Chapitre 6

Grammaires, automates et musique

6.1. Introduction

Ce chapitre est un panorama de l'utilisation des grammaires et des automates en informatique musicale, depuis les travaux des pionniers dans les années soixante jusqu'à aujourd'hui. Il comporte un rappel des notions élémentaires de théorie des langages, un bref historique des réalisations qui ont marqué les quatre décennies écoulées, et une présentation détaillée de deux exemples, concernant d'une part l'enrichissement harmonique des grilles de jazz, et d'autre part l'analyse des séquences musicales sérielles. Le code *Common Lisp* ou *Lex* correspondant est disponible sur le *web*, ainsi que des fichiers midi ou audio illustrant ce chapitre (www.info.unicaen.fr/~marc/publi/grammaires). Cette présentation est issue d'un enseignement donné à l'université de Caen pendant une dizaine d'années, dans le cadre de projets de licence et maîtrise d'informatique.

6.2. La hiérarchie de Chomsky

6.2.1. *Grammaires formelles*

On rappelle dans cette partie les principales notions sur les grammaires formelles qui seront utilisées dans la suite. Elles sont tirées des ouvrages classiques [AHO 89, EIL 74, GRO 69].

Chapitre rédigé par Marc CHEMILLIER.

Une grammaire formelle est une description d'un ensemble de séquences de symboles. Les programmes informatiques, par exemple, sont des séquences dont les symboles sont les caractères alphanumériques du clavier, enchaînés selon les règles d'une grammaire formelle. Dans les applications linguistiques des grammaires formelles, les symboles sont les mots du dictionnaire et les classes grammaticales.

Formellement, une *grammaire* est définie par la donnée de deux ensembles disjoints de symboles, les terminaux *T* que l'on note par des minuscules et les non-terminaux *N* notés par des majuscules, d'un ensemble fini de *productions* (ou *règles de réécriture*) de la forme $\alpha \rightarrow \beta$ où α et β sont des séquences de symboles de *N* ou *T*, et d'un symbole particulier *S* non-terminal appelé *axiome*. Si $\alpha \rightarrow \beta$ est une production de la grammaire, la séquence $\gamma\alpha\delta$ peut se dériver en $\gamma\beta\delta$. Cela signifie que le fragment α peut être remplacé par β à l'intérieur des séquences dans lesquelles il apparaît.

Le *langage* engendré par la grammaire est l'ensemble de toutes les séquences de symboles terminaux que l'on peut obtenir par dérivation à partir de l'axiome, en utilisant les productions.

La hiérarchie de Chomsky distingue quatre classes de grammaires formelles, selon la forme de leurs productions [CHO 56] :

– type 0 : aucune restriction,

– type 1 (contextuelles, ou *context-sensitive*) : productions de la forme $\alpha A\beta \rightarrow \alpha\gamma\beta$ où *A* est un non-terminal et γ une séquence avec au moins un symbole,

– type 2 (algébriques, ou *context-free*) : productions de la forme $A \rightarrow \alpha$ où *A* est un non-terminal,

– type 3 (régulières ou linéaires) : productions de la forme $A \rightarrow wB$ ou $A \rightarrow w$, où *A* et *B* sont des non-terminaux, et *w* une séquence de terminaux.

Les grammaires de type 1 sont dites « contextuelles », car les séquences α et β y jouent le rôle de *contextes* dans lesquels on peut remplacer le symbole non-terminal *A* par la séquence γ. Notons que la condition sur le nombre de symboles de γ interdit toute production dans laquelle le membre de droite serait plus court que le membre de gauche.

Les grammaires algébriques (type 2) sont caractérisées par le fait que le membre gauche des productions est réduit à un seul symbole (non-terminal). Les grammaires régulières (type 3) sont des cas particuliers de grammaires de type 2, avec une restriction imposée au membre de droite α, qui ne peut contenir qu'un seul symbole non-terminal *B* situé à la fin.

Si l'on désigne par L_0, L_1, L_2, L_3, les ensembles de langages engendrés respectivement par les grammaires de types 0, 1, 2, 3, on a la suite d'inclusions suivante :

$$L_0 \supset L_1 \supset L_2 \supset L_3$$

Ainsi, tout langage régulier (appartenant à L_3) est aussi un langage algébrique (appartenant à L_2), car les grammaires régulières sont des cas particuliers de grammaires algébriques. Lorsque l'on parle du « type » d'un langage, on désigne le plus petit ensemble qui contient ce langage, dans la suite d'inclusions ci-dessus. Ce type correspond à la grammaire *la plus simple* permettant de décrire ce langage. Les ensembles étant emboîtés les uns dans les autres, un même langage peut être décrit par plusieurs grammaires de types différents, qui ne correspondent pas toutes au type du langage lui-même. Par exemple, les *ensembles finis* de séquences sont des langages de type régulier, comme le langage {*ababb*} réduit à une séquence unique. Pourtant, ce langage peut être engendré par la grammaire $S \rightarrow AB$, $A \rightarrow ab$, $B \rightarrow abb$ qui est algébrique, sa première production contenant deux non-terminaux en partie droite. Mais la grammaire régulière $S \rightarrow abB$, $B \rightarrow abb$ engendre le même langage, qui peut donc être donné par deux grammaires de types différents. Les langages de type algébrique sont caractérisés par des phénomènes de récursion infinie, comme le langage {*ab, aabb, aaabbb,* etc.} engendré par la grammaire $S \rightarrow aSb$, $S \rightarrow ab$. Les langages de type régulier ne comportent pas ce phénomène, et c'est la raison pour laquelle la grammaire algébrique précédente peut être remplacée par une grammaire régulière.

