(i) étant le majorant des t''2(i'), après correction la configuration $\Omega(E_{k'},E_k)$ garde la valeur δ, ou exceptionnellement prend la valeur γ. Dans ce dernier cas on est dans la situation 2, qui est acceptable quelles que soient les propriétés des objets. On peut donc incrémenter *i*.

(2) shift1(i) > 0. La correction 1 augmente t'1(i). La configuration topologique $\Omega(E_{kk},E_k)$ prend encore la valeur γ. Pour tous les autres objets précédents $E_{k'}$ avec $k' \neq kk$, on avait $\delta(E_{k'},E_k)$ ou $\gamma(E_{k'},E_k)$. Après correction on obtient $\delta(E_{k'},E_k)$. Si l'objet $E_{k'}$ possède la propriété ContEnd(j), avec j = Obj(k'), et qu'on avait $\gamma(E_{k'},E_k)$ avant correction, deux situations sont à envisager:



Cette situation est acceptable en raison de la configuration topologique des objets $E_{k'}$ et E_{kk} qui respecte la condition ContEnd(j'). La deuxième situation est la suivante:



avec deux cas:

(2.1) i' > ii. Ce cas est contraire à l'hypothèse car il reviendrait à valider la situation 4 lors du positionnement de $E_{k'}$, sachant qu'on a ContEnd(j').

(2.2) i' < ii. Appelons *inext'* le rang de l'objet suivant $E_{k'}$ dans la séquence. Puisque $E_{k'}$ n'est pas dans la situation 4 on a nécessairement Ts(inext') = t''2(i'), et ContEndPrev(inext) est vrai lors du positionnement de l'objet suivant. On ne peut donc pas valider la condition 1, ce qui revient à dire que l'objet suivant recouvre partiellement $E_{k'}$. Par conséquent le positionnement de E_k n'affecte pas la contrainte sur $E_{k'}$. ■

Corollaire

Pour nseq = 1 en temps lisse, **Positionner()** trouve toujours une solution sans décrémentation. En effet, **Situation()** retourne toujours *inacceptable* dans ce cas, et l'ensemble solution ens_sol1(i) est {Break tempo}. Cet ensemble n'étant pas vide, il n'y a pas de décrémentation.

8.9 Ensemble solution 2 (*Solution set 2*)

Si l'ensemble solution ens_sol1(i) est vide, on garde en mémoire la valeur de Ts(i) = Tsm, et l'on pose:

$$shift2(i) \leftarrow -shift1(i)$$

ce qui veut dire informellement: “si l'on ne peut pas augmenter t'1(i) de la valeur shift1(i), il faut diminuer Ts(i) de -shift1(i)”. Ici encore la correction sera **canonique**, car |shift2(i)| est la valeur minimale qui permette d'arriver à la situation 2 pour l'objet de rang *i*.

La décrémentation de *i* utilise les instructions:

```
Décrémentation:
  Si (non backtrack)  shift2[iprev] <— shift2[i];
  i <— iprev;
  Pour (iprev = i - 1; iprev ≥ -1)
  |    Si (iprev < 0)  Retour(échec);
  |    Si (Seq[nseq][iprev] > 0) interrompre Pour;
  |    iprev <— iprev - 1;
  Finpour
  k <— Seq[nseq][i]; /* k est strictement positif */
  Si (backtrack) aller à Nouveau_choix2;
  Si (t"2[i] < Tsm) aller à Décrémentation;
  t'1[i] <— t"1[i]; t'2[i] <— t"2[i];  /* On valide la correction
     précédente */
  ens_sol2[i] <—
     Choix_possibles(i,nseq,shift2[i],t"1[i],t"2[i],Tsm,nature_temps,2);
  aller à Nouveau_choix2;
Fin
```

La variable *backtrack* a la valeur *faux*. La nouvelle valeur de *i* est *iprev*, le rang de l'objet précédent non vide.[251] On cherche l'objet E_{kk} en comparant t"2(i) avec Tsm pour tout i' < i:

```
Si (t"2[i] < Tsm) aller à Décrémentation
```

A chaque étape de décrémentation on affecte la valeur shift2(i) à shift2(iprev).[252]

Les affectations:

```
t'1[i] <— t"1[i]
t'2[i] <— t"2[i]
```

ne modifient rien si l'on révise l'objet E_{kk} pour la première fois. Nous verrons un autre cas par la suite (§8.13). On détermine ensuite un ensemble de solutions ens_sol2(i) en appelant de nouveau la fonction

Choix_possibles(i,nseq,shift,t1,t2,Ts,nature_temps,numéro)

avec shift = shift2(i), t1 = t"1(i), t2 = t"2(i), Ts = Tsm, et numéro = 2.

(Voir au §8.1 les instructions de cette fonction.)

Décalage local

Il est accepté si Reloc(j) est vérifié. A noter que si shift2(i) est du signe opposé de shift1(i), la correction 1 étant canonique (i.e. |shift1(i)| est minimal), la correction devra être effectuée de nouveau (appel de la fonction **Autre_correction()**, voir infra). Nous montrerons que ce processus ne conduit pas à un bouclage.

Décalage global

Accepté si shift2(i) > 0 et si Bt(i) est vérifié.

Troncature en fin

Elle est possible pour shift2(i) < 0 si l'objet possède la propriété TruncEnd(j) et que:

---

[251] Si iprev = 0 il n'y a pas d'objet précédent et la procédure **Positionner()** échoue.

[252] Il est nécessaire de réaliser une pile pour les valeurs de shift2, ainsi que pour ens_sol1 et ens_sol2, pour permettre le *backtracking*.

$$t"1 < Tsm + shift2(i)$$

Cette dernière condition est symétrique de la condition t2 > Ts pour la correction 1: la troncature n'était possible qu'en situation 3.

8.10 Correction 2

Elle s'effectue, comme précédemment, en choisissant une solution dans l'ensemble solution 2, soit arbitrairement soit par un choix de l'utilisateur:

```
Nouveau_choix2:
  Si (ens_sol2[i] = vide) aller à Nouveau_choix1;
  backtrack <— faux;
  δ2[i] <— dΔ2[i] <— 0;
  sol2 <— Prochain_choix(ens_sol2[i]);
  ens_sol2[i] <— ens_sol2[i] \ {sol2};
  t"1[i] <— t'1[i];
  t"2[i] <— t'2[i] + shift2[i];
  Si (sol2 = Truncate end)
  alors
  |    TrEnd[k] <— -shift2[i];
  sinon
  |    TrEnd[k] <— 0;
  |    t"1[i] <— t"1[i] + shift2[i];
  |    Si (sol2 = Shift object) δ2[i] <— shift2[i];
  |    Si (sol2 = Break tempo) dΔ2[i] <— shift2[i];
  Finsi

  shift1[i] <— Ts[i] - t"1[i];
  shift3 <— Autre_correction1(i,nseq,shift1[i],t"2[i]);
  Si (shift3 = 0)
  alors
  |    aller à Incrémentation;
  sinon
  |    aller à Nouveau_choix2;
  Finsi
```

Si ens_sol2(i) est vide on saute à **Nouveau_choix1** pour réviser le choix de la première correction. En effet, si l'on ne révisait pas la correction 1 il faudrait “propager la contrainte shift2(i)” vers les objets précédant E_k. Cette contrainte n'étant pas nécessairement satisfaite, nous montrons que dans ce cas la révision de la correction 1 peut amener une solution. Quatre cas peuvent être envisagés:

(1) Le nouveau choix est {Truncate beginning}. Dans ce cas t'2(i) est modifié et l'on examine des situations nouvelles pour les objets de rang supérieur à *i*.

(2) Le nouveau choix est {Break tempo}: impossible, car si Bt(i) est vérifié alors ens_sol2(i) ne peut être vide.

(3) Le nouveau choix est {Shift object}: impossible, car si Reloc(j) est vérifié alors ens_sol2(i) ne peut être vide.

(4) ens_sol1(i) est vide. Dans ce cas on saute à **Décrémentation** et l'on “propage la contrainte” en faisant:

$$shift2(iprev) = shift2(i) .$$

Si la contrainte est satisfaite par un objet de rang i' < i, soit par une correction du type 2 soit par le choix d'une autre solution pour la correction de type 1, l'algorithme reprend le positionnement des objets de rang *i"* compris entre *i'* et *i* en recalculant les ens_sol1(i"). Puisque l'objet de rang *i* ne possède pas la

propriété Reloc(j) et que Bt(i) n'est pas vérifié, la seule solution qui sera envisagée pour la correction 1 sera {Truncate beginning}. Elle aboutira exactement aux mêmes solutions que dans le cas (1). Or si l'une des solutions résultant de ce choix était valide on n'aurait pas pu arriver au cas (4).

Cette analyse montre qu'il n'est pas utile en fait de propager la contrainte shift2(i) vers les objets précédents lorsque backtrack = faux et que ens_sol2(i) est vide immédiatement après la décrémentation. La situation sera différente en *backtracking* ou après avoir appelé **Autre_correction1()**, puisque dans ces deux cas ens_sol2(i) pourra devenir vide même si les conditions Reloc(j) ou Bt(i) sont vérifiées.

8.11 Autre correction 1 (*alternate correction 1*)

Une fois que la correction 2 a été effectuée il est nécessaire de vérifier que la configuration topologique $\Omega(E_{kk},E_k)$ est encore acceptable. A cet effet on appelle la fonction[253] **Autre_correction1**(i,nseq,shift,t2) avec shift = shift1(i) et t2 = t"2(i):

```
Fonction Autre_correction1(i,nseq,shift,t2)

Début
  k <— Seq[nseq][i];
  j <— Obj[k];
  t1 <— t"1[i];
  Si (shift = 0)  Retour(0);           /* Situation 2 */
  Si (shift < 0 et (non ContBeg[j]) et (non ContEndPrev[i]))
             Retour(0);                /* Situation 1 */
  Si (shift > 0)                       /* Situation 3 ou 4 */
  alors
  |    Si(OverBeg[j] et OverEndPrev[i] et (((t1 + shift) ≤ t2) ou (non
  |      ContEnd[j]))  Retour(0);
  |    Si (TruncBeg[j])
  |    alors
  |    |    Si ((t1 + shift) ≤ t2)      /* Situation 3 */
  |    |    alors
  |    |    |    TrBeg[k] <— TrBeg[k] + shift;
  |    |    |    t"1[i] <— t"1[i] + shift;
  |    |    |    Retour(0);
  |    |    Finsi
  |    Finsi
  Finsi
  Retour(shift);
Fin
```

La fonction retourne 0 dans les situations acceptables. Dans la situation 3 elle effectue une troncature en début si la propriété TruncBeg(j) est vérifiée, puis elle retourne 0. Dans tous les autres cas, elle retourne la valeur shift1(i).

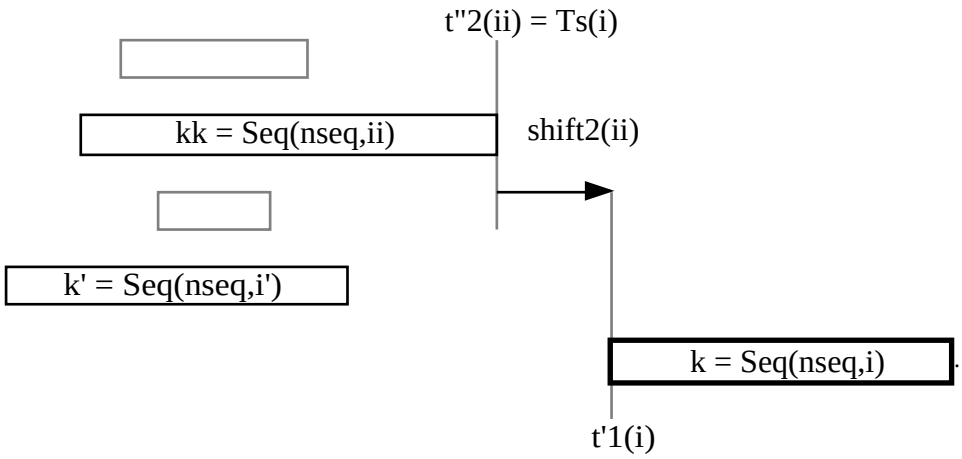
Lorsque la valeur retournée est 0 on incrémente de nouveau la valeur de *i*. Si la valeur retournée est non nulle, on cherche une autre solution dans ens_sol2(i). Lorsque ens_sol2(i) est vide on revient à **Nouveau_choix1**, puis en fonction du résultat on incrémente ou on décrémente *i*.

[253] Cette fonction examine aussi t"1(i) mais nous ne l'avons pas placé dans les paramètres parce que la fonction peut modifier sa valeur.

8.12 Effet de la correction 2 (*effect of correction 2*)

Nous allons montrer que la correction 2 améliore toujours la configuration, et qu'une solution peut être trouvée, si elle existe, en un nombre fini d'étapes. Considérons les deux cas:

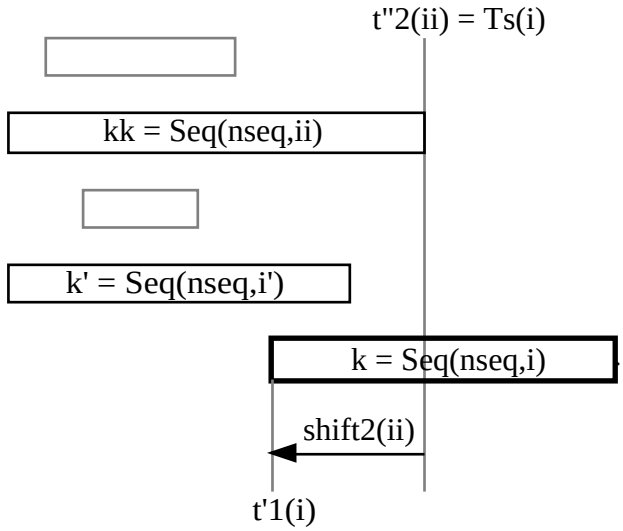
(1) Décrémentation à partir de l'objet E_k de rang *i*, sachant que shift1(i) < 0. On remonte à l'objet E_{kk} défini précédemment. La configuration $\Omega(E_{kk},E_k)$ est $\delta(E_{kk},E_k)$:



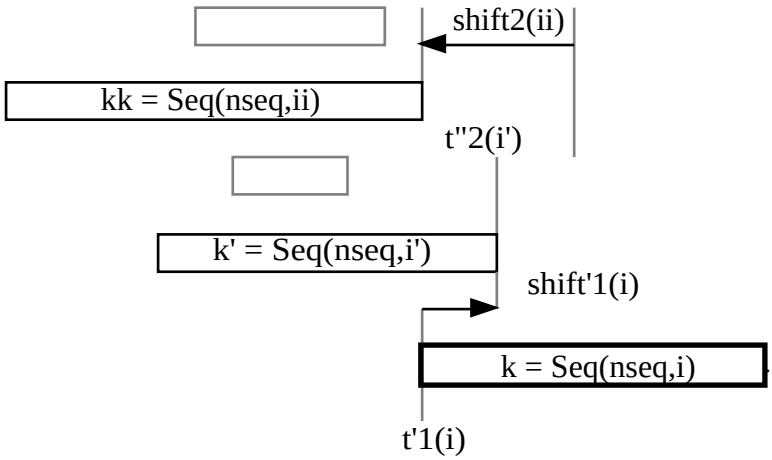
et pour tout *i'* tel que ii < i' < i on a nécessairement $\delta(E_{k'},E_k)$ avec k' = Seq(nseq,i'). Pour tous ces objets on est d'autre part dans la situation 4 puisque Ts(i') = t''2(ii) > t'2(i').[254] Après la correction 2 sur E_{kk} on retrouve exactement les mêmes situations et les objets se positionnent donc identiquement. Seul l'objet E_{kk} a donc été repositionné, et l'on revient à l'objet E_k avec la configuration $\gamma(E_{kk},E_k)$, c'est à dire la situation 2 qui est toujours acceptable.

[254] Il ne peut y avoir égalité parce qu'on est remonté à l'objet de rang maximum *ii* tel que Ts(i) = t''2(ii). (Voir **Décrémentation**).

(2) Décrémentation à partir de l'objet E_k de rang *i*, sachant que shift1(i) > 0. On remonte de la même manière à l'objet E_{kk}:



Pour tout *i'* tel que ii < i' < i on est encore dans la situation 4 puisque Ts(i') = t''2(ii) > t'2(i'). Pour la même raison que précédemment, seul l'objet E_{kk} a été repositionné. Il peut exister toutefois un objet $E_{k'}$, avec k' = Seq(nseq,i') et ii < i' < i, qui "annule une partie de la correction":



Dans ce cas, on revient à l'objet E_k avec une nouvelle valeur shift'1(i).

Théorème IX.3

shift'1(i) < shift1(i).

Preuve

On avait la situation 4 pour tous les objets $E_{k'}$, avec k' = Seq(nseq,i') et ii < i' < i. Donc t''2(i') < Ts(i) avant la correction de l'objet E_{kk}. Donc shift'1(i) < -shift2(ii). Comme shift2(ii) = -shift1(i), la relation est prouvée. ■

Théorème IX.4

La décrémentation sur un objet aboutit en un nombre fini d'étapes, soit à une solution, soit à un échec.

Preuve

La décrémentation est relancée avec shift2(i) = -shift'1(i). L'objet E_{kk} est maintenant remplacé par l'objet $E_{k'}$. Comme ii < i' < i, la suite des valeurs de *i'* est strictement croissante et bornée supérieurement par *i*. Cette suite est donc finie. ■

8.13 Corrections multiples (*multiple corrections*)

La correction 2 peut être effectuée plusieurs fois pour un objet de rang *i*. La première fois, lors de la décrémentation sur un objet de rang i1 > i, la seconde fois lors de la décrémentation sur un objet de rang i2 > i1, et ainsi de suite. Ces corrections sont cumulées grâce aux affectations

```
t'1[i] <— t"1[i]
t'2[i] <— t"2[i]
```

dans lesquelles t'1(i) et t'2(i) sont les coordonnées à partir de laquelle doit se faire la nouvelle correction, et t"1(i) et t"2(i) celles résultant de la précédente.

Théorème IX.5

Pour tout objet le nombre de corrections de type 2 est fini.

Preuve

Soit l'objet de rang *i* qui reçoit une première correction de type 2 à cause de la décrémentation sur un objet de rang i1 > i. Dans la preuve du théorème IX.4 nous avons établi que le positionnement de l'objet de rang *i1* se faisait en effectuant la correction de type 2 sur les objets de rang *i'*, où les valeurs de *i'* forment une suite strictement croissante. Par conséquent l'objet de rang *i* n'est corrigé qu'une seule fois dans la décrémentation sur l'objet de rang *i1*. Les corrections suivantes résultent donc nécessairement de décrémentations sur des objets de rang supérieur à *i1*. Ces objets étant en nombre fini, le nombre de corrections de type 2 sur l'objet de rang *i* est fini. ■

Ce théorème n'est pas nécessaire pour prouver que la procédure est algorithmique, mais il est introduit en raison de son analogie[255] avec le théorème IX.4.

8.14 Effectivité de la procédure (*effectiveness of the procedure*)

Théorème IX.6

La procédure **Positionner()** est algorithmique.

Preuve

Pour chaque valeur de *i* telle que Seq(nseq,i) > 0 on peut avoir:

(1) ens_sol1(i) ≠ vide. Dans ce cas on incrémente *i*.