6.2.2. *Equivalence entre grammaires et automates*

La théorie des langages montre qu'il existe une équivalence entre les grammaires et certaines classes d'automates. Les automates sont des modèles abstraits de machines qui lisent des séquences de symboles. L'ensemble des séquences lues par une telle machine est le langage « reconnu » par l'automate. On montre ainsi que les grammaires régulières engendrent les langages reconnus par des *automates finis,* et les grammaires algébriques, ceux reconnus par des *automates à pile*.

Un automate fini est défini formellement par un ensemble fini d'*états*, parmi lesquels on distingue un état *initial* et des états *terminaux*, et par un ensemble de *flèches* étiquetées par des symboles reliant entre eux certains états. Les séquences reconnues par un automate fini sont les successions d'étiquettes obtenues en partant de l'état initial et en suivant un chemin dans l'automate jusqu'à un état terminal.

Voici, figure 6.1, un exemple d'automate à deux états, reconnaissant les séquences qui contiennent au moins un *b*. L'état 1 marqué par une flèche entrante est initial, l'état 2 marqué par un double cercle est terminal.

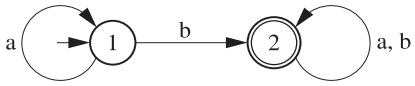

Figure 6.1. *Un automate fini*

Les langages reconnus par automate fini peuvent être représentés par ce que l'on appelle une *expression rationnelle* (théorème de Kleene). Une expression rationnelle est obtenue en appliquant à des ensembles finis de séquences de symboles, les opérations d'union ensembliste (ou de disjonction notée par une barre « | »), de concaténation et d'étoile. L'étoile consiste à itérer un ensemble de séquences, c'est-à-dire à concaténer bout-à-bout un nombre indéfini de séquences de cet ensemble (c'est la formalisation de la notion de « boucle » omniprésente dans les musiques électroniques actuelles). Voici une expression régulière pour l'automate de la figure 6.1 :

a b (a|b)**

Il est parfois utile, dans les applications musicales, de définir un automate avec des étiquettes sur les états plutôt que sur les flèches (voir figure 6.5). Une flèche d'un état à un autre indique que deux symboles peuvent se succéder. On se ramène sans difficulté à la représentation usuelle, en déplaçant les étiquettes des états vers les flèches qui y arrivent, et en ajoutant un état supplémentaire avec des flèches vers tous les états qui ne sont l'arrivée d'aucune flèche.

6.2.3. *Tables de transition et chaînes de Markov*

Pour décrire un automate fini, on utilise parfois une *matrice* ou *table de transition*, dont les lignes et les colonnes sont numérotées par les états de l'automate, et dans laquelle on indique aux croisements les symboles qui permettent de passer d'un état à un autre. Le tableau 6.1 est la table de transition de l'automate en figure 6.1.

	1	2
1	*a*	*b*
2	0	*a, b*

Tableau 6.1. *Table de transition de l'automate*

Lorsque la table de transition est munie de probabilités affectées aux différentes transitions partant de chaque état, on dit que l'automate est une *chaîne de Markov*.

Les tables de transition sont un moyen simple de réaliser informatiquement un automate fini. La fonction Common Lisp ci-après réalise l'automate précédent, par une simple récursion sur la séquence analysée représentée par une liste :

```
(defun automate (etat seq)
  (if (null seq)
      (if (member etat *etats-terminaux*)
          'Reconnu 'Non-reconnu)
      (automate (transition etat (car seq))
                (cdr seq))))
```

La fonction de transition détermine le nouvel état obtenu après lecture d'une lettre, par consultation de la table :

```
(defun transition (etat lettre)
  (cadr (assoc (list etat lettre) *table-transition*
               :test #'equal)))

(setq *table-transition*
'(((1 a) 1)
  ((1 b) 2)
  ((2 a) 2)
  ((2 b) 2)))

(setq *etats-terminaux* '(2))
```

Voici deux exemples d'exécution en partant de l'état initial 1. La première séquence n'est pas reconnue, car elle ne contient pas de *b*. La deuxième s'arrête dans l'état 2 qui est terminal, donc elle est reconnue par l'automate :

```
 ? (automate 1 '(a a a))
Non-reconnu

 ? (automate 1 '(a b a a a))
Reconnu
```

Une variante de la fonction précédente consiste à remplacer la récursion simple associée à une table, par des récursions croisées entre plusieurs fonctions correspondant à chaque état.

La représentation par table pose des problèmes de stockage quand la table est trop grande. Dans ce cas, il faut modifier la fonction de transition ci-dessus, en ne calculant les transitions qu'au fur et à mesure de la lecture, de façon « dynamique ».

6.2.4. *Transductions rationnelles*

Une *transduction rationnelle* est un automate fini « avec sorties », c'est-à-dire dont les flèches sont étiquetées par des couples. Lorsque l'on parcourt un chemin dans une transduction, la succession des éléments en première position dans chaque couple donne la séquence d'entrée, et celle des éléments en deuxième position donne la séquence de sortie correspondante. La figure 6.2 donne un exemple de transduction à trois états. L'étiquette 1 indique qu'il n'y a pas de symbole en sortie.