(2) ens_sol1(i) = vide. Dans ce cas on décrémente *i*.

---

[255] Cette analogie résulte de la symétrie du problème de positionnement d'objets; symétrie qui n'est pas évidente en raison du traitement séquentiel qui “favorise” l'ordre croissant dans la séquence.

Dans le cas (2) on aboutit, après un nombre fini d'étapes (théorème IX.4), soit à un échec, soit à une nouvelle configuration pour laquelle ens_sol1(i) ≠ vide. On peut donc incrémenter *i*. La valeur de *i* étant bornée supérieurement, la procédure s'arrête en un nombre fini d'étapes. ■

Théorème IX.7

> Si pour une valeur donnée de *nseq* **Positionner()** retourne une solution telle que $\Delta(i) \neq 0$ pour une valeur de *i*, alors il existe une solution lorsque T(i) est remplacé par $T(i)+\Delta(i)$ pour tout *i*.

Preuve

Lorsqu'on relance la boucle principale, les valeurs de $\alpha(k)$ sont inchangées. Autrement dit, les "tailles" des objets: t2[k]-t1[k] restent les mêmes. Soit *i* le rang du premier objet pour lequel T(i) a été modifié. Il est évident que le positionnement de tous les objets de rang inférieur à *i* est possible comme précédemment. Par ailleurs, la situation qui avait provoqué le décalage global au rang *i* était:

(1) pour une correction de type 1, soit la situation 3 ou la situation 4 (voir §8.6). Dans ce cas on avait pris $\Delta(i)$ = shift1(i), et on est à présent, avec la nouvelle valeur de T(i), dans la situation 2 qui est acceptable.

(2) pour une correction de type 2, on avait pris $\Delta(i)$ = shift2(i), sachant que la contrainte ContBeg(j) (avec j = Obj(Seq(nseq,i))) pouvait être satisfaite. L'objet de rang *inext* était passé de la situation 1 à la situation 2. Il est à présent en situation 2.

Il est facile de voir que le même raisonnement peut s'appliquer aux objets de rang supérieur à *i*. La fonction **Positionner()** retourne donc nécessairement une solution dans ce cas. ■

8.15 Backtracking

Une énumération de solutions est possible grâce au *backtracking*. Si la première solution est refusée par l'opérateur, la décrémentation est lancée à partir du dernier objet de la séquence. Cette décrémentation s'effectue sans "propagation de contrainte" shift2(i), mais avec le drapeau backtrack = vrai. Le système revient alors sur le dernier choix effectué (sur une correction de type 1 ou 2).

9. Espace de recherche et solutions canoniques (*search space and canonic solutions*)

Il est facile de montrer que le *backtracking* permet d'afficher un ensemble fini de solutions. Nous appelons **solution canonique** tout élément de cet ensemble.

A toute solution canonique on peut associer un ensemble, vide ou infini, de solutions non canoniques obtenues, informellement, en modifiant les paramètres d'exécution de chaque objet sans induire de nouvelle contrainte avec les objets voisins dans la séquence.

L'espace de recherche de solutions au problème de la mise en temps d'une structure polymétrique est donc en général vide ou infini. L'algorithme que nous avons décrit permet d'explorer un sous-ensemble fini de cet espace.

Le cardinal de cet espace est fonction du nombre de décisions que peut prendre l'utilisateur (ou, par défaut, le système) lors des corrections de type 1 ou 2.

10. Complexité de l'algorithme de mise en temps (*complexity of the time-setting algorithm*)

On étudie ici la complexité de la procédure de positionnement en fonction du nombre *imax* d'objets sonores dans chaque séquence[256] et du nombre *nmax* de séquences. On suppose que le but de l'algorithme est de trouver la première solution ou, à défaut, de retourner un message d'échec.

10.1 Procédure Positionner() (*"Positionner()" procedure*)

Un cas typique est celui où l'ensemble solution ens_sol1(i) n'est pas vide: la décrémentation n'est alors pas utilisée, et la solution est trouvée après *imax* étapes d'incrémentation. Rappelons que ce cas est toujours réalisé en temps lisse et pour nseq = 1, ou encore lorsque les propriétés ContBeg(j) et OverBeg(j) ne sont pas prises en compte.[257]

L'algorithme peut aussi s'arrêter après un nombre d'étapes inférieur à *imax* lorsque le problème est impossible.

Dans le cas le plus défavorable, la décrémentation est invoquée pour le positionnement de chaque objet. Cette situation peut se produire par exemple si tous les objets sont disposés en tuilage[258] et qu'ils possèdent les propriétés BrkTempo(j), (non Reloc(j)), (non OverBeg(j)) et (non(TruncBeg(j)).

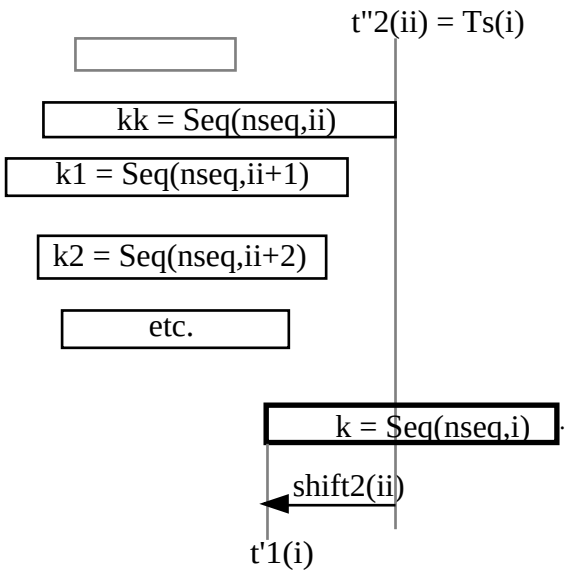
Le cas le plus défavorable, pour une décrémentation, est réalisé à partir de l'objet E_k de rang *i*, avec shift1(i) > 0, lorsque chaque correction de type 2 sur un objet précédant E_k ramène une nouvelle valeur shift'1(i) > 0, par exemple:

---

256 Ce nombre est identique pour chaque séquence pour une formule polymétrique complète (voir chapitre VIII §2).

257 Dans l'implémentation réalisée, deux drapeaux (*flags*) peuvent être activés pour inhiber ces propriétés. L'activation peut être provoquée par un dépassement du temps de calcul alloué à la procédure.

258 i.e. avec la configuration topologique $\beta 4(E_{k1},E_{k2})$ pour deux objets tels que k1 = Seq(nseq,i1), k2 = Seq(nseq,i2), et i2 = i1 + 1.



Au pire, on a ii = 1. Dans ce cas, le nombre d'étapes de décrémentation, pour le positionnement de l'objet E_k, est:

Recherche de l'objet de rang 1:	i-1 étapes
Retour à l'objet de rang i:	i-1 étapes
Total pour l'objet de rang 1:	2(i-1) étapes
…	
Total pour l'objet de rang i':	2(i-i') étapes
…	
Total pour l'objet de rang i-1:	2 étapes

Total pour le positionnement de E_k:

$$2 \sum_{i' = 1}^{i-1} (i - i') = i \,.\, (i - 1)$$

Si l'on envisage la décrémentation la plus défavorable pour tous les objets de la séquence, le nombre d'étapes de l'algorithme est alors:

$$\sum_{i=1}^{imax} i\,(i-1)$$

c'est à dire de l'ordre de $O(imax^3)$.

10.2 Instanciation d'une structure polymétrique (*instanciating a polymetric structure*)

La structure polymétrique comporte *nmax* arguments de longueur maximale *imax*. Lorsqu'aucun décalage global n'est effectué, la complexité de l'algorithme est donc $O(nmax.imax^3)$.

Dans le cas le plus défavorable, pour chaque valeur de *nseq* telle que nseq > 1 un décalage global (solution {Break tempo}) est nécessaire. L'algorithme doit être relancé pour nseq > 1. D'après le théorème IX.7, la fois suivante le positionnement est possible pour la séquence *nseq* sans nouveau décalage global. Dans ce cas, la complexité est donc $O(nmax^2.imax^3)$.

11. Résumé et discussion de cette méthode (*summary and discussion of the method*)

Nous avons décrit une méthode permettant d'instancier des objets sonores définis par des “prototypes” (ou classes d'objets). Ces prototypes sont des séquences typiques de messages dont on connaît à la fois la durée totale et, dans le cas d'objets striés, une période de référence. Par ailleurs, l'instanciation des objets se fait sur une structure temporelle qui est à son tour lisse ou striée. A partir de ces données la méthode permet de calculer le “coefficient de dilatation/contraction” de chaque objet, autrement dit de déterminer sa durée physique dans le contexte de la structure musicale. Le même objet peut ainsi être instancié avec des dilatations/contractions différentes si sa durée symbolique est différente, ou encore si l'on passe du temps lisse au temps strié et vice-versa.

A tout objet strié on peut associer un “pivot” représentant un point temporel particulier qui, en l'absence d'autres contraintes, doit être positionné sur une strie de la structure temporelle. Le pivot par défaut est l'origine de l'intervalle support temporel. Un objet sans pivot peut être considéré comme un objet avec un pivot arbitraire déplaçable (propriété Reloc).

Chaque objet possède des propriétés topologiques (conditions de recouvrement et continuité avec les objets voisins de la séquence) et métriques (conditions de dilatation/contraction). La mise en temps d'une structure musicale arbitraire se ramène ainsi à un problème de satisfaction de contraintes, avec la possibilité de relâcher certaines contraintes lorsque le problème n'admet pas de solution. La résolution des contraintes peut occasionner le déplacement de certains objets par rapport aux stries, ou encore le décalage global des stries (le classique “point d'orgue”); elle peut aussi causer la troncature du début ou de la fin de certains objets.

En temps strié, la configuration des intervalles supports temporels successifs n'est pas nécessairement celle d'une séquence stricte. L'ordre des symboles dans la séquence est toutefois reflété par celui des positions des pivots des objets correspondants. La configuration est acceptable dans la mesure où un objet strié est perçu comme un événement “centré” sur son pivot et non sur le début de son intervalle support temporel.

Un avantage de l'algorithme décrit est qu'il est de complexité $O(n^3)$ dans le pire cas, où *n* représente la longueur d'une séquence, et linéaire dans la plupart des cas. Il est donc bien adapté aux conditions de “temps réel”, sachant qu'en cas de dépassement du temps de calcul alloué, le système peut relâcher les contraintes de continuité puis de recouvrement afin de fournir une solution approximative en temps linéaire.

Un autre avantage est qu'il traite les objets dans l'ordre de la séquence, avec des retours en arrière (*backtracking*) en cas d'échec. Cette disposition permet une exécution pas-à-pas avec une prise de décision manuelle sur les déplacements, troncatures d'objets, ou relâchements de contraintes. On couvre ainsi plusieurs modes d'interaction avec l'utilisateur, depuis la composition “pensée” jusqu'à l'improvisation automatique.

X. Conclusion

Les travaux présentés dans cette étude apportent un certain nombre de réponses dans le domaine de l'informatique théorique comme dans celui de la musicologie.

Pour ce qui est de l'informatique, nous avons défini une classe de langages formels (grammaires BP) qui permettent de décrire des familles de chaînes de symboles comportant des **motifs de répétition** et des **transformations homomorphiques**. Les dérivations peuvent êtres contextuelles, le cas échéant avec des contextes “négatifs”, de sorte qu'il est possible de modéliser des “schémas” de réécriture faciles à appréhender intuitivement. Pour ces grammaires nous avons mis au point un algorithme déterministe de **test d'appartenance** qui permet un contrôle des étapes de la reconnaissance de formes. Le formalisme grammatical intègre bien les notions de connaissances déclarative et procédurale, comme l'ont prouvé les applications expérimentales sur des langages non triviaux.

Nous avons posé le problème de l'**acquisition des connaissances** dans des langages réguliers pour lesquels n'existent ni lexique ni segmentation préalable. L'algorithme d'apprentissage que nous avons présenté permet d'inférer à la fois le lexique et les règles de structure profonde (tout en se limitant aux grammaires de type 2). Nous avons montré, à partir d'un exemple, le rôle que pouvait jouer un dialogue entre la machine et l'expert dans l'amélioration du processus d'inférence.

La deuxième partie de l'étude concerne la représentation de **processus concurrentiels** soumis à des **contraintes de synchronisation** qui peuvent être explicites ou incomplètement formulées. Résoudre le problème de la synchronisation dans un univers d'événements revient à expliciter toutes les relations d'antériorité ou de simultanéité sur ces événements. Ayant posé ce problème dans le cas général, nous avons défini une classe d'univers d'événements que l'on peut représenter avec des expressions parenthésées. Cette classe est intéressante parce qu'elle permet de résoudre efficacement le problème de la synchronisation en vérifiant la cohérence de l'univers d'événements. L'algorithme de **résolution de formule polymétrique** permet de compléter la structure en faisant l'hypothèse d'une description la plus “simple” possible. Nous avons ensuite envisagé le cas où chaque symbole représentait un **objet sonore**, i.e. une séquence de messages déclenchant des processus (de synthèse du son par exemple). Un certain nombre de contraintes d'ordre métrique ou topologique conditionnent l'instanciation (la “**mise en temps**”) de ces objets. Nous avons proposé un algorithme rapide de résolution pour une classe de contraintes pertinente pour des objets sonores utilisés en composition musicale.

Pour ce qui est de l'apport musicologique, nous avons défendu une approche qui consiste à décrire la musique, non pas en tant que **produit** (l'œuvre musicale sous sa forme sonore ou ses représentations picturales) mais en tant qu'**activité**. Les types d'activité qui ont retenu notre attention sont l'**improvisation** et la **composition à**

partir de règles.[259] La modélisation des activités est réalisée en insérant l'ordinateur dans l'**environnement de tâches** des musiciens, soit en tant qu'apprenti (dans un contexte de tradition orale), soit en tant qu'outil (dans un contexte de composition contemporaine). Les modèles ainsi développés sont donc des **machines** (grammaires formelles ou automates).

Nous avons proposé une approche nouvelle pour le **traitement du temps** dans une structure d'**objets sonores**, introduisant les notions de **temps symbolique** (ou virtuel), de structure du temps (**strié** ou **lisse**) et de **synchronisation par pivots**.

Les travaux présentés dans le cadre de cette thèse sont loin d'épuiser les sujets abordés. On peut tout au plus les considérer comme un premier pas de la musicologie cognitive et de l'informatique théorique dans des domaines jusqu'ici réservés aux sciences humaines, à la musicologie traditionnelle, et à un certain pragmatisme technologique. En premier lieu, les propriétés algébriques des modèles nécessiteraient une étude plus approfondie, susceptible de dégager des propriétés utiles pour l'analyse musicale elle-même — en particulier pour tout ce qui touche aux processus concurrentiels.[260] Les méthodes d'inférence inductive appliquées aux langages formels font encore l'objet de travaux théoriques (dans le domaine de la reconnaissance syntaxique de formes) qui pourront déboucher sur une étude plus approfondie du transfert de connaissances dans les situations concrètes d'enseignement. Enfin, il paraît souhaitable d'établir une “passerelle” théorique entre les modèles de structure syntaxique proposés dans cette étude et un certain nombre de travaux réalisés en musicométrie.[261]

[259] Laske 1989a.

[260] Voir par exemple Chemillier & Timis 1988.

[261] Par exemple, Vecchione, Lerdahl & Jackendoff, Camilleri, Baroni, Nattiez, Chemillier & Timis, Smaill & Wiggins, pour n'en citer que quelques uns.

Bibliographie

ALLEN, James F. & Henry A. KAUTZ, 1985
A model of naive temporal reasoning. In J.R. Hobbs & R.C. Moore (Eds.) *Formal Theories of the Commonsense World*, Ablex, Norwood NJ (USA), pp.251-68.

ALLOUCHE, Jean Paul, 1987
Automates finis en théorie des nombres. *Expositiones Mathematicae* 5, pp.239-66.

ALLOUCHE, Jean Paul, & André MOURET, 1988
Libertés non anarchiques, automates finis et champs matriciels. *Colloque International "Structures Musicales et Assistance Informatique"*, Marseille, pp.45-50.

AMES, Charles, 1989
Tutorial and Cookbook for COMPOSE. Frog Peak Music, Oakland CA (USA), 1989.

ANDERSON, David P., & Ron KUIVILA, 1989
Continuous abstractions for discrete event languages. *Computer Music Journal*, 13, 3, pp.11-23.

ANGLUIN, Dana, 1980a
Inductive inference of formal languages from positive data. *Information & Control*, 45, 2, pp.117-35.

1980b
Finding patterns common to a set of strings. *Journal of Computer and System Sciences* 21, pp.46-62.

1982
Inference of reversible languages. *Journal of the ACM*, 29, 3, pp.741-65.

1983
Inductive inference: theory and methods. *Computing Surveys*, 15, 3, pp.237-69.

BAILY, John, 1989
Operational and representational models of music structure. *Symposium on Music Cognition*, Massachussets Institute of Technology, 8 novembre, 21 pages.

BALABAN, Mira, 1990
Music structures: a temporal-hierarchical representation for music. Rapport technique FC-TR-021 MCS-313, Ben Gurion University (Israël). A paraître dans *Musikometrika* 2.

BALABAN, Mira, & Neil V. MURRAY, 1989
The logic of time structures: temporal and nonmonotonic features. *11th Intern. Joint Conf. on Artificial Intelligence*. Detroit MI (USA), pp.1285-90.

BAYLE, François, 1988
Savoir-faire. *Colloque International "Structures Musicales et Assistance Informatique"*, Marseille, pp.67-76.

BEL, Bernard, 1987a
Grammaires de langages rythmiques. Mémoire de DEA de Mathématiques et Informatique, GIA, Faculté des Sciences de Luminy, Université Marseille II (France).

1987b
Grammaires de génération et de reconnaissance de phrases rythmiques. *6ème Congrès AFCET*: *Reconnaissance des Formes et Intelligence Artificielle*, Antibes (France), pp.353-66.

1988

Raga: approches conceptuelles et expérimentales. *Colloque International "Structures Musicales et Assistance Informatique"*, Marseille (France), pp.87-108.

1989a

Pattern grammars in formal representations of musical structures. Workshop on AI and Music, *11th Intern. Joint Conf. on Artificial Intelligence* (IJCAI), Detroit MI (USA), pp.118-45.

1989b

Inférence d'un langage formel. Note n°380, GRTC, Centre National de la Recherche Scientifique, Marseille (France).

1990a

Inférence de langages réguliers. *Journées Françaises de l'Apprentissage* (JFA), Lannion (France), pp.5-27.

1990b

Grammaires BP pour des airs de sonneurs de cloches. Note interne, Laboratoire Musique et Informatique de Marseille (France).

1990c

Bol Processor BP2: reference manual. Documentation sur réseau télématique.

1990d

En finir avec l'ethnomusicologie? In B. Vecchione et B. Bel (Eds.) *Le Fait Musical — Sciences, Technologies, Pratiques*, actes du Colloque "Musique et Assistance Informatique" (MAI 90), Marseille (France), pp.453-63.

1990e

Time and musical structures. *Interface*, 19, 2-3, pp.107-35.

BEL, Andréine, & Bernard BEL, 1990

Activité créatrice "transformationnelle" — Théorie et pratique. In B. Vecchione et B. Bel (Eds.) *Le Fait Musical — Sciences, Technologies, Pratiques*, actes du Colloque "Musique et Assistance Informatique" (MAI 90), Marseille (France), pp.193-206.