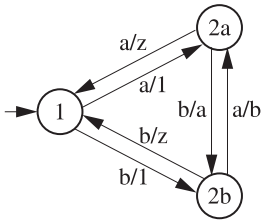

Figure 6.2. *Une transduction rationnelle*

Cette transduction transforme la séquence d'entrée en remplaçant les couples de lettres consécutives identiques par la lettre *z*. Le mécanisme de lecture et d'écriture est le suivant :

– si l'on lit deux lettres consécutives identiques, on écrit la lettre *z* en sortie, et on poursuit la lecture deux lettres plus loin ;

– sinon, on recopie la première lettre en sortie, et on poursuit la lecture en se décalant d'une lettre dans la séquence d'entrée.

Pour la séquence d'entrée *aabaaab,* la séquence de sortie est *zbzab*. Les états 2*a* et 2*b* se « souviennent » de la dernière lettre lue, respectivement *a* ou *b*. Tous les états sont terminaux, avec la convention supplémentaire que si l'on s'arrête dans les états 2*a* ou 2*b,* on rajoute la lettre correspondante à la fin de la séquence de sortie.

Les transductions rationnelles modélisent un type simple de transformation de séquences, dans lequel les symboles de la séquence d'arrivée dépendent de k symboles consécutifs dans la séquence de départ (elles généralisent la notion de *mapping* associant un à un les symboles d'entrée et de sortie, ce qui correspond à $k = 1$). On montre en théorie des langages qu'une transduction rationnelle conserve

le type des langages, c'est-à-dire qu'elle transforme tout langage régulier ou algébrique en un langage de même type. Aussi est-il parfois utile de remplacer l'automate d'un langage par celui d'un autre langage choisi pour être plus facilement implémentable, en combinant celui-ci à une transduction rationnelle qui passe de l'un à l'autre.

Dans les méthodes utilisées pour la génération de séquences musicales, il est fréquent que le problème soit ainsi « factorisé » en plusieurs transductions rationnelles, qui effectuent des traitements successifs. Le plus souvent, la génération d'un langage musical est obtenue par une grammaire ou un automate produisant un « squelette » de ce langage (par exemple des profils mélodiques ou des successions de degrés harmoniques), puis une ou plusieurs transductions permettent d'« habiller » les séquences obtenues pour fabriquer de véritables séquences musicales. Cette construction est schématisée par la figure 6.3.

Figure 6.3. *Factorisation d'un générateur musical en traitements successifs*

On verra que les exemples de Barbaud, de Cope, et de la génération de grilles de blues présentés ci-après illustrent cette construction fondamentale.

6.2.5. *Notion de* parsing

Les grammaires et les automates peuvent être utilisés aussi bien dans le sens de la génération que de l'analyse. Dans ce dernier cas, un « analyseur de séquences » (ou *parser* en anglais) a pour tâche de déterminer si une séquence peut être obtenue par dérivation à partir de l'axiome et des productions d'une grammaire. Les automates finis ou à pile sont des modèles d'analyseurs efficaces pour les grammaires de types 3 et 2. En revanche, pour les grammaires plus complexes (types 1 ou 0), il n'existe pas de méthode efficace pour construire un tel analyseur.

La possibilité de réaliser des analyseurs efficaces explique que les grammaires de types 2 et 3 soient utilisées dans la compilation des programmes informatiques. Un compilateur fonctionne en deux phases. Tout d'abord, le programme à compiler est analysé par un automate fini (plus précisément une transduction rationnelle), qui reconnaît les unités lexicales du langage de programmation (mots-clés, opérateurs, identificateurs, constantes, etc.) et élimine les commentaires, sauts de lignes, etc. La séquence d'unités ainsi construite est ensuite traitée par une grammaire algébrique, qui effectue l'analyse syntaxique, c'est-à-dire le contrôle des structures imbriquées

(début-fin, si-alors, parenthésages, etc.). Les commandes Lex et Yacc du système UNIX permettent de construire des analyseurs pour chacune de ces deux phases. Lex construit un automate fini à partir d'une description sous forme d'expression régulière. Yacc construit un automate à pile à partir d'une grammaire algébrique BNF (Backus-Nauer-Form).

Le listing ci-après est une description de l'automate ci-dessus dans le format utilisé par Lex, qui comporte trois parties séparées par des % % : les déclarations de variables auxiliaires sous forme d'expressions régulières, puis les expressions régulières reconnues par l'automate associées à des actions à exécuter sous forme de code C, et enfin les procédures complémentaires en C. La première partie, facultative, est absente dans l'exemple ci-dessous :

```
 % %
a*b(a | b)*    { printf(« Reconnu\n ») ; return 1 ; }
\n                    { printf(« Non reconnu\n ») ; return
1 ; }
.               ;

 % %
int yywrap()   { return 1 ;}
main()         { printf(« >« ) ; yylex() ; }
```

Ce fichier est donné en entrée à la commande Lex, qui produit en sortie un programme C, donnant lui-même après compilation, une commande exécutable qui réalise l'automate décrit dans le fichier. L'exécution de la commande flex -t toto.l > toto.c ; cc toto.c -o toto fabrique une fonction toto réalisant l'automate dont la description est contenue dans le fichier toto.l. Les deux exemples suivants reprennent ceux de la fonction Common Lisp discutés plus haut :

```
$ toto
> aaa
Non reconnu

$ toto
> abaaa
Reconnu
```

6.3. Grammaires formelles et musique

6.3.1. *L'article de Curtis Roads*

Dans cette partie, on retrace quelques étapes historiques de l'utilisation musicale des grammaires et des automates, en s'arrêtant sur deux automates particuliers

(Barbaud, Cope). Il s'agit d'un choix personnel de quelques exemples significatifs parmi beaucoup d'autres possibles. Les paragraphes suivants indiquent également des ouvrages généraux dans lesquels le lecteur trouvera des références complémentaires.