BERWICK, Robert C., & Sam PILATO, 1987

Learning syntax by automata induction. *Machine Learning*, 2, 1, pp.9-38.

BIERMANN, A.W., & J.A. FELDMAN, 1972

A survey of grammatical inference. In S. Watanabe (Ed.) *Frontiers of Pattern Recognition*, Academic Press, New York NJ (USA), pp.31-54.

BLACKING, John, 1972

Extensions and limits of musical transformations. *SEM meetings*, Toronto (Canada). (Non publié)

1974

Ethnomusicology as a key subject in the social sciences. *In Memoriam Antonio Jorge Dias*, 3, Lisbonne (Portugal), pp.71-93.

1984

What languages do musical grammars describe? In *Musical Grammars and Computer Analysis*, M. Baroni & L. Callegari, Eds., Olschki, Florence (Italie), pp.363-70.

1985

Dialectical ethnomusicology: making sense of Ellis and Adler in 1985. *European Seminar of Ethnomusicology*, Queen's University of Belfast (UK), 23-28 mars.

1989

Interview with Keith Howard. Queen's University of Belfast, 28-30 septembre.

BLEVIS, Eli, Michel FERET & Michael JENKINS, 1990

A computational paradigm for exploring creative musical thought. In B. Vecchione et B. Bel (Eds.) *Le Fait Musical — Sciences, Technologies, Pratiques*, actes du Colloque "Musique et Assistance Informatique" (MAI 90), Marseille (France), pp.363-93.

BOOTH, T.L., & R.A. THOMPSON, 1973

Applying probability measures to abstract languages. *IEEE Transactions on Computers*, C-22, 5, pp.442-50.

BORIN, Gianpaolo, Giovanni DE POLI & Augusto SARTI, 1990

Formalization of the sound generation process — Structures and metaphors. In B. Vecchione et B. Bel (Eds.) *Le Fait Musical — Sciences, Technologies, Pratiques*, actes du Colloque "Musique et Assistance Informatique" (MAI 90), Marseille (France), pp.231-41.

BOULEZ, Pierre, 1963

Penser la musique aujourd'hui. Gonthier, Paris (France).

BYRD, Donald, 1977

An integrated computer music software system. *Computer Music Journal*, 1, 2, pp.55-60.

CAMILLERI, Lelio, 1984

A grammar of the melodies of Schuberts Lieder. In M. Baroni & L. Callegari (Eds.) *Musical Grammars and Computer Analysis*, Olschki, Florence (Italie), pp.229-36.

CAMILLERI, Lelio, Francesco CARRERAS & Chiara DURANTI, 1990

An expert system prototype for the study of musical segmentation. *Interface*, 19, 2-3, pp.147-54.

CAMURRI, Antonio, Marcello FRIXIONE & Renato ZACCARIA, 1990

A music knowledge representation system combining symbolic and analogic approaches. In B. Vecchione et B. Bel (Eds.) *Le Fait Musical — Sciences, Technologies, Pratiques*, actes du Colloque "Musique et Assistance Informatique" (MAI 90), Marseille (France), pp.385-93.

CASE, John, & Christopher LYNES, 1982

Machine inductive inference and language identification. *International Colloquium on Algorithms, Languages and Programming*, pp.107-15.

CHEMILLIER, Marc, 1987

Monoïde libre et musique: deuxième partie. *RAIRO — Informatique Théorique et Applications*, 21, 4, pp.379-418.

1990a

Structure et méthode algébriques en informatique musicale. Thèse de Doctorat. Laboratoire d'Informatique Théorique et de Programmation, Paris (France).

1990b

Langages musicaux et automates — La rationalité du langage sériel. In B. Vecchione et B. Bel (Eds.) *Le Fait Musical — Sciences, Technologies, Pratiques*, actes du Colloque "Musique et Assistance Informatique" (MAI 90), Marseille (France), pp.303-17.

CHEMILLIER, Marc, & Dan TIMIS, 1988

Toward a theory of formal musical languages. *14th International Computer Music Conference*, Cologne (RFA), pp.175-83.

CHIARUCCI, Henri, 1972

Essai d'analyse structurale d'œuvres musicales. *Musique en jeu*, 12, pp.11-43.

CHOMSKY, Noam, 1964

Current issues in linguistic theory. In Katz & Fodor (Eds.) *Readings in the Philosophy of Language*, Prentice-Hall, Englewood Cliffs NJ (USA).

1965

Aspects of the theory of syntax. MIT Press, Cambridge MA (USA).

CHOMSKY, Noam, & Morris HALLE, 1973

Principes de phonologie générative. Seuil, Paris (France). Ed. originale: Harper & Row, New York, 1968.

COINTE, Pierre, & Xavier RODET, 1983

FORMES: A new object-language for managing of hierarchy of events. Rapport IRCAM, Paris (France).

COONS, E., & D. KRAENHENBUEHL, 1958

Information as a measure of structure in music. *Journal of Music Theory*, 2, pp.127-61.

COOPER, G., & L. MEYER, 1960

The rhythmic structure of music. University of Chicago Press, Chicago (USA), 1960.

COURTOT, Francis, 1990

Représentation évolutive pour l'aide à la composition. In B. Vecchione et B. Bel (Eds.) *Le Fait Musical — Sciences, Technologies, Pratiques*, actes du Colloque "Musique et Assistance Informatique" (MAI 90), Marseille (France), pp.343-61.

CROCHEMORE, Maxime, 1984

Linear searching for a square in a word. *Bulletin of EATCS*, 24, pp.66-72.

DANNENBERG, Roger B., 1984

An on-line algorithm for real-time accompaniment. *International Computer Music Conference* (ICMC), IRCAM, Paris, pp.193-8.

DELALANDE, François, 1988

Eléments d'analyse de la stratégie de composition. In B. Vecchione et B. Bel (Eds.) *Le Fait Musical — Sciences, Technologies, Pratiques*, actes du Colloque "Musique et Assistance Informatique" (MAI 90), Marseille (France), pp.51-65.

DELIEGE, Célestin, 1983

A propos de l'ouvrage de Lerdahl et Jackendoff "A Generative Theory of Tonal Music". *Revue de Musicologie des Universités Canadiennes*, 4, pp.141-83.

1984

Les fondements de la musique tonale. Lattès, Paris (France).

1989

La set-theory ou les enjeux du pléonasme. *Analyse Musicale*, 4ème trimestre, pp.64-79.

DEUTSCH, Diana, 1982

Grouping mechanisms in music. In D. Deutsch (Ed.) *The Psychology of Music*, Academic Press, New York NJ (USA).

DIENER, Glendon, 1988

T-Trees: an active data structure for computer music. *Proceedings of the 14th International Computer Music Conference*, Cologne (RFA), pp.184-88.

DIETTERICH, Thomas G., & Ryszard S. MICHALSKI, 1983

A comparative review of selected methods for learning from examples. In R.S. Michalski, J.G. Carbonell et T.M. Mitchell (Eds.) *Machine Learning: An Artificial Intelligence Approach*, Morgan Kaufmann, Los Altos CA (USA), 1983, pp.41-82.

DROMAN, D., 1983

Exploring MIDI — The Musical Instrument Digital Interface. *IMA Publications*, North Hollywood CA (USA).

DUTHEN, Jacques, & Marco STROPPA, 1990
Une représentation de structures temporelles par synchronisation de pivots. In B. Vecchione et B. Bel (Eds.) *Le Fait Musical — Sciences, Technologies, Pratiques,* actes du Colloque "Musique et Assistance Informatique" (MAI 90), Marseille (France), pp.417-9.

EMMET, Dorothy, 1976
"Motivation" in Sociology and Social Anthropology. *Journal of Social Behaviour*, 6, 1, pp.85-104.

ERIKSON, Robert F., 1977
The DARMS project: a status report. *Computers and the Humanities*, 9, 6, pp.291-8.

EVANS, Thomas G., 1969
Grammatical inference in pattern analysis. *3d International Symposium on Computer and Information Sciences*, Miami Beach CA (USA), pp.183-202.

FARNSWORTH, Paul, 1958
The social psychology of music. Dryden, New York (USA).

FELD, Stephen, 1974
Linguistic models in ethnomusicology. *Ethnomusicology,* 18, 2, pp.197-217.

FORTE, Allen, 1980
Aspects of rhythm in Webern's tonal music. *Music Theory Spectrum,* II, pp.90-109.

FU, King Sun, & T.L. BOOTH, 1975a
Grammatical inference: introduction and survey - Part I. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-5, 1, pp.95-111.

1975b
Grammatical inference: introduction and survey - Part II. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-5, 4, pp.409-23.

FUCHS, W., 1961
Musical analysis by mathematics. *Gravesano Blätter*, 1, Gravesano (Suisse).

GOLD, E. Mark, 1967
Language identification in the limit. *Information and Control*, 10, pp.447-74.

1978
Complexity of automation identification from given data. *Information and Control*, 37, pp.302-20.

GOTTLIEB, Robert S., 1977
The major traditions of North Indian tabla drumming. Emil Katzbichler, Munich (RFA).

GOURLAY, J.S., 1986
A language for music printing. *CACM*, 29, 5, pp.388-401.

GUéNOCHE, Alain, 1989
Algorithmes d'apprentissage dans les chaînes de caractères. *Journées Françaises de l'Apprentissage,* St Malo (France).

HAMEL, K., 1984
Musprint manual. Triangle resources, Cambridge MA (USA).

HART, Johnson M., 1980
Derivation structures for strictly context-sensitive grammars. *Information & Control* 45, pp.68-89.

1976
Right and left parses in phrase-structure grammars. *Information & Control,* 32, pp.242-62.

HIRSCHBERG, D.S., 1975

A linear space algorithm for computing maximal common subsequences. *Communications of the ACM*, 18, 6, pp.341-3.

HONING, Henkjan, 1990

Issues in the representation of time and structure in music. Centre for Knowledge Technology, Utrecht (Pays Bas). [Non publié]

JACKENDOFF, Ray, & Fred LERDAHL, 1982

A grammatical parallel between music and language. In M. Clynes (Ed.) *Music, Mind and Brain: the Neuropsychology of Music*, Plenum Press, New York NJ (USA), pp.83-117.

JAFFE, David, 1985

Ensemble timing in computer music. *Computer Music Journal*, 9, 4, pp.38-48.

JAKOBSON, R., 1963

Essais de linguistique générale. Editions de Minuit, Paris (France).

JONES, Jacqueline, Don L. SCARBOROUGH & Benjamin O. MILLER, 1990

GTSIM — A computer simulation of music perception. In B. Vecchione et B. Bel (Eds.) *Le Fait Musical — Sciences, Technologies, Pratiques*, actes du Colloque "Musique et Assistance Informatique" (MAI 90), Marseille (France), pp.435-41.

JONES, Kevin, 1989

Compositional applications of stochastic processes. In C. Roads (Ed.) *The Music Machine*, MIT Press, Cambridge MA (USA), pp.381-97.

KAIN, Richard Y., 1981

Automata theory: machines and languages. Krieger, Malabar (USA).

KATAYOSE, Haruhiro, & Seiji INOKUCHI

Learning performance rules in music interpretation system. In B. Vecchione et B. Bel (Eds.) *Le Fait Musical — Sciences, Technologies, Pratiques*, actes du Colloque "Musique et Assistance Informatique" (MAI 90), Marseille (France), pp.423-34.

KAUFMANN, A., & E. PICHAT, 1977

Méthodes mathématiques non numériques et leurs algorithmes. 2 volumes. Masson, Paris (France).

KIPPEN, Jim, 1985

The dialectical approach: a methodology for the analysis of tabla music. *Bulletin of the International Council for Traditional Music*, UK Chapter.

1987

An ethnomusicological approach to the analysis of musical cognition. *Music Perception* 5, 2, pp.173-95.

1988

The Tabla of Lucknow: a Cultural Analysis of a Musical Tradition. Cambridge University Press, Cambridge (UK).

1990

Music and the computer: some anthropological considerations. In B. Vecchione et B. Bel (Eds.) *Le Fait Musical — Sciences, Technologies, Pratiques*, actes du Colloque "Musique et Assistance Informatique" (MAI 90), Marseille (France), pp.41-50.

KIPPEN, Jim, & Bernard BEL, 1989a

The identification and modelling of a percussion "language", and the emergence of musical concepts in a machine-learning experimental set-up. *Computers & Humanities* 23, 3, pp.199-214.

1989b

Can a computer help resolve the problem of ethnographic description?, *Anthropological Quarterly*, 62, 3, pp.131-44.

1989c

A pragmatic application for computers in experimental ethnomusicology. *ALLC/ICCH Conference*, Toronto (Canada), 6-10 juin.

1990

Modelling music with grammars: formal language representation in the Bol Processor. In A. Marsden and A. Pople. (Eds.) *Computer Representations and Models in Music*, Academic Press, Londres (UK). (A paraître)

KRONLAND-MARTINET, Richard, & Alex GROSSMANN, 1991

Applications of time-frequency and time-scale (wavelet transform) methods to the analysis, synthesis and transformation of natural sounds. In C. Road, G. De Poli & A. Picciali (Eds.) *Representations of musical signals*, MIT Press. (A paraître)

LALOUM, Claude, & Gilbert ROUGET, 1965

Deux chants liturgiques Yoruba. *Journal de la Société des Africanistes*, 35, 1, pp.109-39.

LANDRAUD, A., Avril, J.F., & P. CHRETIENNE, 1989

An algorithm for finding a common structure shared by a family of strings. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, II, 8, pp.890-5.

LASKE, Otto, 1972a

On problems of a performance model for music. Institute of Sonology, Utrecht State University (Pays Bas).

1972b

In search of a generative grammar for music. In *Perspectives of New Music*, Fall-Winter 1973, Spring-Summer 1974, rédigé en 1972, pp.351-78.

1973

On the understanding and design of aesthetic artefacts. In P. Faltin & H.P. Reinecke (Eds.) *Musik und Verstehen*, Arno Volk, Cologne (RFA), pp.189-216.

1975

Introduction to a generative grammar for music. Sonological reports 1b, Institute of Sonology, Utrecht State University (Pays Bas)

1984

KEITH: a rule system for making music-analytical discoveries. In M. Baroni & L. Callegari (Eds.) *Musical Grammars and Computer Analysis*, Olschki, Florence (Italie), pp.165-99.

1988a

Comments on the first workshop on A.I. and music, 1988 AAAI Conference, St Paul MN, 12 pages. (Non publié)

1988b

A three-phase approach to designing knowledge-based systems. *CC-AI*, 5, 2, pp.147-64.

1989a

Composition theory: An enrichment of music theory. *Interface*, 18, pp.45-59.

1989b

Music composition as hypothesis formation: a blackboard concept of musical creativity. In *Knowledge-based systems*, Guilford (UK). (A paraître)

1990a

Three prototypical approaches in artistic composition. (Non publié).

1990b

The computer as the artist's alter-ego in music composition. *Leonardo*, 23, 1, pp.53-66.

1990c

Two paradigms of music research — Composition and Listening. In B. Vecchione et B. Bel (Eds.) *Le Fait Musical — Sciences, Technologies, Pratiques*, actes du Colloque "Musique et Assistance Informatique" (MAI 90), Marseille (France), pp.179-92.

1991a

Toward an epistemology of composition. *Interface.* (A paraître).

1991b

Two approaches to knowledge acquisition in music. In M. Balaban, K. Ebcioglu & O. Laske (Eds.) *Musical Knowledge*, section III(A), chap.5, American Association for Artificial Intelligence. (A paraître)

1991c

An epistemic approach to musicology. In G. Haus (Ed.) *Music Processing*, Ephraim Nissan (series Ed.), JAI Press. (A paraître)

LEMAN, Marc, 1990

Some epistemological considerations on symbolic and subsymbolic processing. In B. Vecchione et B. Bel (Eds.) *Le Fait Musical — Sciences, Technologies, Pratiques*, actes du Colloque "Musique et Assistance Informatique" (MAI 90), Marseille (France), pp.129-43.

LERDAHL, Fred & Ray JACKENDOFF, 1983

A Generative Theory of Tonal Music. MIT Press, Cambridge MA (USA).

LIBERMAN, M., & A. PRINCE, 1977

On stress and lingusitic rhythm. *Lingusitic Enquiry*, 8, 2, pp.249-336.

LIDOV, David, 1972

An example (from Kulintang) of a generative grammar for melody. *SEM meetings*, Toronto (Canada).

LISCHKA, Christoph, 1990

Some remarks concerning the application of neural networks to music and musicology. In B. Vecchione et B. Bel (Eds.) *Le Fait Musical — Sciences, Technologies, Pratiques*, actes du Colloque "Musique et Assistance Informatique" (MAI 90), Marseille (France), pp.223-7.

LOTHAIRE, M., 1983

Combinatorics on words. Encyclopedia of Mathematics and its Applications, 17, Addison-Wesley, Reading MA (USA).

MAKANIN, G.S., 1977

The problem of solvability of equations in a free semi-group. *Math. USSR Sbornik* 32, 2. Voir aussi *American Mathematical Society*, 1978.

MARQUES, Simonne, 1990

Musique et Mouvement: Pour une pédagogie "naturaliste" de l'écoute et de la production musicales. Edisud, Aix-en-Provence (France). (A paraître)

MARYANSKI, F.J., & T.L. BOOTH, 1977

Inference of finite-state probabilistic grammars. *IEEE Transactions on Computers*, C-26, 6, pp.521-36.
[Certaines anomalies de cet article ont été corrigées par B.R. Gaines: Maryanskis Grammatical Inferencer, *IEEE Transactions on Computers*, C-27, 1, 1979, pp.62-64]

MAXWELL, J.T., & S.M. ORNSTEIN, 1983

Mockingbird: A composer's amanuensis. CSL-83-2, Xerox Corporation, Pao Alto CA (USA).

MAZURKIEWICZ, Antoni, 1984a

Semantics of concurrent systems: a modular fixed-point trace approach. *5th European Workshop on Applications and Theory of Petri Nets*, 1984a, pp.353-71.

1984b

Traces, histories, graphs: instances of a process monoïd. *Lecture Notes in Computer Science* 176, pp.115-33.

McADAMS, Stephen, & A. BREGMAN, 1985

Hearing musical streams. In C. Roads & J. Strawn (Eds.) *Foundations of Computer Music*, MIT Press, Cambridge MA (USA).

MEYER, L.B., 1956

Emotion and meaning in music. University of Chicago Press, Chicago IL (USA).

1957

Meaning in music and information theory. *Journal of Aesth. & Art Criticism*, 15.

MICHALSKI, Ryszard S., 1983

A theory and methodology of inductive learning. In R.S. Michalski, J.G. Carbonell et T.M. Mitchell (Eds.) *Machine Learning: An Artificial Intelligence Approach*, Morgan Kaufmann, Los Altos CA (USA), pp.83-134.

MINSKY, Marvin, 1986

The Society of Mind. Simon & Schuster, New York NJ (USA).

MOLES, A., 1958

Théorie de l'information et perception esthétique. Denoël, Paris (France).

MOLINO, Jean, 1975

Fait musical et sémiologie de la musique. *Musique en Jeu*, 17, pp.37-62.

1988

Musique et Machine. *Colloque International "Structures Musicales et Assistance Informatique"*, Marseille, pp.141-54.

MOOG, Bob, 1986

MIDI: Musical Instrument Digital Interface. *Journal of the Audio Engineering Society*, 34, 5, pp.394-404.

NATTIEZ, Jean-Jacques, 1976

Fondements d'une sémiologie de la musique. Union Générale d'Editions (10-18), Paris (France).

NEWEL, Alan, 1980

Physical symbol systems. *Cognitive Science*, 4, 2, pp.135-83.

OPPO, Franco, 1984

Per una teoria generale del languaggio musicale. In M. Baroni & L. Callegari (Eds.) *Musical Grammars and Computer Analysis*, Olschki, Florence (Italie), pp.115-30.

OSHERSON, Daniel N. & Scott WEINSTEIN, 1982

Criteria of language learning. *Information and Control*, 52, pp.123-38.

OSHERSON, Daniel N., Michael STOB, & Scott WEINSTEIN, 1986
Systems that learn. MIT Press, Cambridge MA (USA).

PASSERON, René, 1984
Pour une approche "poïétique" de la création. In Les Enjeux, *Encyclopaedia Universalis France*, pp.149-57.

PIATTELLI-PALMARINI, Massimo, 1979
Théories du langage, théories de l'apprentissage — le débat entre Jean Piaget et Noam Chomsky. Points 138, Seuil, Paris (France).

PHILIPPOT, Michel, 1970
Muss es sein. In A. Boucourechliev (Ed.) *L'Arc 40: Beethoven*, Aix-en-Provence (France), pp.82-95.

REVESZ, György, 1985
Introduction to Formal Languages. McGraw-Hill Computer Science Series, New York NJ (USA).

RISSET, Jean-Claude, 1989a
Développements récents en informatique et psychoacoustique musicale. In *Etat de la Recherche Musicale (au 1er janvier 1989)*, Agence Régionale pour la Coordination des Activités Musicales et Chorégraphiques, Aix-en-Provence (France), 19 pages, non paginé.

1989b
Computer music experiments 1964-… In C. Roads (Ed.) *The Music Machine*, MIT Press, Cambridge MA (USA), pp.67-74.

ROOZENDAAL, Ron, 1990
Generative processes in music: musical composition. In B. Vecchione et B. Bel (Eds.) *Le Fait Musical — Sciences, Technologies, Pratiques*, actes du Colloque "Musique et Assistance Informatique" (MAI 90), Marseille (France), pp.207-21.

ROADS, Curtis, 1984
An overview of music representations. In M. Baroni & L. Callegari (Eds.) *Musical Grammars and Computer Analysis*, Olschki, Florence (Italie), pp.7-37.

ROUSSEL, A., 1987
Programmation de l'algorithme de Makanin en Prolog II. Mémoire de DEA, Groupe Intelligence Artificielle, Université Marseille II.

RUWET, Nicolas, 1966
Méthodes d'analyse en musicologie. *Revue de musicologie* 20, 1966, pp.65-90. [Réimprimé en 1972 dans *Langage, Musique, Poésie*.]

1972
Langage, Musique, Poésie. Seuil, Paris (France)

SALOMAA, Arto, 1973
Formal Languages. Academic Press, New York NJ (USA).

SCHAEFFER, Pierre, 1966
Traité des objets musicaux. Seuil, Paris (France).

SCHENKER, Heinrich, 1935
Der freie Satz. Universal Edition, Vienne (Autriche).

SCHOTTSTAEDT, B., 1983
PLA: a composer's idea of a language. *Computer Music Journal*, 7, 1, pp.11-20.

SIMON, Herbert A., 1969
The sciences of the artificial. MIT Press, Cambridge MA (USA).

SMAILL, Alan, & Geraint WIGGINS, 1990

Hierarchical music representation for composition and analysis. In B. Vecchione et B. Bel (Eds.) *Le Fait Musical — Sciences, Technologies, Pratiques*, actes du Colloque "Musique et Assistance Informatique" (MAI 90), Marseille (France), pp.261-79.

SMITH, L.C., 1973

Editing and printing music by computing. *Journal of Music Theory*, 17, 2.

1976

SCORE — A musician's approach to computer music. *Journal of the Audio Engineering Society*, 20, 1, pp.7-14.

SMOLIAR, Stephen W., 1971

A parallel processing model of musical structures. Rapport AI TR-242, Dept. of Computing and Information Sciences, MIT, Cambridge MA (USA).

1990a

Composition theory and memory. Résumé soumis au Comité Scientifique, Colloque "Musique et Assistance Informatique" (MAI 90), Marseille (France), Marseille (France).

1990b

The mental processes of musical composition. In B. Vecchione et B. Bel (Eds.) *Le Fait Musical — Sciences, Technologies, Pratiques*, actes du Colloque "Musique et Assistance Informatique" (MAI 90), Marseille (France), pp.171-8.

TODD, Peter M., 1989

A connectionist approach to algorithmic composition. *Computer Music Journal*, 13, 4, pp.27-43.

VAN BENTHEM, J.F.A.K., 1983

The Logic of Time. Reidel, Dordrecht-Boston-Londres.

VECCHIONE, Bernard, 1978

Pour une théorie générale des grammaires musicales formelles. Université de Provence Aix-Marseille I (France).

1984

Pour une science de la réalité musicale. Thèse de Doctorat de IIIème cycle, Université de Provence Aix-Marseille I (France).

1985

La réalité musicale: éléments d'épistémologie musicologique. Thèse de Doctorat d'Etat, Université Paris VIII (France).

1990a

Les sciences et les technologies de la musique — La révolution musicologique des années 1970-1980. In B. Vecchione et B. Bel (Eds.) *Le Fait Musical — Sciences, Technologies, Pratiques*, actes du Colloque "Musique et Assistance Informatique" (MAI 90), Marseille (France), pp.69-128.

1990b

Sciences, technologies et pratiques de la composition — Le raisonnement créateur. In B. Vecchione et B. Bel (Eds.) *Le Fait Musical — Sciences, Technologies, Pratiques*, actes du Colloque "Musique et Assistance Informatique" (MAI 90), Marseille (France), pp.147-70.

1990c

L'analyse métaformalisante. In B. Vecchione et B. Bel (Eds.) *Le Fait Musical — Sciences, Technologies, Pratiques*, actes du Colloque "Musique et Assistance Informatique" (MAI 90), Marseille (France), pp.319-42.

VERCOE, Barry, 1984
The synthetic performer in the context of live performance. *International Computer Music Conference* (ICMC), IRCAM, Paris.

XENAKIS, Iannis, 1963
Musiques formelles. *La Revue Musicale*, Paris (France). Traduction augmentée: *Formalized music*. Indiana University Press, Bloomington (USA), 1971.

1972
Vers une métamusique. In *Architecture et Musique*, Casterman, Paris (France), pp.38-70.

YAVELOW, C., 1986
Music software for the Apple Macintosh. *Computer Music Journal*, 9, 3, pp.52-67.

ZIELONKA, Wieslaw, 1987
Notes on finite asynchronous automata. *RAIRO: Informatique théorique et Applications*, 21, 2, pp.99-135.

Annexes

1. Grammaire d'un *qa'ida* (*grammar of a* qa'ida) — Afaq Husain Khan

L'alphabet terminal du langage (onomatopées rythmiques ou *bol*s) et l'application *miroir*:

dha —> ta —> ta
ti —> ti
ge —> ke —> ke
na —> na
dhee —> tee —> tee
tr —> tr
kt —> kt
- —> -

Les sous-grammaires transformationnelles:

GRAM#1 [1] RND
GRAM#1 [2] <100> S —> (=+ S64 ;)
GRAM#1 [3] <100> S —> ... *Autres longueurs de phrases*

GRAM#2 [1] LIN
GRAM#2 [2] <100> S64 —> L16 + S48
GRAM#2 [3] <100> S64 —> L14 S50
GRAM#2 [4] <100> S64 —> L12 S52
GRAM#2 [5] <100> S64 —> L24 S40
GRAM#2 [6] <100> S48 —> M16 + S32
GRAM#2 [7] <100> S48 —> M14 S34
GRAM#2 [8] <100> S48 —> M40 O8
GRAM#2 [9] <100> S50 —> M18 + S32
GRAM#2 [10] <100> S50 —> M34 + S16
GRAM#2 [11] <100> S50 —> M18 + * (=+ N14) O18
GRAM#2 [12] <100> S16 —> O16
GRAM#2 [13] <100> S50 —> V10 A'8-2 * (= N18 +) + O16
GRAM#2 [14] <100> S50 —> M20 * (= N14 +) + O16
GRAM#2 [15] <100> S52 —> V28 A8-2 O18
GRAM#2 [16] <100> S52 —> M20 + S32
GRAM#2 [17] <100> S40 —> M8 + S32
GRAM#2 [18] <100> S32 —> * (=+ N16 +) + S16
GRAM#2 [19] <100> S32 —> * (= /6+ V12 /4 A8 +) + O16
GRAM#2 [20] <100> S32 —> * (= /6+ A16 V8 + /4) + O16
GRAM#2 [21] <100> S32 —> * (= /6+ V24 + /4) + O16
GRAM#2 [22] <100> S32 —> * (= /8+ N'16 + A16 + /4) + O16
GRAM#2 [23] <100> S32 —> * (=+ N14) O18
GRAM#2 [24] <100> S34 —> O34
GRAM#2 [25] <100> S34 —> * (= N18 +) + S16
GRAM#2 [26] <5> S16 —> /8+ A16 O16 ; /4

GRAM#3 [1] RND
GRAM#3 [2] <100> LEFT M16 —> V16
GRAM#3 [3] <100> LEFT N18 —> V18
GRAM#3 [4] <100> LEFT (=+ L16 —> (=+ A16

```
GRAM#3 [5] <100>  LEFT  (=+ L16  —>  (=+ V16
GRAM#3 [6] <100>  LEFT  (=+ L14  —>  (=+ V10  A'6-2
GRAM#3 [7] <100>  LEFT  M14  —>  A'16-2
GRAM#3 [8] <100>  LEFT  (=+ L14  —>  (=+ A16-2
GRAM#3 [9] <100>  LEFT  (=+ L14  —>  (=+ A'16-2
GRAM#3 [10] <100>  LEFT  (=+ L12  —>  (=+ A16-4
GRAM#3 [11] <100>  LEFT  (=+ L24  —>  (=+ V24
GRAM#3 [12] <100>  LEFT + M16 + * —> + V10  A'6 + *
GRAM#3 [13] <100>  LEFT + M16  —> + A'16
GRAM#3 [14] <100>  LEFT  M8 + * —>  A'8 + *
GRAM#3 [15] <100>  LEFT  M16 + * —>  V8  A'8 + *
GRAM#3 [16] <100>  LEFT  M14  —>  V8  A'8-2
GRAM#3 [17] <100>  LEFT  M18 + * —>  V10  A'8 + *
GRAM#3 [18] <100>  LEFT  M18 + * —>  V18 + *
GRAM#3 [19] <100>  LEFT  M34 +  —>  V28  A'6 +
GRAM#3 [20] <100>  LEFT  M34 +  —>  V26  A'8 +
GRAM#3 [21] <100>  LEFT  M20 + * —>  V12  A'8 + *
GRAM#3 [22] <100>  LEFT  M20  —>  V20
GRAM#3 [23] <100>  LEFT  M40  —>  V8  A'8-2  V26
GRAM#3 [24] <100>  LEFT  * (= /8+ N'16  —>  * (= /8+ V16
GRAM#3 [25] <100>  LEFT  * (= /8+ N'16  —>  * (= /8+ N16
GRAM#3 [26] <100>  LEFT  N16 + —>  V12  A4 +
GRAM#3 [27] <100>  LEFT  N16 + —>  V8  A8 +
GRAM#3 [28] <100>  LEFT  N16 + —>  V10  C6 +
GRAM#3 [29] <100>  LEFT  * (=+ N16  —>  * (=+ A16
GRAM#3 [30] <100>  LEFT  M14  —>  V8  A8-2
GRAM#3 [31] <100>  LEFT  * (=+ N14  —>  * (=+ V8  A8-2
GRAM#3 [32] <100>  LEFT  * (=+ N14  —>  * (=+ A16-2
GRAM#3 [33] <100>  LEFT  N14 + —>  V6  A8 +
GRAM#3 [34] <100>  LEFT  O8 ; —>  A8 ;
GRAM#3 [35] <100>  LEFT  O34 ; —>  V26  A8 ;
GRAM#3 [36] <100>  LEFT  O18 ; —>  V10  A8 ;
GRAM#3 [37] <100>  LEFT  O16 ; —>  V8  A8 ;
GRAM#3 [38] <100>  LEFT + O16 ; —> + A16 ;
-------------------------------------------------------------------------------------------------------------------
 GRAM#4 [1]  LIN
 GRAM#4 [2] <100>  V30  —>  V  V  V28
 GRAM#4 [3] <100>  V28  —>  V  V  V26
 GRAM#4 [4] <100>  V26  —>  V  V  V24
 GRAM#4 [5] <100>  V24  —>  V  V  V  V  V20
 GRAM#4 [6] <100>  V20  —>  V  V  V18
 GRAM#4 [7] <100>  V18  —>  V  V  V16
 GRAM#4 [8] <100>  V16  —>  V  V  V  V  V12
 GRAM#4 [9] <100>  V12  —>  V  V  V10
 GRAM#4 [10] <100>  V10  —>  V  V  V8
 GRAM#4 [11] <100>  V8  —>  V  V  V6
 GRAM#4 [12] <100>  V6  —>  V  V  V  V  V  V
-------------------------------------------------------------------------------------------------------------------
GRAM#5 [1]  LIN
GRAM#5 [2] <100>  ? V  —>  ? -
GRAM#5 [3] <100>  V  —> dha
GRAM#5 [4] <100>  V  V  —> trkt
GRAM#5 [5] <100>  V  V  —> dheena
GRAM#5 [6] <100>  V  V  —> teena
GRAM#5 [7] <100>  V  V  —> dhati
GRAM#5 [8] <100>  V  V  —> gena
GRAM#5 [9] <100>  V  V  —> dhage
GRAM#5 [10] <100> + V  V  —> +tidha
GRAM#5 [11] <100> - V  V  —> -tidha
GRAM#5 [12] <100> kt V  V  —> kttidha
GRAM#5 [13] <100> na V  V  —> natidha
GRAM#5 [14] <100> ge V  V  —> getidha
```

```
GRAM#5 [15] <100> - V  V  —> -ti-
GRAM#5 [16] <100> kt V  V  —> ktti-
GRAM#5 [17] <100> ge V  V  —> geti-
GRAM#5 [18] <100> na V  V  —> nati-
GRAM#5 [19] <100>  V  V  V  —> dhagena
GRAM#5 [20] <100>  V  V  V  —> teenake
GRAM#5 [21] <100>  V  V  V  —> dheenage
GRAM#5 [22] <100>  V  V  V  —> dhatrkt
GRAM#5 [23] <100>  V  V  V  —> trktdha
GRAM#5 [24] <100>  V  V  V  V  —> tidhagena
GRAM#5 [25] <100>  V  V  V  V  —> dhagedheena
GRAM#5 [26] <100>  V  V  V  V  —> teena-ta
GRAM#5 [27] <100>  #ti  V  V  V  V  —>  #ti tidhatrkt
GRAM#5 [28] <100>  V  V  V  V  —> dheenagena
GRAM#5 [29] <100>  V  V  V  V  —> teenakena
GRAM#5 [30] <100>  V  V  V  V  V  —> dhagenadheena
GRAM#5 [31] <100>  V  V  V  V  V  —> dhagenateena
GRAM#5 [32] <100>  V  V  V  V  V  —> dhagenadhati
GRAM#5 [33] <100>  V  V  V  V  V  —> dhatrktdhati
GRAM#5 [34] <100>  V  V  V  V  V  —> dhatidhatrkt
GRAM#5 [35] <100>  V  V  V  V  V  —> dhatidhagena
GRAM#5 [36] <100>  V  V  V  V  V  V  —> dheenagedhatrkt
GRAM#5 [37] <100>  V  V  V  V  V  V  —> genagedhatrkt
GRAM#5 [38] <100>  V  V  V  V  V  V  —> dhagedheenagena
GRAM#5 [39] <100>  V  V  V  V  V  V  —> dhageteenakena
GRAM#5 [40] <100>  #ti  V  V  V  V  V  V  —>  #ti tidhagedheenage
GRAM#5 [41] <100>  V  V  V  V  V  V  —> teenakegenage
GRAM#5 [42] <100>  V  V  V  V  V  V  V  V  —> dhagenagenanagena
GRAM#5 [43] <100>  V  V  V  V  V  V  V  V  —> dhatidhagedheenagena
GRAM#5 [44] <100>  V  V  V  V  V  V  V  V  —> dhatidhageteenakena
GRAM#5 [45] <100>  V  V  V  V  V  V  V  V  A8  —> dhatrktdhatidhatrkt A8
GRAM#5 [46] <100>  V  V  V  V  V  V  V  V  A'8  —> dhatrktdhatidhatrkt A'8
-----------------------------------------------------------------------------------------------------------------
 GRAM#6 [1]  ORD
 GRAM#6 [2]  LEFT + ? ? ? ? ? ? ? ? ? ? A'6-2  —> + ? ? ? ? ? ? ? ? ? ? dhageteena
 GRAM#6 [3]  LEFT + ? ? ? ? ? ? ? ? ? ? C6-2  —> + ? ? ? ? ? ? ? ? ? ? dhagedheena
 GRAM#6 [4]  LEFT + ? ? ? ? ? ? ? ? ? ? A'6-2  —> + ? ? ? ? ? ? ? ? ? ? dhageteena
 GRAM#6 [5]  LEFT + ? ? ? ? ? ? ? ? A8-2  —> + ? ? ? ? ? ? ? ? dhatidhagedheena
 GRAM#6 [6]  LEFT  A8-2  ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ; —> dhatidhagedheena ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ;
 GRAM#6 [7]  LEFT + ? ? ? ? ? ? ? ? A'8-2  —> + ? ? ? ? ? ? ? ? dhatidhageteena
 GRAM#6 [8]  LEFT + ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? A'8-2  —> + ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? dhatidhageteena
 GRAM#6 [9]  LEFT  A'6 + —> dhageteenakena+
 GRAM#6 [10]  LEFT  A'8 + —> dhatidhageteenakena+
 GRAM#6 [11]  LEFT  A4 + —> dheenagena+
 GRAM#6 [12]  LEFT  C6 + —> dhagedheenagena+
 GRAM#6 [13]  LEFT  A8 + —> dhatidhagedheenagena+
 GRAM#6 [14]  LEFT  A8 ; —> dhatidhagedheenagena;
 GRAM#6 [15]  LEFT + A16-4  —> +dhatidhagenadhatrktdhatidhage
 GRAM#6 [16]  LEFT + A16-2  #ge —>+dhatidhagenadhatrktdhatidhagedheena #ge
 GRAM#6 [17]  LEFT + A'16-2  #ke  —>+dhatidhagenadhatrktdhatidhageteena #ke
 GRAM#6 [18]  LEFT + A16  —> +dhatidhagenadhatrktdhatidhagedheenagena
 GRAM#6 [19]  LEFT + A'16  —> +dhatidhagenadhatrktdhatidhageteenakena
-----------------------------------------------------------------------------------------------------------------
```

Gabarits engendrés par cette grammaire:

[1] TEM
[2] (=+.....+.....+ * (=+.....+) +.....;)
[3] (=+.....+.....+ * (=+.....+) +/8+.....;/4;)
[4] (=+.....+.....+ * (=/6+...../4.....+) +.....;)
[5] (=+.....+.....+ * (=/6+.....+/4) +.....;)
[6] (=+.....+.....+ * (=/8+.....+.....+/4) +.....;)
[7] (=+.....+.....+ * (=+.....) .....;)
[8] (=+.....+.....;)
[9] (=+.....+..... * (=.....+) +.....;)
[10] (=+.....+..... * (=.....+) +/8+.....;/4;)
[11] (=+.....+ * (=+.....+) +.....;)
[12] (=+.....+ * (=+.....+) +/8+.....;/4;)
[13] (=+.....+ * (=/6+...../4.....+) +.....;)
[14] (=+.....+ * (=/6+.....+/4) +.....;)
[15] (=+.....+ * (=/8+.....+.....+/4) +.....;)
[16] (=+.....+ * (=+.....) .....;)
[17] (=+.....+.....;)
[18] (=+.....+/8+.....;/4;)
[19] (=+..... * (=.....+) +.....;)
[20] (=+..... * (=.....+) +.....;)
[21] (=+.....;)

2. Exemple d'analyse par le *Bol Processor* BP1 (*example of an analysis by Bol Processor BP1*)

Extrait de l'analyse d'une phrase à l'aide de la grammaire précédente montrant l'utilisation des gabarits et la dérivation canonique contextuelle à droite. Temps d'exécution sur Apple IIc (21 gabarits): 1'43"

Phrase analysée (Ustad. Afaq Husain, fév. 87):

dhatidhage nadhatrkt dhatidhage dheenatrkt dhadhatrkt dhatidha- dhatidhage teena-ta teena-ta titakena tatitake teenakena /8 dhatidhagenadhatrkt dhatidhagedheenagena gena-dhatidhagena dhatidhagedheenagena

Essai des gabarits:

[2](=+dhatidhagenadhatrktdhatidhagedheenatrkt+dhadhatrktdhatidha-dhatidhageteena-ta+*(=+teena-tati takenatatitaketeenakena+)+dhatidhagenadhatrktdhatidhagedheenagena;) ... *gabarit incomplet: échec*

[3] **(=+dhatidhagenadhatrktdhatidhagedheenatrkt+dhadhatrktdhatidha-dhatidhageteena-ta+*(=+teena-tatitakenatatitaketeenakena+)+/8+dhatidhagenadhatrktdhatidhagedheenagenagena-dhatidhagenadhatidhagedheenagena ;/4;)** *gabarit plausible*

Essai d'analyse sur ce gabarit:

(=+dhatidhagenadhatrktdhatidhagedheenatrkt+dhadhatrktdhatidha-dhatidhageteena-ta+*(=+teena-tatitakenatatitaketeenakena+)+/8+A16 gena-dhatidhagenadhatidhagedheenagena;/4;)
(=+A16-2 trkt+dhadhatrktdhatidha-dhatidhageteena-ta+*(=+teena-tatitakenatatitaketeenakena+)+/8+A16 gena-dhatidhagenadhatidhagedheenagena;/4;)
(=+A16-2 trkt+dhadhatrktdhatidha-dhatidhageteena-ta+*(=+teena-tatitakenatatitaketeenakena+)+/8+A16 gena-dhatidhagena A8 ;/4;)
(=+A16-2 trkt+dhadhatrktdhatidha-dhatidhageteena-ta+*(=+teena-tatitakena A8+)+/8+A16 gena-dhatidhagena A8 ;/4;)
(=+A16-2 trkt+dhadhatrktdhatidha- A'8-2 -ta+*(=+teena-tatitakena A8+)+/8+A16 gena-dhatidhagena A8 ;/4;)
(=+A16-2 trkt+dhadhatrktdhatidha- A'8-2 -ta+*(=+teena-tatitakena A8+)+/8+A16 gena- VVVVV A8 ;/4;)
(=+A16-2 trkt+dhadhatrktdhatidha- A'8-2 -ta+*(=+teena-tatitakena A8+)+/8+A16 gena VVVVVV A8 ;/4;)
(=+A16-2 trkt+dhadhatrktdhatidha- A'8-2 -ta+*(=+teena-tatitakena A8+)+/8+A16 VVVVVVVV A8 ;/4;)
(=+A16-2 trkt+dhadhatrktdhatidha- A'8-2 -ta+*(=+teena- VVVVV A8+)+/8+A16 VVVVVVVV A8 ;/4;)
(=+A16-2 trkt+dhadhatrktdhatidha- A'8-2 -ta+*(=+teena VVVVV A8+)+/8+A16 VVVVVVVV A8 ;/4;)
(=+A16-2 trkt+dhadhatrktdhatidha- A'8-2 -ta+*(=+VVVVVVVV A8+)+/8+A16 VVVVVVVV A8 ;/4;)
(=+A16-2 trkt+dhadhatrktdhatidha- A'8-2 V ta+*(=+VVVVVVVV A8+)+/8+A16 VVVVVVVV A8 ;/4;)
(=+A16-2 trkt+dhadhatrktdhatidha V A'8-2 V ta+*(=+VVVVVVVV A8+)+/8+A16 VVVVVVVV A8 ;/4;)
(=+A16-2 trkt+dhadhatrktdhati VV A'8-2 V ta+*(=+VVVVVVVV A8+)+/8+A16 VVVVVVVV A8 ;/4;)
(=+A16-2 trkt+dha VVVVVVV A'8-2 V ta+*(=+VVVVVVVV A8+)+/8+A16 VVVVVVVV A8 ;/4;)
(=+A16-2 trkt+VVVVVVVV A'8-2 V ta+*(=+VVVVVVVV A8+)+/8+A16 VVVVVVVV A8 ;/4;)
(=+A16-2 VV+VVVVVVVV A'8-2 V ta+*(=+VVVVVVVV A8+)+/8+A16 VVVVVVVV A8 ;/4;)
(=+A16-2 VV+VVVVVVVV A'8-2 V ta+*(=+VVVVVVVV A8+)+/8+A16 VVV6 A8 ;/4;)
(=+A16-2 VV+VVVVVVVV A'8-2 V ta+*(=+VVVVVVVV A8+)+/8+A16 V8 A8 ;/4;)
(=+A16-2 VV+VVVVVVVV A'8-2 V ta+*(=+VV V6 A8+)+/8+A16 V8 A8 ;/4;)
(=+A16-2 VV+VVVVVVVV A'8-2 V ta+*(=+V8 A8+)+/8+A16 V8 A8 ;/4;)
(=+A16-2 VV+VVV6 A'8-2 V ta+*(=+V8 A8+)+/8+A16 V8 A8 ;/4;)
(=+A16-2 VV+V8 A'8-2 V ta+*(=+V8 A8+)+/8+A16 V8 A8 ;/4;)
(=+A16-2 VV+V8 A'8-2 V ta+*(=+V8 A8+)+/8+A16 O16 ;/4;)
(=+A16-2 VV+V8 A'8-2 V ta+*(=+N16+)+/8+A16 O16 ;/4;)
(=+A16-2 VV+M14 V ta+*(=+N16+)+/8+A16 O16 ;/4;)
(=+L14 VV+M14 V ta+*(=+N16+)+/8+A16 O16 ;/4;)
(=+L14 VV+M14 V ta+*(=+N16+)+S16 ;)
(=+L14 VV+M14 V ta+S32 ;) ... *échec de l'analyse*

Essai d'autres gabarits:

...

[5] (=+dhatidhagenadhatrktdhatidhagedheenatrkt+dhadhatrktdhatidha-dhatidhageteena-ta+*(=/6+teena-tatita kenatatitaketeenakenatati takenatatrkt+4)+dhatidhagedheenagenagena-dhatidhagena ;) ... *échec*

...

[6] **(=+dhatidhagenadhatrktdhatidhagedheenatrkt+dhadhatrktdhatidha-dhatidhageteena-ta+*(=/8+teenatatitakenatatitaketeenakena+dhatidhagenadhatrktdhatidhagedheenagena+4)+gena-dhatidhagenadhatidhagedheenagena;)** ...*gabarit plausible*

Essai d'analyse sur ce gabarit: échec de l'analyse

Essai d'autres gabarits:

...

[12] **(=+dhatidhagenadhatrktdhatidhagedheenatrktdhadhatrktdhatidha-dhatidhageteena-ta+*(=+teena-tatitakenatatitaketeenakena+)+/8+dhatidhagenadhatrktdhatidhagedheenagenagena-dhatidhagena dhatidhagedheenagena ;/4;)**... *gabarit plausible*

Essai d'analyse sur ce gabarit:

(=+dhatidhagenadhatrktdhatidhagedheenatrktdhadhatrktdhatidha-dhatidhageteena-ta+*(=+teena-tatitakenatatitaketeenakena+)+/8+A16 gena-dhatidhagenadhatidhagedheenagena;/4;)
(=+A16-2 trktdhadhatrktdhatidha-dhatidhageteena-ta+*(=+teena-tatitakenatatitaketeenakena+)+/8+A16gena-dhatidhagenadhatidhagedheenagena;/4;)
(=+A16-2 trktdhadhatrktdhatidha-dhatidhageteena-ta+*(=+teena-tatitakenatatitaketeenakena+)+/8+A16 gena-dhatidhagena A8 ;/4;)
(=+A16-2 trktdhadhatrktdhatidha-dhatidhageteena-ta+*(=+teena-tatitakena A8+)+/8+A16 gena-dhatidhagena A8 ;/4;)
(=+A16-2 trktdhadhatrktdhatidha-dhatidhageteena-ta+*(=+teena-tatitakena A8+)+/8+A16 gena- VVVVV A8 ;/4;)
(=+A16-2 trktdhadhatrktdhatidha-dhatidhageteena-ta+*(=+teena-tatitakena A8+)+/8+A16 gena VVVVVV A8 ;/4;)
(=+A16-2 trktdhadhatrktdhatidha-dhatidhageteena-ta+*(=+teena-tatitakena A8+)+/8+A16 VVVVVVVV A8 ;/4;)
(=+A16-2 trktdhadhatrktdhatidha-dhatidhageteena-ta+*(=+teena- VVVVV A8+)+/8+A16 VVVVVVVV A8 ;/4;)
(=+A16-2 trktdhadhatrktdhatidha-dhatidhageteena-ta+*(=+teena VVVVVV A8+)+/8+A16 VVVVVVVV A8 ;/4;)
(=+A16-2 trktdhadhatrktdhatidha-dhatidhageteena-ta+*(=+VVVVVVVV A8+)+/8+A16 VVVVVVVV A8 ;/4;)
(=+A16-2 trktdhadhatrktdhatidha-dhatidhage VVVV+*(=+VVVVVVVV A8+)+/8+A16 VVVVVVVV A8 ;/4;)
(=+A16-2 trktdhadhatrktdhatidha-dhati VVVVVV+*(=+VVVVVVVV A8+)+/8+A16 VVVVVVVV A8 ;/4;)
(=+A16-2 trktdhadhatrktdhatidha- VVVVVVVV+*(=+VVVVVVVV A8+)+/8+A16 VVVVVVVV A8 ;/4;)
(=+A16-2 trktdhadhatrktdhatidha VVVVVVVVV+*(=+VVVVVVVV A8+)+/8+A16 VVVVVVVV A8 ;/4;)
(=+A16-2 trktdhadhatrktdhati VVVVVVVVVV+*(=+VVVVVVVV A8+)+/8+A16 VVVVVVVV A8 ;/4;)
(=+A16-2 trktdha VVVVVVVVVVVVVV+*(=+VVVVVVVV A8+)+/8+A16 VVVVVVVV A8 ;/4;)
(=+A16-2 VVVVVVVVVVVVVVVVV+*(=+VVVVVVVV A8+)+/8+A16 VVVVVVVV A8 ;/4;)
(=+A16-2 VVVVVVVVVVVVVVVVV+*(=+VVVVVVVV A8+)+/8+A16 VVV6 A8 ;/4;)
(=+A16-2 VVVVVVVVVVVVVVVVV+*(=+VVVVVVVV A8+)+/8+A16 V8 A8 ;/4;)
(=+A16-2 VVVVVVVVVVVVVVVVV+*(=+VV V6 A8+)+/8+A16 V8 A8 ;/4;)
(=+A16-2 VVVVVVVVVVVVVVVVV+*(=+V8 A8+)+/8+A16 V8 A8 ;/4;)
(=+A16-2 VVVVVVVVVVV V6+*(=+V8 A8+)+/8+A16 V8 A8 ;/4;)
(=+A16-2 VVVVVVVVV V8+*(=+V8 A8+)+/8+A16 V8 A8 ;/4;)
(=+A16-2 VVVVVVV V10+*(=+V8 A8+)+/8+A16 V8 A8 ;/4;)
(=+A16-2 VVVVV V12+*(=+V8 A8+)+/8+A16 V8 A8 ;/4;)
(=+A16-2 VV V16+*(=+V8 A8+)+/8+A16 V8 A8 ;/4;)
(=+A16-2 V18+*(=+V8 A8+)+/8+A16 V8 A8 ;/4;)
(=+A16-2 V18+*(=+V8 A8+)+/8+A16 O16 ;/4;)
(=+A16-2 V18+*(=+N16+)+/8+A16 O16 ;/4;)
(=+A16-2 M18+*(=+N16+)+/8+A16 O16 ;/4;)
(=+L14 M18+*(=+N16+)+/8+A16 O16 ;/4;)
(=+L14 M18+*(=+N16+)+S16 ;)
(=+L14 M18+S32 ;)
(=+L14 S50 ;)
(=+S64 ;)
S ... *Succès de l'analyse*

Essai des autres gabarits, etc...

3. Algorithme d'interprétation de formules polymétriques (*algorithm for polymetric structure interpretation*)

(Voir chapitre VIII §5)

```
Programme principal

   Constantes:
   MAXBUF = espace de travail

   Entrée:
   A[MAXBUF]

   Sorties:
   B[MAXBUF], P/Q, Prod, Ratio

Début
   s <— oldscale <— 1;
   pos <— 0;
   P/Q <— 0/1;
   fixtempo <— faux;
   localtempo <— 1;
   Si (POLY(A,B,pos,P,Q,localtempo,fixtempo,oldscale,s)[262] = échec)
   alors
   |     Afficher "Interprétation impossible";
   |     Stop;
   Finsi
   Calculer Prod = ppcm des tempi en B;
   Ratio <— Prod / s;
Fin


Procédure POLY(A,B,pos,localtempo,fixtempo,oldscale,s)

   MAXSEQ = majorant des longueurs des champs de A
   MAXDEPTH = majorant du nombre de champs de A

   C[MAXDEPTH][MAXSEQ], p[MAXDEPTH], q[MAXDEPTH], scale[MAXDEPTH], vargap[MAXDEPTH],
      pgap[MAXDEPTH], qgap[MAXDEPTH], fixtempo[MAXDEPTH], pp[MAXDEPTH], r[MAXDEPTH],
      E[MAXBUF]

Début
   pgap[1]/qgap[1] <— 0/1;
   oldpos <— pos;
   restart <— faux;

Analyse:
   a <— 1;
   pos <— oldpos;
   scale[a] <— oldscale;  /* a = 1 */
   p[a]/q[a] <— 0/1;
   singlegap <— tempomark <— newk <— newg <— just_fill_gap <— vargap[a] <— fixtempo[a]
      <— faux;
   g <— h <— 0;
   tempo <— localtempo;
   Ecrire "/tempo" après C[a][];
```

[262] Dans tous les appels de procédures ou de fonctions nous supposons que le passage de paramètres se fait par *valeurs* (et non par *adresses* comme en langage C). C'est pourquoi aucun pointeur ne figure comme argument d'une fonction ou procédure.

```
Pour (i = pos; A[i] ≠ fin de ligne)
|     Si (non tempomark et A[i] = "/")
|     alors
|     |     tempomark <— vrai;
|     sinon
|     |     Si (A[i] est un chiffre x)
|     |     alors
|     |     |     Si (tempomark)
|     |     |     alors
|     |     |     |     h <— 10*h + x;      /* Ce nombre indique un nouveau tempo */
|     |     |     |     newk <— vrai;
|     |     |     sinon
|     |     |     |     g <— 10*g + x;      /* Ce nombre indique un silence */
|     |     |     newg <— vrai;
|     |     |     Finsi
|     |     sinon
|     |     |     tempomark <— faux;
```

Correction_tempo:

```
|     |     |     Si (newk)
|     |     |     alors
|     |     |     |     newk <— faux;
|     |     |     |     Si (newg)
|     |     |     |     alors
|     |     |     |     |     singlegap <— vrai; /* On change de tempo localement pour
|     |     |     |     |        un */
|     |     |     |     |     oldtempo <— tempo; /* …silence fractionnaire */
|     |     |     |     |     tempo <— tempo * h;
|     |     |     |     sinon
|     |     |     |     |     tempo <— h * scale[a];   /* On change de tempo:
|     |     |     |     |        indication explicite */
|     |     |     |     |     fixtempo[a] <— vrai;
|     |     |     |     |     fixtempo <— vrai;  /* Information qui sera propagée aux
|     |     |     |     |        sous-structures */
|     |     |     |     Finsi
|     |     |     |     Si (non newg) ...
|     |     |     |     /* Procédure facultative permettant d'effacer la marque de
|     |     |     |        tempo précédente le cas échéant. */
|     |     |     |     Finsi
|     |     |     |     Ecrire "/tempo" après C[a][];
|     |     |     Finsi
|     |     |     Si (newg)
|     |     |     alors
|     |     |     |     newg <— faux;
|     |     |     |     Ecrire g fois "-" après C[a][];
|     |     |     |     /* On a remplacé un entier par le même nombre de silences
|     |     |     |        élémentaires "-" */
|     |     |     |     g <— 0;
|     |     |     |     p[a]/q[a] <— p[a]/q[a] + g*scale[a]/tempo;
|     |     |     |     Si (singlegap)   /* Silence fractionnaire */
|     |     |     |     alors
|     |     |     |     |     singlegap <— faux;
|     |     |     |     |     tempo <— oldtempo;
|     |     |     |     |     Ecrire "/tempo" après C[a][];
|     |     |     |     Finsi
|     |     |     Finsi
|     |     |     Si (just_fill_gap)
|     |     |     alors
|     |     |     |     just_fill_gap <— faux;
|     |     |     |     i <— i+1;
|     |     |     |     Continuerpour;
|     |     |     Finsi
|     |     |     Si( A[i] = "…")  /* Silence indéterminé */
|     |     |     alors
|     |     |     |     Si (restart)  /* On analyse la structure pour la seconde fois
|     |     |     |        */
|     |     |     |     alors
```

```
|     |     |     |     |     newk <— newg <— vrai;
|     |     |     |     |     h <— qgap[a] * scale[a];
|     |     |     |     |     g <— tempo * pgap[a];
|     |     |     |     |     just_fill_gap <— vrai;
|     |     |     |     |     aller à Correction_tempo;
|     |     |     |     sinon
|     |     |     |     |     Si (non vargap[a])
|     |     |     |     |     alors
|     |     |     |     |     |     vargap[a] <— vrai;
|     |     |     |     |     sinon
|     |     |     |     |     |     Afficher "Erreur: pas plus d'un silence indéterminé
|     |     |     |     |     |        par argument."
|     |     |     |     |     |     Retour (échec)
|     |     |     |     |     Finsi
|     |     |     |     Finsi
|     |     |     Finsi
|     |     |     Si (A[i] = "{")
|     |     |     alors
|     |     |     |     Si (POLY (A,E,i+1,p[a],q[a],tempo,fixtempo[a],scale[a],s) =
|     |     |     |        échec)
|     |     |     |     alors
|     |     |     |     |     Retour (échec);
|     |     |     |     Finsi
|     |     |     |     Si (s > 1)
|     |     |     |     alors
|     |     |     |     |     tempo <— s * tempo;
|     |     |     |     |     scale[a] <— s * scale[a];
|     |     |     |     |     Ecrire "\s" après C[a][];
|     |     |     |     |     /* Marque de changement d'échelle */
|     |     |     |     Finsi
|     |     |     |     Ecrire "{" après C[a][];
|     |     |     |     Copier E[] après C[a][];
|     |     |     |     Ecrire "}" après C[a][];
|     |     |     |     Ecrire "/tempo" après C[a][];
|     |     |     Finsi
|     |     |     Si (A[i] = "}")
|     |     |     alors
|     |     |     |     pos <— i;
|     |     |     |     aller à Résolution;
|     |     |     Finsi
|     |     |     Si (A[i] = ",")
|     |     |     alors
|     |     |     |     Ecrire fin de ligne après C[a][];
|     |     |     |     a <— a + 1;
|     |     |     |     p[a]/q[a] <— 0/1;
|     |     |     |     Si (non restart)
|     |     |     |     alors
|     |     |     |     |     pgap[a]/qgap[a] <— 0/1;
|     |     |     |     Finsi
|     |     |     |     vargap[a] <— fixtempo[a] <— faux;
|     |     |     |     scale[a] <— oldscale;
|     |     |     |     tempo <— localtempo;
|     |     |     |     Ecrire "/tempo" après C[a][];
|     |     |     Finsi
|     |     |     Si (A[i] est un terminal "z")
|     |     |     alors
|     |     |     |     p[a]/q[a] <— p[a]/q[a] + scale[a]/tempo;
|     |     |     |     Ecrire "z" après C[a][];
|     |     |     Finsi
|     |     Finsi
|     Finsi
|     i <— i+1;
Finpour
Ecrire fin de ligne après C[a][];
```