Les grammaires musicales ont fait l'objet d'un article de synthèse à la fin des années soixante-dix, dans lequel Curtis Roads dressait un bilan des recherches menées durant les deux décennies précédentes [ROA 79]. Il rappelait les principales notions concernant les grammaires formelles, et envisageait différentes manières de les appliquer à la musique en comparant leurs avantages respectifs. Il insistait sur le fait que pour construire une grammaire musicale, il n'y a pas une manière unique de choisir les *symboles*, ceux-ci pouvant aussi bien être des notes que des objets plus complexes comme des chiffrages d'accords, des sections d'une forme musicale, etc., selon l'utilisation que l'on veut faire de cette grammaire. Les principales références présentées dans l'article sont celles de Ruwet [RUW 71], Nattiez [NAT 75], Laske [LAS 73], Smoliar, Moorer [MOO 72], Winograd [WIN 68], Lerdahl et Jackendoff, et Roads. Ce panorama concluait une période marquée par le rayonnement de la linguistique, particulièrement sensible dans les travaux d'analyse paradigmatique développés en France par Nicolas Ruwet.

Parmi les recherches citées par Roads, celles de Lerdahl et Jackendoff ont donné naissance quelques années plus tard au livre *A Generative Theory of Tonal Music* [LER 83] qui a marqué les études de psychologie de la musique. Mais ce travail ne contribue pas directement aux recherches sur les applications musicales des grammaires formelles, bien qu'il s'inspire de la linguistique, car son propos est plutôt de modéliser de façon non-complètement formelle la perception des structures métriques et tonales de la musique.

Au cours des deux décennies suivantes (années 1980 et 1990), divers ouvrages récapitulatifs ont vu le jour [BAR 84, CROSS 91, SCH 93], regroupant des sélections d'articles qui traitent de ce sujet. Ces recherches concernent un large éventail de musiques, incluant le jazz, comme on le verra dans la partie suivante, et les musiques non occidentales [BEC 79, BEL 92, HUG 91, EST], et montrent la persistance de l'intérêt porté aux grammaires formelles, dans un contexte où d'autres modèles concurrents ont vu le jour, notamment la programmation logique avec contraintes.

Parallèlement aux travaux se référant explicitement aux grammaires, une autre tradition de recherches a développé le point de vue des automates, formellement équivalent comme il a été rappelé plus haut. Xenakis a été un pionnier dans l'utilisation musicale des automates avec probabilités de transition, c'est-à-dire des chaînes de Markov [XEN 63], dont il s'est servi pour calculer des successions de nuages de sons dans *Analogiques A* (1958), ou de glissandi enchevêtrés dans *Syrmos* (1958). Barbaud a également utilisé les chaînes de Markov dès les années soixante,

pour construire des générateurs de musique tonale [BAR 65, BAR 68]. Les automates finis, quant à eux, interviennent dans un joli travail de Greussay sur la modélisation d'une pièce des *Mikrokosmos* de Bartok [GRE 73, RIO 79].

Aujourd'hui, les travaux de Cope sur la simulation stylistique adoptent des modèles apparemment plus complexes, comme les *réseaux de transition augmentés* [COP 91], qui se ramènent dans certains cas à des automates finis au sens usuel, comme on en verra un exemple au paragraphe 6.3.2.2.

6.3.2. *Deux automates musicaux*

6.3.2.1. *Barbaud*

Cette section présente de façon plus détaillée la construction de deux automates musicaux. Le premier fait partie d'un ensemble d'automates de musique tonale décrits par le compositeur Pierre Barbaud (mort en 1990). Comme tous les automates de Barbaud [BAR 65, BAR 68, BAR 75, BAR 93], celui-ci est « factorisé » en plusieurs traitements successifs, selon le principe décrit figure 6.3. Tout d'abord, une chaîne de Markov produit une séquence d'accords (une sorte de « basse chiffrée ») à partir d'une table d'enchaînements munie de probabilités, dans laquelle on effectue des tirages aléatoires successifs. Cette séquence est ensuite traitée en complétant les accords et en ajoutant des motifs mélodiques prédéfinis, par tirages dans des tables de positions d'accords et d'ornements.

La figure 6.4 est le début d'un contrepoint original de Barbaud à trois voix. Il fait partie d'un ensemble de fichiers calculés par lui, contenant des séquences musicales codées au format de la machine de synthèse *Biniou* [BRO 98]. Le programme qui a produit ce contrepoint à trois voix est partiellement décrit dans le chapitre 5 du *Vademecum de l'ingénieur en musique* [BAR 93]. La chaîne de Markov engendre des successions d'accords parfaits en rondes, tels que ceux des mesures un, trois ou quatre dans l'exemple ci-dessus. La table de transition comporte 160 couples définissant les enchaînements d'accords possibles, et il n'y a pas d'états terminaux (l'exemple de la figure 6.4 s'interrompt brusquement sans cadence conclusive à la quatre-vingtième mesure).