Résolution:

```
k <— a;
```

```
fixlength <— restart <— faux;

/* La durée par défaut est celle du premier argument */
a0 <— 0; pmax <— p[0];  qmax <— q[0];

L <— 1;
Pour (a = 1;  a ≤ k)
|     Si (vargap[a])
|     alors
|     |     a <— a+1;
|     |     Continuerpour;
|     Finsi
|     L <— ppcm(L, p[a]);
|     Si (fixtempo[a])
|     alors
|     |     Si ((p[f]/q[f] ≠ pmax/qmax) et fixlength)
|     |     alors
|     |     |     Afficher "Erreur: conflit de durées."
|     |     |     Retour (échec);
|     |     sinon
|     |     |     fixlength <— 1;
|     |     |     pmax <— p[a]; qmax <— q[a]
|     |     |     a0 <— a;
|     |     |     fixtempo <— 1
|     |     Finsi
|     Finsi
|     a <— a+1;
Finpour
```

Silences indéterminés:

```
Pour (a = 1;  a ≤ k)
|     Si (vargap[a])
|     |     restart <— vrai;
|     |     pgap[a]/qgap[a] <— pmax/qmax - p[a]/q[a];
|     |     Si (pgap[a] < 0)
|     |     alors
|     |     |     Si (fixtempo[a])
|     |     |     alors
|     |     |     |     Afficher "Erreur: pas assez de temps pour insérer un silence"
|     |     |     |     Retour (échec);
|     |     |     sinon
|     |     |     |     m <— Partie entière de (p[a].qmax / q[a].pmax);
|     |     |     |     pgap[a]/qgap[a] <— pmax/qmax - p[a]/(m.q[a]);
|     |     |     Finsi
|     |     Finsi
|     Finsi
|     a <— a+1;
Finpour

Si(restart)
alors
|     aller à Analyse;
Finsi

P/Q <— P/Q + pmax/qmax;
```

Calcul de r[a]:

```
M <— 1;
Pour (a = 1;  a ≤ k)
|     pp[a] <— L / p[a];
|     M <— ppcm(M, q[a] * pp[a]);
|     a <— a+1;
Finpour
Pour (a = 1;  a ≤ k)
|     r[a] <— M / (q[a] * pp[a]);
|     a <— a+1;
```

```
Finpour
```

Corrections d'échelle:

```
/* On relit la formule de droite à gauche en effectuant les corrections d'échelle
   marquées "\x" */

s <— 1;
Pour (a = 1;  a ≤ k)
|     Pour (i = longueur de C[a][];  i ≥ 0)
|     |     Si (C[a][i] = "\")
|     |     alors
|     |     |     s <— s * C[a][i+1];
|     |     |     i <— i-1;
|     |     Sinon
|     |     |     Si (C[a][i] = "/")
|     |     |     alors
|     |     |     |     C[a][i+1] <— s * C[a][i+1];
|     |     |     |     i <— i-1;
|     |     |     Finsi
|     |     Finsi
|     |     i <— i-1;
|     Finpour
Finpour
```

Calcul de s:

```
/* On calcule ss tel que pour toute indication de  tempo x dans C[a][] l'expression:
   <ss * x * r[a] * oldscale / scale[a]>  soit entière. */

ss <— 1;
Pour (a = 1;  a ≤ k)
|     Pour (i = 0;  i ≤ longueur de C[a][])
|     |     Si (C[a][i] = "/" et C[a][i+1] = "x")
|     |     alors
|     |     |     ss <— ppcm (ss, scale[a] / pgcd(x * r[a] * oldscale, scale[a]));
|     |     |     i <— i+1;
|     |     Finsi
|     |     i <— i+1;
|     Finpour
|     a <— a+1;
Finpour

s <— ss * r[a0];
```

Copie dans B:

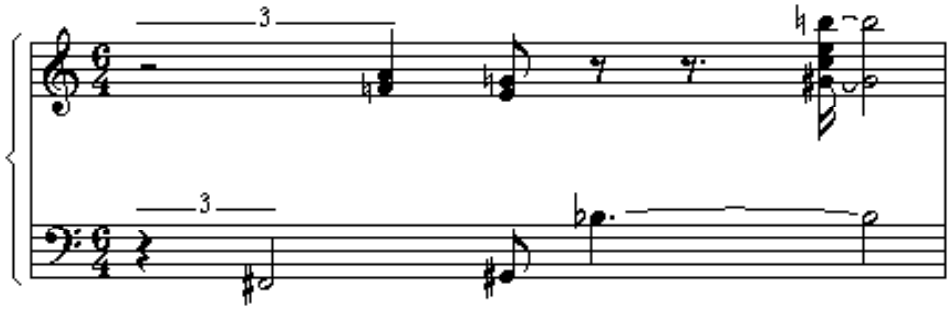
```
/* On recopie la formule dans B[] */

j <— 0;
```

```
   Pour (a = 1;  a ≤ k)
   |     Pour (i = 0;  i ≤ longueur de C[a][])
   |     |     Si (C[a][i] = "/" et C[a][i+1] = "x")
   |     |     alors
   |     |     |     x' <— ss * x * r[a] * oldscale / scale[a];
   |     |     |     B[j] <— "/"; B[j+1] <— "x'";
   |     |     |     j <— j+2;
   |     |     |     i <— i+1;
   |     |     sinon
   |     |     |     B[j] <— C[a][i];    /* On recopie simplement C[][] dans B */
   |     |     |     j <— j+1;
   |     |     Finsi
   |     |     i <— i+1;
   |     Finpour
   |     Ecrire "," après B[];
   |     a <— a+1;
   Finpour
   Supprimer dernière "," de B[];
   Ecrire fin de ligne après B;
   Retour (succès);
Fin
```

4. Exemple de traitement de formule polymétrique en notation tonale (*example of polymetric structure processing in staff notation*)

(Voir chapitre VIII)

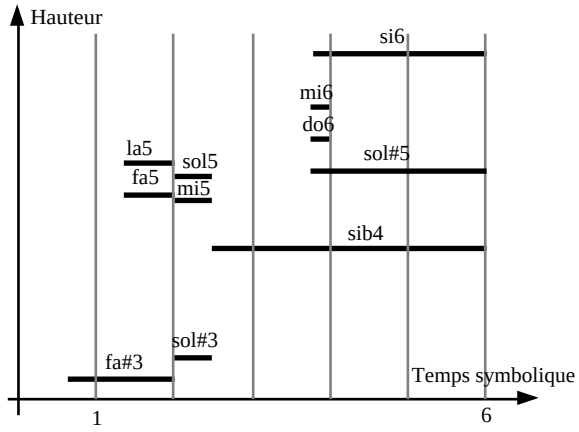
L'exemple suivant est proposé par Ames[263]:



Représentation 1: notation sur portée

Une représentation “physique” des objets sonores correspondants est la suivante:

[263] 1989, p.2



Représentation 2: diagramme

Dans COMPOSE (comme dans la plupart des logiciels de type “séquenceur”) on représente ces objets sous forme d'une table[264]:

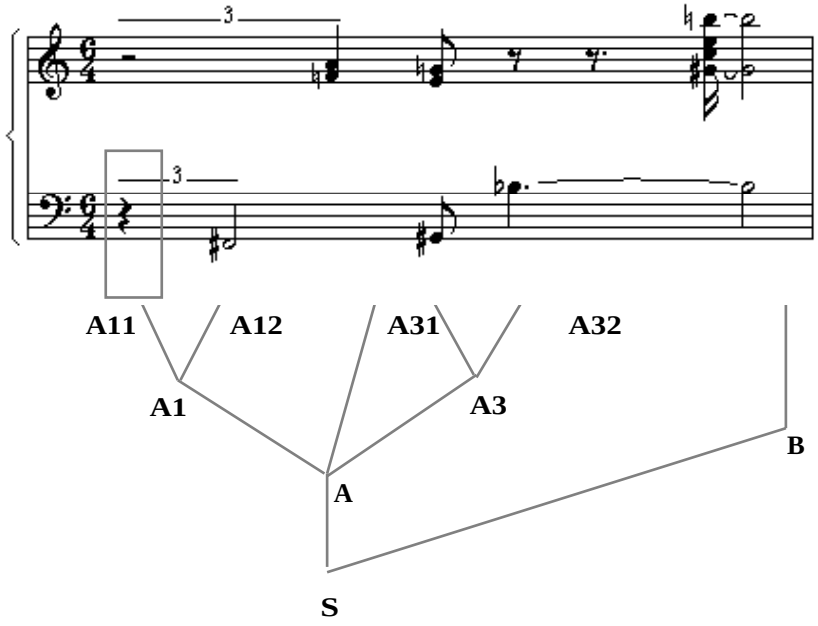
Période	Durée	Hauteur
.667	.667	R *[Silence]*
.667	1.333	fa#3
.667	.667	fa5:la5
.5	.5	sol#3:mi5:sol5
1.25	3.5	sib4
0	2.25	do6:mi6
2.25	.25	sol#5:si6

Représentation 3: table

où *Période* désigne le temps symbolique écoulé entre le déclenchement d'un groupe d'objets (représenté sur une même ligne) et celui du groupe suivant.

Il est assez difficile de passer de la représentation 1 à la représentation 3 sans disposer de celle en diagramme. Il n'y a en effet aucun lien conceptuel entre ces deux représentations, la dernière étant de type atomiste et la première suggérant des regroupements possibles en structure arborescente. Plusieurs analyses sont possibles pour cette phrase musicale prise hors de son contexte. Nous en proposons une ci-dessous, choisie moins pour sa pertinence que pour illustrer la représentation polymétrique:

[264] *op.cit.* p.3. Une erreur sur les durées (lignes 2 et 3) a été corrigée. Nous avons traduit les notes en notation italienne.



Un exemple de regroupement arborescent

Cette analyse correspond à la grammaire

S —> {A,… B}
A —> {2,A1,2 A2} {4,A3}
A1 —> {A11 …, - A12}
A3 —> A31 A32 [ou, de manière équivalente, A3 —> {A31 A32}]
A11 —> - [même remarque]
A12 —> {2,fa#3}
A2 —> {fa5,la5}
A31 —> {1/2,sol#3,mi5,sol5}
A32 —> {3/2,sib4&} {2,&sib4}
B —> {1/4,sol#5&,do6,mi6,si6&} {2,&sol#5,&si6}

dont l'unique dérivation est:

{{2, {- …, {2, fa#3}}, 2 {fa5, la5}} {4, {1/2, sol#3, mi5, sol5} {3/2, sib4&} {2, &sib4}}, … {1/4, sol#5&, do6, mi6, si6&} {2, &sol#5, si6}}

Dans cette représentation, “sol#5&” désigne le début de l'objet sonore “sol#5”, et “&sol#5” la fin du même objet. “&” est donc un opérateur de concaténation. On peut constater que l'information sur les durées est redondante dans la mesure où: 4 = 1/2 + 3/2 + 2.

Après résolution on obtient la formule dilatée:

{{{-_ _ _ _ _ _ _ {fa#3_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ }}, -_ _ _ _ _ _ _ -_ _ _ _ _ _ _ {fa5_ _ _ _ _ _ _, la5_ _ _ _ _ _ _ }}{{sol#3_ _ _ _ _, mi5_ _ _ _ _, sol5_ _ _ _ _ }{sib4_ &}{&sib4_ }}, -_ _ {sol#5_ _ &, do6_ _, mi6_ _, si6_ _ &}{&sol#5_ _, &si6_ }}

La formule réduite correspondante est

/12 {/12 {/6 {/18 -, {/9 fa#3}/18} /6, /18 - - {/18 fa5, /18 la5}/18}/12 {/12 {/24 sol#3, /24 mi5, /24 sol5} /12 {/8 sib4&} /12 {/6 &sib4} /12} /12, /12 /48 - - - - - - - - - - - - - - - - /12 {/48 sol#5&, /48 do6, /48 mi6, /48 si6&} /12 {/6 &sol#5, /6 si6} /12} /12

ce qui donne (pour Prod=144) la formule minimale:

{{{/18 -, {/9 fa#3}}, /18 - - {fa5, la5}}{{/24 sol#3, mi5, sol5} {/8 sib4&} {/6 &sib4} }, 15/48 {/48 sol#5&, do6, mi6, si6&} {/6 &sol#5, si6}}

5. Exemples de mise en temps de structures (*examples of time-setting of structures*)

(Voir chapitre IX)

5.1 Exemples d'objets sonores non vides (*examples of non-empty sound-objects*)

(Voir chapitre IX §4-6)

Le tableau ci-dessous contient les données de six prototypes d'objets sonores non vides striés: “a”, “a'”, “b”, “c”, “d”, “e”. L'objet “a'” est identique à “a” en ce qui concerne la séquence de messages, mais ses propriétés sont différentes.

La première ligne du tableau est la durée (en secondes) du prototype. Les périodes de référence Tref(j) (correspondant à la période du métronome) sont données sur la deuxième ligne. Les durées (symboliques) des prototypes d'objets sonores “a” et “a'” sont de 1 unité de temps, c'est à dire identiques pour un tempo donné, bien que la saisie des prototypes ait été faite à des vitesses différentes[265]. De la même manière, “b” et “c” durent 2 unités de temps, et “d” et “e” 1/2 unité de temps.

	a	a'	b	c	d	e
Dur(j)	1.0	1.5	2.0	4.0	1.0	0.5
Tref(j)	1.0	1.5	1.0	2.0	2.0	1.0
Properties	PivBeg OkRescale OverBeg OverEnd TruncEnd	PivBeg OkExpand Reloc BrkTempo TruncEnd	PivBeg OkRescale Reloc OverEnd TruncBeg	PivBeg OkRescale OverBeg OverEnd	PivEnd OkRescale OverBeg OverEnd	PivCent OkRescale OverBeg OverEnd

Dans certains exemples nous prendrons des objets “a”, “b”, “c”, etc., semblables aux précédents mais **lisses**, autrement dit sans pivot et avec des périodes de référence nulles.

[265] L'intérêt d'une saisie avec des références de temps distinctes est de disposer d'une quantification des dates (*quantization*) adaptée aux besoins de l'écriture musicale, autrement dit de pouvoir assigner un sous-multiple de la période de référence aux dates des actions élémentaires de type ON.

5.2 Calcul de $\alpha(k)$ et positionnement d'objets en séquence

(Voir chapitre IX §7)

Considérons la formule polymétrique

/2 abc /3 de

qui s'écrit en représentation dilatée

/6 a _ _ b _ _ c _ _ d _ e _

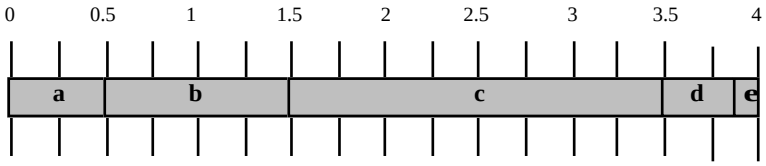
avec un facteur d'échelle Ratio = 6. On est ramené à une séquence de 14 symboles:

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
E_k	a	_	_	b	_	_	c	_	_	d	_	e	_	NIL

5.2.1 Objets lisses ou striés en temps lisse non mesuré

On adopte la configuration $\gamma(X,Y)$ pour les intervalles supports temporels de deux objets non vides consécutifs (séquence stricte).

L'interprétation de l'exemple dans ce cas est



/2 a b c /3 d e
(temps lisse non mesuré, objets lisses ou striés)

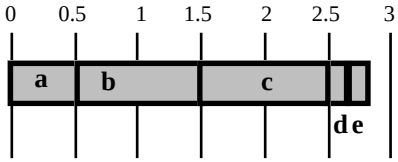
avec $\alpha(k)$ = 0,5 pour “a”, “b” et “c”, et 0,33 pour “d” et “e”.

5.2.2 Séquence d'objets striés en temps lisse mesuré

Prenons Tclock = 1s. L'exemple précédent donne:

	a	b	c	d	e
d(k)	1/2	1/2	1/2	1/3	1/3
D(j)	(1s)	(2s)	(4s)	(1s)	(0,5s)
Tref(j)	1	1	2	2	1
$\alpha(k)$	0,5	0,5	0,25	0,166	0,33
durée physique	0,5s	1s	1s	0,16s	0,16s

(Les données entre parenthèses ne sont pas utilisées pour le calcul.)

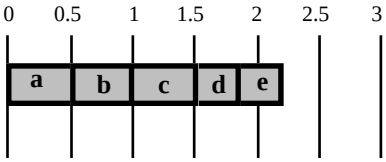


/2 a b c /3 d e
(temps lisse mesuré, objets striés)

5.2.3 Séquence d'objets lisses en temps lisse mesuré

On obtient cette fois:

	a	b	c	d	e
d(k)	1/2	1/2	1/2	1/3	1/3
D(j)	1s	2s	4s	1s	0,5s
α(k)	0,5	0,25	0,125	0,33	0,66
durée physique	0,5s	0,5s	0,5s	0,33s	0,33s



/2 a b c /3 d e
(temps lisse mesuré, objets lisses)

5.2.4 Séquence d'objets mixtes en temps lisse

On calcule α(k) comme précédemment pour chaque objet sonore, puis on positionne les objets en séquence stricte.