L'ajout d'ornements fonctionne en traitant la séquence harmonique par blocs de deux accords consécutifs. Elle tire des motifs mélodiques dans la table pour monnayer les trois notes de ces deux accords (ajout de retards, broderies, notes de passage), en fonction de leurs chiffrages et de l'intervalle entre leurs notes de basse. Il se peut, selon le tirage effectué, que l'un des deux accords ne soit pas modifié par le traitement, auquel cas il reste en rondes. Pour économiser l'espace utilisé par la table, les motifs sont stockés à une transposition près, c'est-à-dire qu'au cours de la génération, ils sont transposés de manière à s'ajuster aux accords de la séquence.

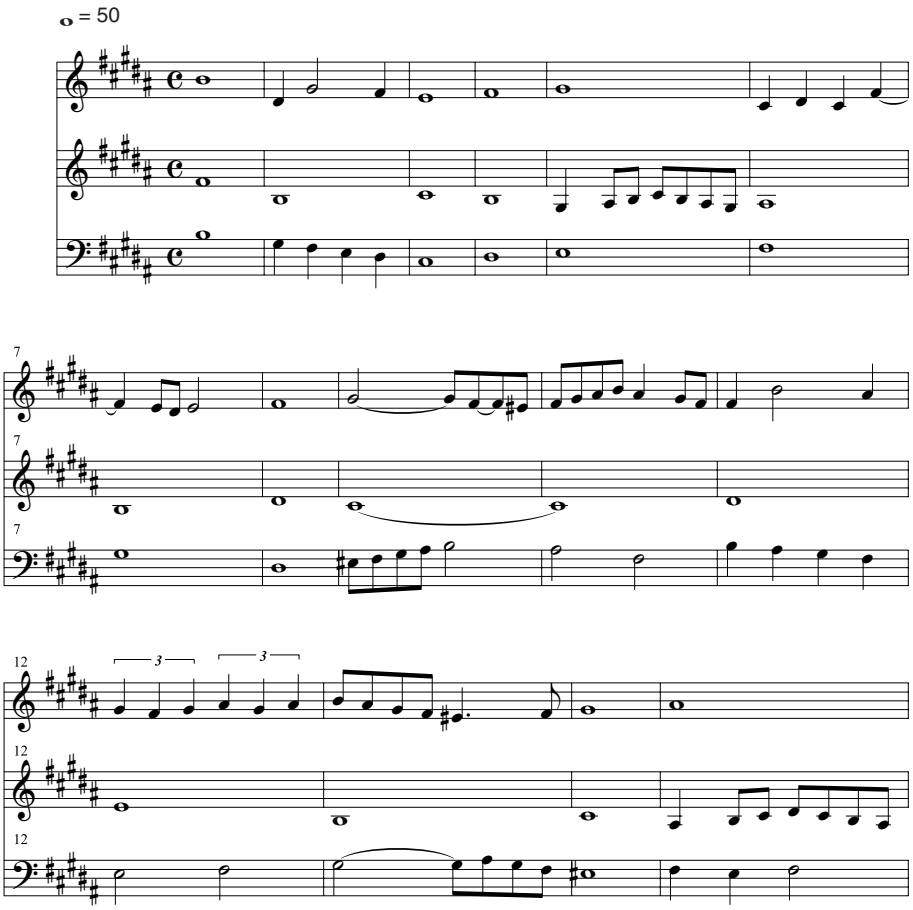

Figure 6.4. *Extrait (mesures 1 à 15) du contrepoint LCONT1.B3I de Pierre Barbaud*

Comme dans la plupart des programmes de Barbaud, le seul paramètre fixé par l'utilisateur est le nombre de mesures de la séquence (quatre-vingt dans l'exemple de la figure 6.4). Les tables utilisées par ce programme ne sont malheureusement pas publiées dans le livre de Barbaud, qui insiste sur l'idée que les tables données par lui ne servent qu'à illustrer une *méthode,* et que le lecteur est invité à imaginer ses propres tables d'enchaînements d'accords et de motifs mélodiques.

6.3.2.2. *Cope*

L'automate de Cope présenté ici fait partie d'un générateur d'inventions à deux voix, dont le code complet en Common Lisp est publié dans le chapitre 4 de

[COP 91]. Le programme recombine des motifs tirés d'un dictionnaire de figures mélodiques empruntées aux inventions de Bach, que Cope appelle des « signatures ». Comme le programme de Barbaud, le générateur est « factorisé » en plusieurs composantes (selon le principe décrit figure 6.3) : un automate fini utilisé dans le sens de la génération produit des séquences qui sont habillées mélodiquement au cours d'un traitement ultérieur, puis transformés par un module de mise en forme contrapunctique. Le programme comporte également un module de *pattern-matching* indépendant, qui détermine les motifs communs à plusieurs inventions de Bach.

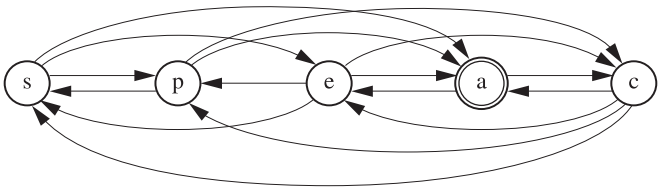

Figure 6.5. *Automate SPEAC de David Cope*

Dans le programme de Cope, le rôle de l'automate fini est limité à la génération de « profils ». Il produit des suites de lettres désignant des intervalles exprimés en demi-tons : $s = 0$, $p = 2$ ou -2, $e = 1$ ou -1, $a = 4$ ou -4, $c = 3$ ou -3. Cet automate est représenté figure 6.5. Tous les états sont initiaux. L'état terminal est *a,* avec la convention que l'on ajoute la lettre *c* après le dernier *a* produit. Le fonctionnement de l'automate est un parcours aléatoire jusqu'à ce que le nombre de *a* obtenus atteigne une certaine valeur. Il peut arriver qu'il tourne indéfiniment si les tirages ne donnent pas suffisamment de *a*.