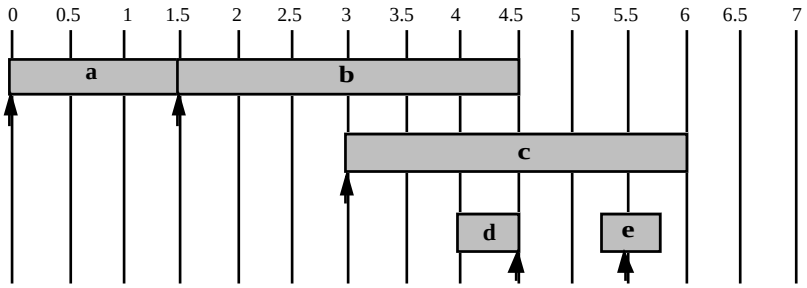
5.2.5 Objets striés en temps strié

Considérons la mise en temps sur un battement métronomique de période 3 secondes. La structure du temps est une progression arithmétique de raison 3000/6 = 500ms. Le tableau ci-dessous donne les valeurs de T(i) en millisecondes, et les objets correspondants E_k de la séquence:

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
T(i)	0	500	1000	1500	2000	2500	3000	3500	4000	4500	5000	5500	6000	6500
E_k	a	_	_	b	_	_	c	_	_	d	_	e	_	NIL
k	1	0	0	2	0	0	3	0	0	4	0	5	0	-1

Pour l'objet “b”, par exemple, on a: i = 4, inext = 7, et Tref(j) = 2s. Δ(i) = 0 pour tout *i*. On trouve donc α(2) = (3000-1500) / 1000 = 1,5. La durée de cet objet est donc: 1,5 . 2 = 3s.

Pour positionner les objets sur les stries on tient compte des propriétés PivBeg pour “a”, “b” et “c”, PivEnd pour “d” et PivCent pour “e”. On obtient l'agencement suivant (échelle graduée en secondes):



/2 a b c /3 d e = /6 a _ _ b _ _ c _ _ d _ e _
(temps strié métronomique, objets striés, période 3s.)

Il est facile de constater que cet agencement est compatible avec les propriétés topologiques des objets.

5.3 Positionnement d'objets dans des structures polymétriques

(Voir chapitre IX §8)

Soit la formule

/3 ab { cde, ab } cd

qui est interprétée

/3 ab { /3 cde, /2 ab } /3 cd

soit, en représentation dilatée

/6 a _ b _ { c_ d _ e _ , a _ _ b _ _ } c _ d _

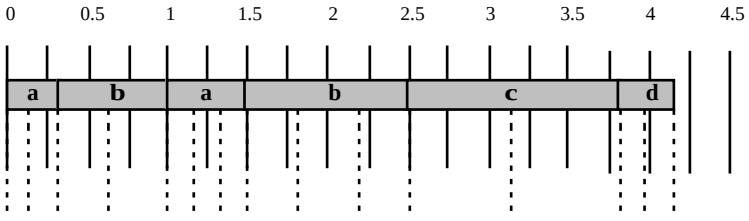
et sous forme de tableau (avec un facteur d'échelle Ratio = 6):

a	_	b	_	a	_	_	b	_	_	c	_	d	_	NIL
_	_	_	_	c	_	d	_	e	_	NIL	_	_	_	_

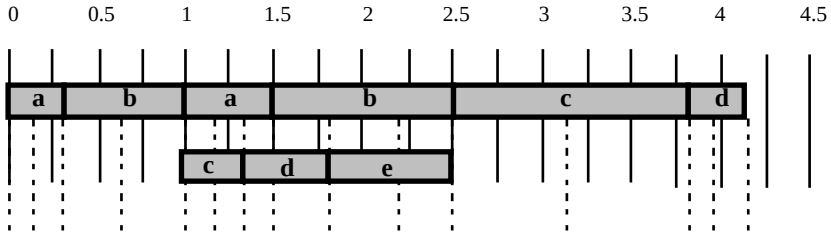
5.3.1 Temps lisse non mesuré, objets lisses

La mise en temps de la première séquence (“ababcd”) ne présente pas de difficulté. La deuxième séquence doit respecter la coïncidence entre “c” et la deuxième occurrence de “a” dans la première séquence, ainsi que (pour les objets qui possèdent la propriété OkRescale) les proportions imposées par la première séquence.

A cet effet, on crée des stries par interpolation des durées physiques des objets non vides de la première séquence:



puis on “met en place” les séquences suivantes sur ces stries:



/3 ab { cde , ab } cd = /6 a _ b _ { c _ d _ e _ , a _ _ b _ _ } c _ d _

(temps lisse non mesuré, objets lisses)

Dans la première séquence on a toujours $\alpha(k) = d(i)$, autrement dit la dilatation de chaque objet est égale à sa durée symbolique. Dans les séquences suivantes, par contre, cette proportion n'est pas respectée, voir par exemple les durées de “c” et “d”. Par contre, les coïncidences des dates de fins d'objets (ici “b” et “e”) sont respectées.

L'interprétation de cet exemple n'a pas créé de recouvrement d'objets appartenant à la même séquence. Des recouvrements peuvent se produire lorsque certains objets ne sont pas compressibles.

5.3.2 Temps strié

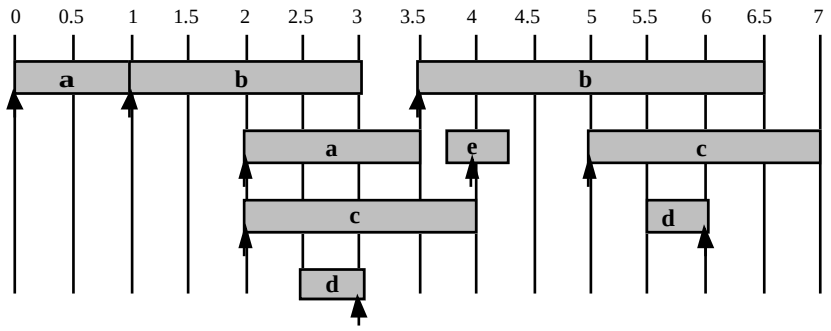
Nous illustrons l'influence des propriétés des objets par trois exemples:

/3 ab { cde , ab} cd
= /3 ab { /3 cde , /2 ab} /3 cd
= /6 a _ b _ { c _ d _ e _ , a _ _ b _ _ } c _ d _

/3 a'b { cde , a'b} cd
= /3 a'b { /3 cde , /2 a'b} /3 cd
= /6 a' _ b _ { c _ d _ e _ , a' _ _ b _ _ } c _ d _

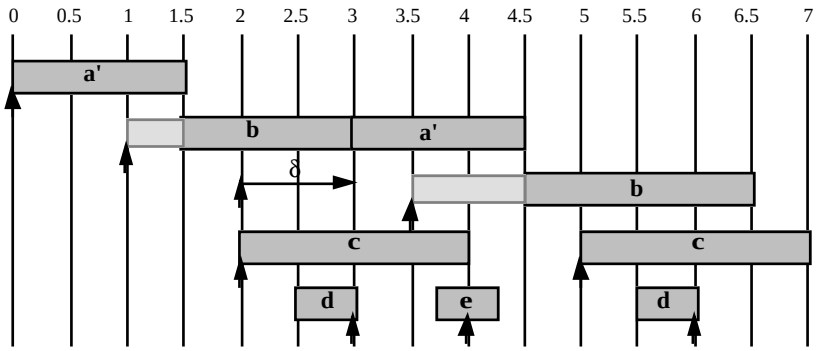
{b - ba, /2 a'c {de, /3 a - c}}
= {/2 b - ba, /2 a'c {/2 de, /3 a - c}}
= /6 {b _ _ - _ _ b _ _ a _ _ , a' _ _ c _ _ {d _ _ e _ _ , a_ - _ c _ }}

Les deux premiers aboutissent à la même représentation symbolique mais à des interprétations différentes même avec un tempo identique (période de métronome = 3 secondes):



/3 ab { ab, cde } cd

(temps métronomique, période 3 sec.)

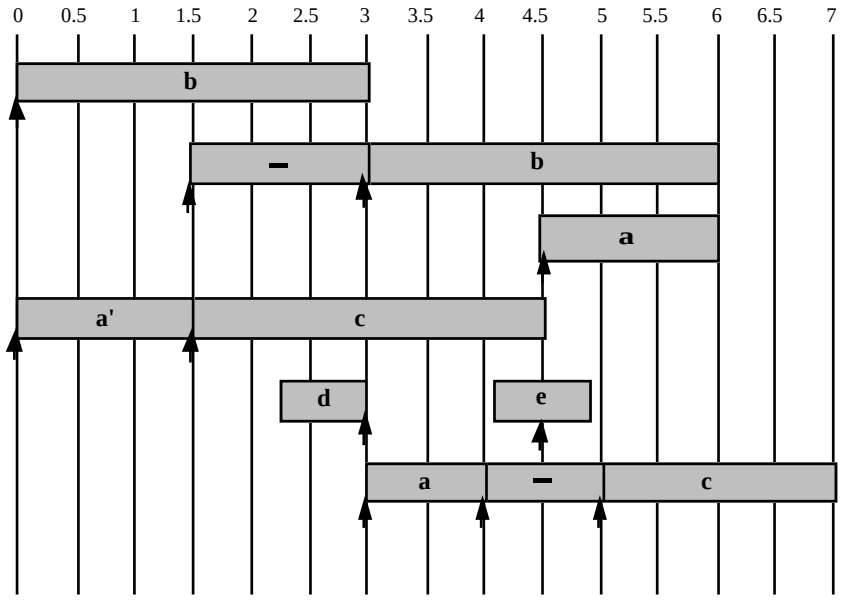


/3 a'b { cde , a'b } cd =
/6 a' _ b _ { a' _ _ b _ _ , c _ d _ e _ } c _ d _

(temps métronomique, période 3 sec.)

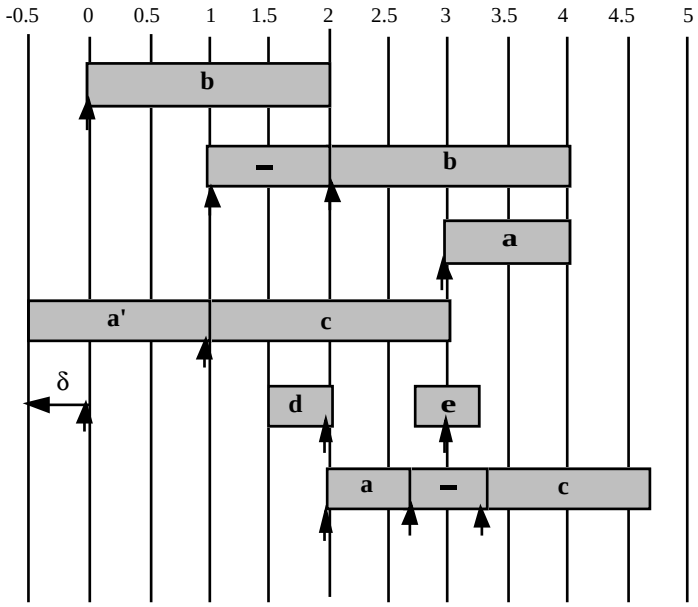
Dans le deuxième exemple, le fait que “a'” ne possède pas la propriété OkCompress empêche la mise à l'échelle de sa durée ($\alpha(k) = 0{,}66$). Le système prend donc la valeur $\alpha(k) = 1$, qui donne la durée de 1,5 secondes. “b” devrait donc commencer avant que “a'” ne soit terminé, mais “b” ne possède pas la propriété OverBeg. On met donc à profit la propriété TruncBeg pour tronquer la partie commençante de “b”. L'objet “a'” ne possèdant pas non plus la propriété OverBeg, la deuxième occurrence de “a'” ne peut donc commencer avant la fin de “b”. On met ici à profit la propriété Reloc qui autorise un décalage local de “a'”. Le problème du recouvrement de “b” se repose alors et peut être résolu de nouveau en tronquant la partie commençante.

Pour le troisième exemple, les interprétations avec une période de métronome de 3 puis de 2 secondes mettent en évidence la propriété “non OkCompress” de “a'”:



{b - ba, /2 a'c {de, /3 a - c}}
= /6 { b _ _ - _ _ b _ _ a _ _ , a' _ _ c _ _ { d _ _ e _ _ , a _ - _ c _ }}

(temps métronomique, période 3 sec.)



{b - ba, /2 a'c {de, /3 a - c}}
= /6 { b _ _ - _ _ b _ _ a _ _ , a' _ _ c _ _ { d _ _ e _ _ , a _ - _ c _ }}

(temps métronomique, période 2 sec.)

Il faut noter que les propriétés de recouvrement OverBeg, OverEnd, de continuité ContBeg, ContEnd ne concernent que les objets d'une *même* séquence. Ainsi, dans les exemples précédents, il y a des recouvrements partiels de “b” par des objets appartenant à d'autres séquences.

6. Algorithme d'instanciation d'objets sonores (*sound-object instantiation algorithm*)

(Voir chapitre IX §8)

```
Procédure Mise_en_temps

Variables globales:
   NMAX = nombre maximal de séquences
   IMAX = longueur maximale d'une séquence
   KMAX = nombre maximum d'objets dans la structure
   Seq[NMAX][IMAX],Δ[IMAX],T[IMAX],BTflag,
   Obj[KMAX],t1[KMAX],t2[KMAX],t"1[KMAX],t"2[KMAX],TrBeg[KMAX],TrEnd[KMAX],
      α[KMAX],δ[KMAX]

Entrée:
   Seq[][],Obj[],T[]

Sortie:
   T[],t1[],t2[],TrBeg[],TrEnd[],α[],δ[]

Début
   Pour (i = 1; i ≤ IMAX)
   |     Δ[i] <— 0;
   |     i <— i + 1;
   Finpour
   Calcul_de_alpha;

Reprise:
   Pour (nseq = 1; nseq ≤ NMAX)
   |     Placer(nseq); /* Calcul de t1[], t2[] */
   |     Si(Positionner(nseq,nature_temps) = échec)
   |     alors
   |     |     Afficher "Mise en temps impossible";
   |     |     /* Ici on peut aussi relancer l'algorithme en relâchant toutes les
   |     |        contraintes ContBeg, OverBeg ou OverEnd. */
   |     |     Stop;
   |     Finsi
   |     Si((nature_temps = lisse) et (nseq = 1))
   |     alors
   |     |     Interpolation_des_stries;
   |     Finsi
   |     BTflag <- faux;
   |     Pour(i = 1; i ≤ IMAX)
   |     |     T[i] <- T[i] + Δ[i];
   |     |     Si(Δ[i] ≠ 0)
   |     |     alors
   |     |     |     BTflag <- vrai;
   |     |     Finsi
   |     |     Δ[i] <- 0;
   |     |     i <- i + 1;
   |     Finpour
   |     Si(BTflag && (nseq > 1)) aller à Reprise;
   |     nseq <— nseq + 1;
   Finpour
Fin
```

Procédure Placer(nseq)

```
Début
   Pour (i = 1, i ≤ IMAX)
   |     k <— Seq[nseq][i]; j = Obj[k];
   |     Si(Obj[k] est un silence)
   |     alors
   |     |     … /* Calcul avec période locale */
   |     sinon
   |     |     t1[k] <— α[k].tmin[j] + T[i];
   |     |     t2[k] <— α[k].tmax[j] + T[i];
   |     Finsi
   |     i <— i + 1;
   Finpour
Fin
```

Fonction Positionner(nseq,nature_temps)

```
Variables locales:
   i,j,k,iprev,inext,ibreak,t'1[IMAX],t'2[IMAX],t"1[IMAX],t"2[IMAX],δ1[IMAX],δ2[IMAX],
   shift1,shift2,Ts[IMAX],dΔ0[IMAX],dΔ1[IMAX],dΔ2[IMAX],
   ContEndPrev[IMAX],OverEndPrev[IMAX],BrkTempoPrev[IMAX],
   ens_sol1[IMAX],ens_sol2;

Début
   Ts[0] <— t'2[0] <— t"2[0] <— -∞ ;
   dΔ0[0] <— dΔ1[0] <— dΔ2[0] <— δ1[0] <— δ2[0]<— 0;
   ContEndPrev[0] <— faux;
   OverEndPrev[0] <— BrkTempoPrev[0] <— vrai;
   i <— j <— k <— iprev <— ibreak <— 0;

Incrémentation:
   dΔ0[i+1] <— dΔ0[i] + dΔ1[i] + dΔ2[i];

   /* Recherche du rang du prochain objet non vide */
   Pour (inext = i+1; inext ≤ IMAX)
   |     dΔ0[inext] <— dΔ0[i+1];
   |     dΔ1[inext] <— dΔ2[inext] <— 0;
   |     Si (Seq[nseq][inext] ≠ 0) interrompre Pour;
   |     inext <— inext + 1;
   Finpour

   /* Mise à jour de Ts */
   Si (Ts[i] < t"2[i])
   alors
   |     Ts[inext] <— t"2[i];
   |     ContEndPrev[inext] <— ContEnd[j];
   |     OverEndPrev[inext] <— OverEnd[j];
   |     BrkTempoPrev[inext] <— BrkTempo[j];
   sinon
   |     Ts[inext] <— Ts[i];
   |     ContEndPrev[inext] <— ContEndPrev[i];
   |     OverEndPrev[inext] <— OverEndPrev[i];
   |     BrkTempoPrev[inext] <— BrkTempoPrev[i];
   Finsi
   iprev <— i;
   i <— inext;
   k <— Seq[nseq][i];
   δ1[i] <— δ2[i] <— 0;
```

```
   Si (k = -1)   /* Fin de séquence "NIL" */
   alors
   |     Si (Solution_acceptée)
   |     alors
   |     |     Pour (i = 1; i ≤ IMAX)
   |     |     |     Δ[i] <— Δ[i] + dΔ0[i]+ dΔ1[i]+ dΔ2[i];
   |     |     |     k <— Seq[nseq][i];
   |     |     |     Si (k > 0)
   |     |     |     alors
   |     |     |     |     t1[k] <— t"1[i];
   |     |     |     |     t2[k] <— t"2[i];
   |     |     |     |     δ[k] <— δ1[i] + δ2[i];
   |     |     |     Finsi
   |     |     |     i <— i + 1;
   |     |     Finpour
   |     |     Retour(succès);
   |     sinon
   |     |     aller à Backtracking;    /*  Non défini ici */
   |     Finsi
   Finsi
   j <— Obj[k];
   TrBeg[k] <— TrEnd[k] <— 0;
   t"1[i] <— t'1[i] <— t1[k] + Δ[i] + dΔ0[i];
   t"2[i] <— t'2[i] <— t2[k] + Δ[i] + dΔ0[i];

   shift1 <— Ts[i] - t"1[i];
   Si (Situation(i,nseq,shift1,nature_temps,t"1[i],t"2[i]) = acceptable)
   alors
   |     shift1 <— 0;
   Finsi
   ens_sol1[i] <— Choix_possibles(i,nseq,t'1[i],t'2[i],shift1,Ts[i],nature_temps,1);
   Si(shift1 = 0)
   alors
   |     aller à Incrémentation
   Finsi
   Si (ens_sol1[i] = vide)
   alors
   |     shift2 <— - shift1;
   |     aller à Décrémentation;
   Finsi
   Aller à Nouveau_choix1;

Nouveau_choix1:
   Si (ens_sol1[i] = vide)
   alors
   |     shift2 = -shift1;
   |     aller à Décrémentation;
   Finsi
   sol1 <— Prochain_choix(ens_sol1[i]);
   ens_sol1[i] <— ens_sol1[i] \ {sol1};
   t'1[i] <— t1[k] + Δ[i] + dΔ0[i] + shift1;
   t'2[i] <— t2[k] + Δ[i] + dΔ0[i];
   dΔ1[i] <— 0;
   δ1[i] <— 0;
   TrEnd[k] <— 0;
```

```
   Si (sol1 = Truncate beginning)
   alors
   |     TrBeg[k] <— shift1;
   sinon
   |     TrBeg[k] <— 0;
   |     t'2[i] <— t'2[i] + shift1;
   |     Si (sol1 = Shift object) δ1[i] <— shift1;
   |     Si (sol1 = Break tempo) dΔ1[i] <— shift1;
   Finsi
   t"1[i] <— t'1[i];
   t"2[i] <— t'2[i];
   aller à Incrémentation;

Nouveau_choix2:
   Si(ens_sol2 = vide)
   alors
   |     Si(ens_sol1[i] ≠ vide)
   |     alors
   |     |     aller à Nouveau_choix1;
   |     sinon
   |     |     Si((i > 1) et (Reloc[j] ou (shift2 > 0 et 0 < ibreak < i)))
   |     |     alors
   |     |     |     aller à Décrémentation;
   |     |     sinon
   |     |     |     Retour(échec);   /* Backtracking possible ici */
   |     |     Finsi
   |     Finsi
   Finsi
   sol2 <— Prochain_choix(ens_sol2);
   ens_sol2 <— ens_sol2 \ {sol2};
   t"1mem <— t"1[i]; t"2mem <— t"2[i];
   δ2mem <— δ2[i]; dΔ2mem <— dΔ2[i];
   t"2[i] <— t"2[i] + shift2;
   Si (sol2 = Truncate end)
   alors
   |     TrEnd[k] <—  TrEnd[k] - shift2;
   sinon
   |     t"1[i] <— t"1[i] + shift2;
   |     Si (sol2 = Shift object) δ2[i] <— δ2[i] + shift2;
   |     Si (sol2 = Break tempo) dΔ2[i] <— dΔ2[i] + shift2;
   Finsi

   shift3 <— Ts[i] - t"1[i];
   shift4 <— Autre_correction1(i,nseq,shift3,t"2[i]);
   Si (shift4 = 0)
   alors
   |     aller à Incrémentation;
   sinon
   |     Si(ens_sol2 ≠ vide)
   |     alors
   |     |     t"1[i] <— t"1mem; t"2[i] <— t"2mem; /* TrEnd[k] n'a pas varié */
   |     |     δ2[i] <— δ2mem; dΔ2[i] <— dΔ2mem;
   |     |     aller à Nouveau_choix2;
   |     sinon
   |     |     shift2 <— - shift4;
   |     |     aller à Décrémentation;
   |     Finsi
   Finsi
```

```
Décrémentation:
   Si(i = 1) Retour(échec);
   Tsm <— Ts[i];
   i <— iprev;
   Pour (iprev = i - 1; iprev ≥ 0)
   |     Si (Seq[nseq][iprev] > 0) interrompre Pour;
   |     iprev <— iprev - 1;
   Finpour
   k <— Seq[nseq][i]; /* k est strictement positif */
   j <— Obj[k];
   Si (Ts[i] = Tsm) aller à Décrémentation;
   ens_sol2 <— Choix_possibles(i,nseq,t"1[i],t"2[i],shift2,Tsm,nature_temps,2);
   aller à Nouveau_choix2;
Fin


Fonction    Choix_possibles(i,nseq,t1,t2,shift,Ts,nature_temps,numéro)

Début
   sol <— vide;
   k <— Seq[nseq][i];
   j <— Obj[k];
   Si ((numéro = 1) et (nature_temps = lisse) et (nseq = 1) et (shift > 0))
   alors
   |     Retour({Break tempo});
   Finsi
   Si (Reloc[j] ou j = 1)
   alors
   |     sol <— sol ∪ {Shift object};
   Finsi
   Si ((numéro = 1) et (i = 1) et (shift > 0) et (BrkTempoPrev[i]
                ou (nature_temps = lisse)))
   alors
   |     sol <— sol ∪ {Break tempo};
   Finsi
   Si ((numéro = 1) et (shift > 0) et TruncBeg[j] et (t2 > Ts))
   alors
   |     sol <— sol ∪ {Truncate beginning};
   Finsi
   Si ((numéro = 2) et (shift < 0) et TruncEnd[j] et (t1 < (Ts + shift)))
   alors
   |     sol <— sol ∪ {Truncate end};
   Finsi
   Retour(sol);
Fin
```

```
Fonction Autre_correction1(i,nseq,shift,t2)

Début
   k <— Seq[nseq][i];
   j <— Obj[k];
   t1 <— t"1[i];
   Si (shift = 0)  Retour(0);  /* Situation 2 */
   Si (shift < 0 et (non ContBeg[j]) et (non ContEndPrev[i]))
      Retour(0);    /* Situation 1 */
   Si (shift > 0)   /* Situation 3 ou 4 */
   alors
   |     Si(OverBeg[j] et OverEndPrev[i] et (((t1 + shift) ≤ t2) ou (non ContEnd[j])))
   |        Retour(0);
   |     Si (TruncBeg[j])
   |     alors
   |     |     Si ((t1 + shift) ≤ t2)
   |     |     alors
   |     |     |     TrBeg[k] <— TrBeg[k] + shift;
   |     |     |     t"1[i] <— t"1[i] + shift;
   |     |     |     Retour(0);
   |     |     Finsi
   |     Finsi
   Finsi
   Retour(shift);
Fin
```

```
Fonction Situation(i,nseq,shift,nature_temps,t1,t2)

Début
   Si (i = 1) Retour(acceptable);
   k <— Seq[nseq][i];
   j <— Obj[k];
   Si ((nature_temps = lisse) et (nseq = 1)) Retour(inacceptable);
   Si ((shift < 0) et (non ContBeg[j]) et (non ContEndPrev[i]))
      Retour(acceptable);                                                    /* 1 */
   Si (shift = 0)  Retour(acceptable);                                       /* 2 */
   Si (shift > 0)
   alors
   |     Si(Obj[k] est un silence) Retour(acceptable);
   |     Si ((Ts ≤ t2) et OverBeg[j] et OverEndPrev[i])  Retour(acceptable);   /* 3 */
   |     Si ((Ts > t2) et OverBeg[j] et OverEnd[j] et OverEndPrev[i]
   |        et (non ContEnd[j])) Retour(acceptable);
   |                                                                         /* 4 */
   Finsi
   Retour(inacceptable);
Fin
```

7. Structures polymétriques — approche algébrique (*polymetric structures — an algebraic approach*)

(Voir chapitre VIII §2)

La notion de séquence a été construite, plus ou moins de manière intuitive, sur une fonction injective θ assignant à chaque étiquette d'objet temporel/atemporel une date symbolique. On considère maintenant le cas où θ n'est pas injective, et on partitionne toute structure finie d'objets temporels $\{(x_i,\theta(x_i))\}$ "ponctuels" en un ensemble de séquences $\{(S_1,\theta_1),\ldots,(S_k,\theta_k)\}$ telles que θ_i est la restriction de θ à S_i, et que θ_i est injective du fait que (S_i,θ_i) est une séquence. Une structure ainsi partitionnée est une **structure polymétrique**. Ce concept est présenté ici en relation avec celui de **trace**, que l'on considère comme une actualisation des réseaux de Petri.[266]

[266] Mazurkiewicz 1984b.

7.1 Traces (*traces*)

On considère un **domaine d'événements** E qui comprend un ensemble E d'"événements", un ensemble X d'"objets" et une relation R entre objets et événements (une partie du produit cartésien X x E). La relation R(x,e) peut être énoncée: "e concerne x", ou encore: "x est sujet à e". A partir de R on définit une **relation de dépendance** D entre événements:

$$D(e_i,e_j) <=> \exists\ x \in X,\ R(x,e_i) \text{ et } R(x,e_j)\ .$$

A cette relation D réflexive, symétrique, mais pas nécessairement transitive (ressemblance), on oppose la relation irréflexive et symétrique d'**indépendance** I telle que I = E x E - D, soit encore

$$\forall e_i,e_j \in E,\ \ D(e_i,e_j) \text{ ou (exclusivement) } I(e_i,e_j)\ .$$

On note par ailleurs E* le monoïde des chaînes d'événements (E, ., λ), où "." représente l'opération de concaténation et λ l'élément neutre pour cette opération (événement vide). On considère enfin la congruence minimale sur E*, notée "≈", telle que:

$$\forall e_i,e_j \in E,\ \ I(e_i,e_j) => e_i\ e_j \approx e_j\ e_i\ .$$

Le quotient (E* / ≈) est appelé l'**algèbre de traces** sur E. ([267])

Di l'on appelle **processus** une trace t ∈ (E* / ≈), toute chaîne e ∈ E* appartenant à la classe *t* est une **observation** de ce processus. Chaque observation est un ensemble totalement ordonné d'événements. Si l'ordre des occurrences de deux événements est inversé dans deux observations du même processus, cela signifie que les occurrences correspondantes de ces événements sont indépendantes bien qu'elles apparaissent ordonnées en raison des différents points de vue.

7.2 Structures polymétriques (*polymetric structures*)

L'ensemble des objets X sert à définir la relation R qui à son tour est utilisée pour construire les relations D et I. On part d'"objets" insécables pour arriver à la notion de séquence: les événements qui concernent le même objet ne peuvent être simultanés; ils sont nécessairement séquentiels donc mutuellement dépendants. Inversement, les événements qui ne concernent aucun objet commun peuvent se produire indépendamment les uns des autres. Adaptée à la description des structures polymétriques, la formulation précédente devient:

Soient un ensemble d'événements (que nous appellons maintenant des **objets temporels**) $E = \{(x_i,\theta(x_i))\}$, dans lequel x_i et $\theta(x_i)$ sont respectivement l'étiquette et la date symbolique de chaque événement, un ensemble $X = \{S_1,\ldots,S_k\}$ d'objets[268], et une relation R telle que:

$$R(S_i,(x_j,\theta(x_j))) <=> (x_j,\theta(x_j)) \in S_i\ .$$

Appelons D la relation de dépendance:

$$D((x_j,\theta(x_j)),(x_{j'},\theta(x_{j'}))) <=> \exists\ S_i \in P\ \ |\ \ (x_j,\theta(x_j)) \in S_i \ \text{ et } \ (x_{j'},\theta(x_{j'})) \in S_i \ \text{ avec } j \neq j'.$$

La condition supplémentaire $j \neq j'$ implique que la relation de dépendance D n'est plus réflexive.

Supposons maintenant que tous les objets de X sont des **séquences**. Puisque la propriété caractéristique d'une séquence S_i est que la restriction θ_i de θ aux objets temporels de cette séquence est injective, la relation de dépendance s'écrit plus simplement:

$$D((x_j,\theta(x_j)),(x_{j'},\theta(x_{j'}))) <=> \theta(x_j) \neq \theta(x_{j'})\ .$$

Nous appelons **structure polymétrique** l'ensemble P de *toutes* les séquences que l'on peut construire sur E. Chaque séquence de P est un ensemble d'objets temporels $(x_i,\theta(x_i))$ *strictement*

---

[267] On peut formuler cette notion de manière équivalente en considérant les chaînes d'un alphabet E dans lequel I est une relation de "commutation partielle". Cette formulation a été appliquée à la musique par Chemillier (1987).

[268] Nous avons conservé ici la désignation "objet" pour faire le lien avec les notions introduites au §7.1.

ordonnés en raison de leur dépendance. Elle peut donc être représentée de manière unique par une chaîne $x_1 \ldots x_n$.

Nous appelons **structure polymétrique complète** une structure polymétrique $P = \{S_1,\ldots,S_k\}$ dans laquelle

(1) $\forall S_i \in P$, $\forall (x_j,\theta(x_j)) \in S_i$, $\exists S_{i'} \in P$ | $\exists (x_{j'},\theta(x_{j'})) \in S_{i'}$, $\theta(x_j) = \theta(x_{j'})$;
(2) toutes les séquences ont le même nombre *n* d'objets temporels ;
(3) $\forall S_i \in P$, $\exists (x_j,\theta(x_j)) \in S_i$, $\theta(x_j) = 0$;
(4) $\forall S_i \in P$, $\forall (x_j,\theta(x_j)) \in S_i$, $\theta(x_j) < n$.

Informellement, à tout objet temporel d'une séquence correspond moins un objet de même date symbolique (nécessairement d'une autre séquence). Etant donné que deux objets d'une même séquence ont des dates distinctes, il est facile de montrer que dans ce cas tous les objets du même rang dans les séquences ont la même date symbolique. Les conditions (3) et (4) impliquent que toutes les dates de l'intervalle [0,n-1] sont présentes dans chaque séquence.

Etant donné un ensemble d'objets temporels $\{(x_j,\theta(x_j))\}$, une structure polymétrique complète est construite à partir des classes d'équivalences:

$$(x_j,\theta(x_j)) = (x_{j'},\theta(x_{j'})) \iff \theta(x_j) = \theta(x_{j'}) \ .$$

Soit *n* le nombre de ces classes ainsi obtenues (durée symbolique de la structure). On complète ensuite les classes par des objets vides ("_",$\theta(x_j)$) de sorte que leurs cardinaux soient égaux. Soit *k* le cardinal commun des classes. On dit que la structure est **polyphonique de degré *k***. On construit enfin un ensemble de *k* chaînes $\{(x_1 \ldots x_n)\}$ telles que $(x_j,\theta(x_j))$ appartient à la classe *j*, ensemble qui doit contenir au moins une occurrence de chaque $(x_j,\theta(x_j))$.

7.3 Objets ponctuels

On a affaire ici à des objets temporels ponctuels (sans durée). On peut remplacer chaque objet temporel du type "intervalle" par deux événements ponctuels de dates distinctes qui représentent son "début" et sa "fin". Appliquons cette construction à l'exemple du chapitre VII §4:

1	2	3	4	5	6	7	8	9	10	11	12	13	14
_	_	a	_	b	c	a	_	_	NIL	_	_	_	_
_	_	_	e	_	-	_	_	f	_	g	NIL	_	_

Chaque objet temporel (x,θ(x),d(x)) où θ(x) est la date symbolique de l'origine de *x* et d(x) sa durée, est remplacé par deux objets ponctuels (x,θ(x)) et (x',θ(x)+d(x)), où *x'* est l'étiquette de la fin. Les paires $(x_j,\theta(x_j))$ sont donc

(a,3)	(c,6)	(e,4)	(f,9)
(b,5)	(b',6)	(-,6)	(g,11)
(a',5)	(a,7)	(c',7)	(a',11)
(a',10)	(e',6)	(-',9)	(f',11)

ce qui donne les classes

0 = ∅	2 = ∅	4 = {e}	6 = {c,-,b',e'}	8 = ∅	10 = ∅
1 = ∅	3 = {a}	5 = {b,a'}	7 = {a,c'}	9 = {f,-'}	11 = {g,a',f'}

que l'on complète ainsi:

0 = {_,_,_,_}	2 = {_,_,_,_}	4 = {e,_,_,_}	6 = {c,-,b',e'}	8 = {_,_,_,_}	10 = {_,_,_,_}
1 = {_,_,_,_}	3 = {a,_,_,_}	5 = {b,a',_,_}	7 = {a,c'}	9 = {f,-',_,_}	11 = {g,a,f',_}

A chaque entier *i* de 0 à 11 est associée ici la liste des objets ponctuels de rang i+1 dans une séquence. On a donc par exemple l'ensemble de séquences suivant:

{ _ _ _ a _ b c a _ _ _ g ,
_ _ _ _ e a' - c' _ f _ f',
_ _ _ _ _ _ b' _ _ _ _ _ ,
_ _ _ _ _ _ e' _ _ -'a' }

L'ordre dans les classes étant arbitraire, on aurait pu écrire un grand nombre d'autres formules équivalentes.

La représentation d'intervalles supports temporels par deux coordonnées est difficile à appréhender.[269] On peut se passer des objets d'étiquettes *x'* à condition que dans chaque séquence l'objet temporel non vide suivant immédiatement un objet temporel non vide *x* ait la même position que l'extrémité *x'* de *x*. Formellement,

$\forall (x_j, \theta(x_j)) \in S_i$ de durée d (avec d > 0),

$\forall (x_{j'}, \theta(x_{j'})) \in S_i$ tel que $\theta(x_{j'})-\theta(x_j) < d$, $x_{j'}$ = "_"

$\exists j'' \in N$ tel que $(x_{j''}, \theta(x_{j''})) \in S_i$, $\theta(x_{j''})-\theta(x_j) = d$, et $x_{j''} \neq$ "_"

La notion de séquence n'est donc pas basée uniquement sur la "non-coïncidence" des objets temporels (restriction de θ injective), mais aussi sur la contiguïté des intervalles qui représentent les durées symboliques des objets temporels.

[269] La fin de l'article de Chemillier et Timis (1988) suggère qu'ils ont rencontré le même type de difficulté.

Acquisition et Représentation de Connaissances en Musique

— Bernard Bel

Cette étude traite de la représentation informatique de connaissances en musique, abordée à partir de deux expériences en grandeur réelle. La première est une méthode d'acquisition de connaissances en ethnographie mettant en interaction un expert (le musicien), un analyste (le musicologue) et une machine dans une situation d'apprentissage. Les schémas d'improvisation des musiciens sont identifiés et exprimés à l'aide de règles de production dans un formalisme dérivé des grammaires génératives et des langages de formes. Un algorithme déterministe de test d'appartenance de chaînes arbitraires au langage défini par une grammaire (sensible au contexte) est présenté, ainsi qu'une technique d'inférence inductive de langages réguliers permettant l'acquisition automatique de connaissances lexicales et syntaxiques.

La seconde expérience s'insère dans l'élaboration d'un environnement de composition musicale assistée par ordinateur. Le problème est ici la représentation du temps dans une structure discrète d'"objets temporels", et plus généralement la synchronisation de processus parallèles. Une méthode est proposée pour la détermination d'une structure à partir de données incomplètes sur la synchronisation des objets. La notion d'"objet sonore" est ensuite explicitée formellement. Un algorithme efficace permet l'instanciation des objets sonores affectés à une structure en tenant compte des contraintes liées à leurs propriétés métriques et topologiques.

Mots clés: langages formels, test d'appartenance, inférence grammaticale, synchronisation, représentation du temps.

This study deals with computer representations of musical knowledge on the basis of two real-scale experiments. The first experiment focusses on knowledge acquisition in ethnography: an expert (the musician), an analyst (the musicologist) and a machine are interacting in a learning situation. Improvisation schemata through which musicians express themselves are identified and formalized with production rules in a formalism derived from generative grammars and pattern languages. A deterministic algorithm is introduced for assessing the membership of arbitrary strings to the langage defined by a given (context-sensitive) grammar. A technique for the inductive inference of regular languages is presented, enabling automatic knowledge acquisition of syntactic and lexical knowledge. The second experiment is part of the design of a computer environment for musical composition. Here the problem is time representation in a discrete structure of "time objects", more generally the synchronization of parallel processes. A method is outlined for the determination of a structure with incomplete data about the synchronization of its objects. The concept of "sound object" is then formally introduced. An efficient algorithm is proposed for the time-setting of objects in a structure, given the constraints arising from their metric and topological properties.

Keywords: *formal languages, membership test, grammatical inference, synchronization, time representation.*