Le traitement effectué ultérieurement « habille » le profil obtenu par un simple remplacement des lettres une par une. Pour chaque lettre, on cherche dans le dictionnaire un motif mélodique dont la distance entre les notes extrêmes soit égale à l'intervalle correspondant. S'il n'y en a pas, on met à la place un motif tiré au hasard. Le résultat produit à l'issue de cette phase est une simple ligne mélodique.

Le module de traitement contrapuntique prend cette ligne en entrée et la place dans la voix supérieure de l'invention, ainsi que dans la voix inférieure après transposition à l'octave inférieur et décalage d'une mesure. Dans la voix supérieure, les mesures paires sont remplacées par des noires à un intervalle de dixième au-dessus des notes de la voix inférieure. Dans la voix inférieure, les mesures impaires (exceptée la première) sont remplacées par des noires une sixte au-dessous des notes de la voix supérieure. Les deux voix sont tronquées par une cadence conclusive, insérée après un nombre de temps fixé par l'utilisateur.

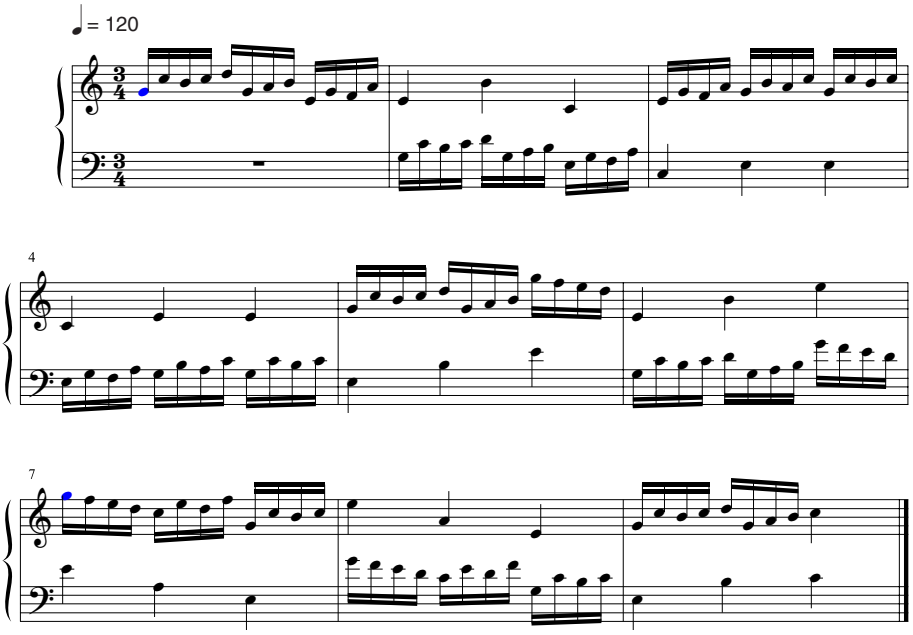

Figure 6.6. *Un exemple d'invention à deux voix (Cope)*

La figure 6.6 montre une séquence produite par le programme de Cope. Ici, le dictionnaire contient des motifs de huit notes consécutives tirés de l'*Invention n° 1* de Bach. Le nombre de *a* fixé pour le profil est dix, et le nombre de temps après lesquels l'invention est tronquée est vingt. Ces trois paramètres (nombre de notes des motifs, nombre de *a* du profil, nombre de temps de l'invention) sont déterminés par l'utilisateur du programme.

L'automate joue un rôle secondaire dans la construction de l'invention, l'essentiel du travail étant réalisé par le module de traitement contrapuntique. Voici le profil engendré par l'automate pour l'exemple de la figure 6.6, avec les intervalles des motifs associés aux lettres. C'est une alternance de *a* et *c*, mais les lettres entre parenthèses correspondent à des motifs qui disparaissent entièrement de la voix supérieure lors du traitement des mesures paires. Les crochets indiquent la portion de la séquence tronquée par la cadence conclusive. La lettre *a* est toujours associée au même motif (dont l'intervalle est 4). En revanche, la lettre *c* est associée à plusieurs motifs (intervalles 8, 5, –2) qui ne correspondent jamais à l'intervalles requis pour cette lettre (3 ou –3). Aussi, le « profil » produit par l'automate ne conditionne-t-il que partiellement le contour mélodique final :

a c (*a*) *c a* (*c*) *a c* (*a*) *c a* (*c*) *a* [*c a c a c a c*]

4 8 (4) 8 4 (5) 4 – 2 (4) – 2 4 (8) 4 [8 4 – 2 4 8 4 8]

Cope précise que le programme présenté ici est incomplet, car il manque des procédures compositionnelles (marches, plan tonal) nécessaires pour construire des inventions plus élaborées comme les deux exemples publiés dans son livre [COP 91, p. 142 et 148].

6.4. Grammaire de substitutions de jazz

6.4.1. *Règles de Steedman*

Cette partie est consacrée à l'étude d'une grammaire de substitutions de jazz, et à la réalisation d'un générateur et d'un analyseur de grilles, dont le code complet est disponible à l'adresse *web* indiquée en introduction. Les substitutions utilisées par les musiciens de jazz consistent à enrichir les grilles harmoniques sur lesquelles ils improvisent, en remplaçant certains accords par d'autres. Elles se prêtent bien à une description au moyen de règles de réécriture, qui permettent de les implémenter sous forme de grammaires formelles. La grammaire étudiée ici est due à Steedman, qui a proposé en 1984, une liste de six règles engendrant des variantes harmoniques de la grille standard du blues de douze mesures [STE 84]. Son article a eu un impact important sur d'autres travaux ultérieurs de Johnson-Laird et Pachet [JOH 91, PAC 98a, PAC 98b], et il a été complété par Steedman lui-même [STE 96]. La description du générateur de substitutions présentée ici est publiée dans [CHE 03].

La grammaire de Steedman est obtenue à partir d'un corpus de neuf grilles de blues middle-jazz et bop empruntées à un ouvrage de Coker sur l'improvisation jazz [COK 64]. Steedman montre que l'on peut dériver toutes les grilles répertoriées par Coker à partir d'une unique grille de blues rudimentaire de trois accords, en utilisant un ensemble de six règles de réécriture. Ces six règles ne jouent pas des rôles de même importance. Deux d'entre elles sont spécifiques : l'avant-dernière ne concerne qu'une grille du corpus et apparaît d'une certaine manière *ad hoc*, et la dernière introduit des accords de septièmes diminuées, ce qui la rend inopérante pour l'analyse de grilles harmoniques qui n'en comportent pas. Dans ce qui suit, nous nous restreindrons aux quatre premières règles publiées par Steedman, et c'est cet ensemble de règles que nous appellerons ici « la grammaire de Steedman ».

Les accords sont notés par des chiffres romains, comme des degrés dans une tonalité indéfinie, auxquels on ajoute éventuellement une abréviation indiquant le type de l'accord choisi parmi trois possibles : majeur (type par défaut), mineur septième (noté m7), et septième de dominante (noté 7). Pour donner une forme plus

concise à la grammaire, Steedman utilise une variable *x* désignant l'un quelconque des degrés, et des fonctions *Dx, Stbx, Sdx* désignant respectivement la dominante, la sus-tonique bémol, et la sous-dominante de *x*, dont le calcul s'effectue facilement sur le cercle présenté en figure 6.7 : *Dx* (respectivement *Stbx, Sdx*) est obtenu à partir de *x* par une rotation de sept unités dans le sens horaire (respectivement une unité et cinq unités), par exemple *Dx* = VII pour *x* = III.

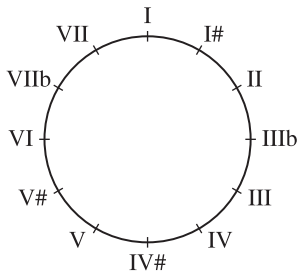

Figure 6.7. *Degrés des accords d'une grille de jazz*

Dans cette grammaire, tous les accords sont considérés comme des symboles non-terminaux. Chaque règle donne naissance à douze règles différentes, lorsque la variable *x* prend comme valeur les douze degrés possibles (et lorsque l'on adapte en conséquence les degrés *Dx, Sdx, Stbx*). Une variable supplémentaire *w* intervient dans la règle 3, pour désigner un accord indéterminé, dont le degré n'a pas été modifié par une règle appliquée précédemment, et dont le chiffrage n'est pas 7. Les règles 1 et 2 ont pour effet de diviser l'unité de longueur en unités plus petites (mesures, demi-mesures). La règle 2 introduit de plus un accord nouveau (sous-dominante), nécessaire pour expliquer le IV apparaissant en deuxième mesure du blues, comme on le voit dans le tableau 6.2, mais il faut noter que son usage réitéré a tendance à modifier profondément le rythme harmonique de la grille. Les règles 3 et 4 sont très utilisées par les musiciens de style bop, et correspondent respectivement à la « préparation II-V » et à la « substitution au triton ».

Règle 1 : $x \rightarrow x\ x$
$x7 \rightarrow x\ x7$
$xm7 \rightarrow x\ xm7$

Règle 2 : $x \rightarrow x\ Sdx$
$x7 \rightarrow x7\ Sdx$
$xm7 \rightarrow xm7\ Sdx$

Règle 3a : $w\ x7 \rightarrow Dx7\ x7$
$w\ x7 \rightarrow Dxm7\ x7$

Règle 3b : $w\ xm7 \rightarrow Dx7\ xm7$

Règle 4 : $Dx7\ x7 \rightarrow Stbx7\ x7$
$Dx7\ x \rightarrow Stbx\ x$
$Dx7\ xm7 \rightarrow Stbxm7\ xm7$

L'axiome *S* de la grammaire est associé à une règle supplémentaire, qui donne le schéma de base du blues (en six unités de deux mesures) :

$S \rightarrow$ I I7 IV I V7 I

La grammaire sous-entend un ensemble de règles implicites par lesquelles chaque accord non-terminal se réécrit comme une copie terminale de lui-même. Les symboles terminaux sont donc des copies des non-terminaux (rappelons que les ensembles de terminaux et de non-terminaux sont supposés disjoints), c'est-à-dire finalement les accords eux-mêmes. Notons que dans le blues, les accords majeurs sont souvent joués de façon « bluesy », c'est-à-dire qu'ils sont identifiés à des accords de septième de dominante. L'application des règles de Steedman, en revanche, distingue ces deux types d'accords, majeur non chiffrés d'un côté, et septième de dominante chiffrés 7 de l'autre. C'est pourquoi Steedman introduit dans son article un chiffrage 7', absent des règles reproduites ici, pour indiquer qu'un accord traité comme majeur dans l'application des règles peut être joué comme septième de dominante en fin de processus. Cet accord chiffré 7' est une sorte de symbole terminal résultant de la réécriture « bluesy » d'un accord majeur non terminal.

Les règles de Steedman sous-entendent également une hypothèse essentielle sur le plan des durées : chaque substitution s'effectue *à durée constante*. Cela signifie que dans les règles ayant un symbole de plus à droite qu'à gauche (règles 1 et 2), les deux accords de droite sont moitié plus courts que celui de gauche. Ces deux règles permettent ainsi de diviser les unités produites par l'axiome (deux mesures) en mesures simples d'abord, puis en demi-mesures, etc. Il est rare que la grille soit divisée au-delà de la demi-mesure, donc l'application potentiellement infinie du processus de division récursif défini par les règles 1 et 2 ne fonctionne concrètement que sur deux ou trois niveaux. Les autres règles (règles 3 et 4), qui ont autant de symboles des deux côtés de la flèche, n'introduisent pas de division des durées.

On notera les grilles sous forme de séquences d'accords, en séparant les mesures par des barres obliques. Ainsi :

I/IV/I/I7/IV/IV/I/VI7/II7/V7/I/I

correspond à la grille représentée dans le tableau 6.2.

I	IV	I	I7
IV	IV	I	VI7
II7	V7	I	I

Tableau 6.2. *Une grille de blues*

L'accord VI7 de la huitième mesure peut être obtenu en appliquant la règle $w\ x7 \rightarrow Dx7\ x7$ avec x = II7 à la grille suivante :

I/IV/I/I7/IV/IV/I/I/II7/V7/I/I

Voici la dérivation complète proposée par Steedman pour obtenir la grille du tableau 6.2 à partir de l'axiome :

I I7 IV I V7 I (production de l'axiome)

– Règles 1 et 2 (chaque unité se divise en deux mesures, avec ajout d'un IV en deuxième mesure par la règle 2) :

I/IV/I/I7/IV/IV/I/I/V/V7/I/I

– Règle 3a (appliquée à V7) :

I/IV/I/I7/IV/IV/I/I/II7/V7/I/I

– Règle 3a (appliquée à II7) :

I/IV/I/I7/IV/IV/I/VI7/II7/V7/I/I

6.4.2. *Moteur d'application des règles*

Dans cette section et la suivante, nous décrivons un petit programme en Common Lisp qui implémente un sous-ensemble de la grammaire de Steedman dans

le sens de la génération [CHE 03]. Le programme comporte un moteur d'application des règles, qui enrichit progressivement une grille de départ (donnée par l'axiome) par application successive des règles, et un « arrangeur » de grille, présenté au paragraphe suivant, qui réalise pour piano solo la séquence harmonique obtenue en fin de processus, avec quelques fragments de style boogie-woogie.

Le principe du moteur d'application des règles est de tirer au hasard une règle, de chercher une position dans la grille pour l'appliquer, puis d'effectuer la substitution. Si la règle tirée n'est pas applicable, on en tire une autre. On recommence ainsi l'opération un nombre de fois fixé à l'avance, ou bien on s'arrête si aucune règle n'est applicable.

La grammaire a été modifiée pour contrôler le découpage de la grille en mesures, en introduisant des « / » dans l'énoncé des règles. Les règles 1 et 2 sont modifiées de la manière suivante :

$$x \rightarrow x\ x$$

est remplacé par :

$$/x/ \rightarrow /x\ x/$$

Ainsi, la division minimale de la grille est fixée à la demi-mesure, car les nouvelles règles 1 et 2 ne sont applicables qu'à des mesures non divisées. Les règles 3 et 4 sont dédoublées. La forme originelle :

$$w\ x7 \rightarrow Dx7\ x7$$

est complétée par :

$$w/x7 \rightarrow Dx7/x7$$

La première s'applique à l'intérieur d'une même mesure, alors que la seconde s'applique de part et d'autre d'une barre de mesure.

Pour introduire les barres de mesure dans le processus de dérivation, on remplace l'axiome par :

$$S \rightarrow \text{I/I/I/I7/IV/IV/I/I/V/V7/I/I}$$

en appliquant systématiquement la règle 1 à tous les symboles de l'axiome originel. En se limitant ainsi aux dérivations de ce nouvel axiome, on exclut certaines séquences, ce qui revient à calculer un sous-ensemble du langage défini par la grammaire de Steedman.

Dans le programme en Common Lisp, les règles sont représentées par des couples formés des parties gauche et droite, dans un codage proche de leur énoncé d'origine, et sont stockées dans une liste. Le moteur comporte une fonction qui détecte si le membre gauche d'une règle apparaît dans une grille, et une fonction qui effectue le remplacement par le membre de droite en cas de succès. La variable apparaissant dans l'énoncé des règles est également traitée par le programme, qui lui affecte une valeur pendant la phase de détection.

La fonction de substitution prend en argument le nombre d'applications de règles souhaité. Cette valeur ne prend pas en compte la règle 1, qui n'enrichit pas à proprement parler la grille. Il peut arriver qu'aucune règle ne soit applicable, auquel cas le processus s'arrête avant que le nombre souhaité initialement ne soit atteint. Quand le degré d'un accord est modifié, on le marque par une * pour indiquer que l'accord ne doit plus être affecté à la variable *w*. Voici deux exemples d'exécution de la fonction, avec respectivement une et dix applications de règles (autres que la règle 1) :

```
 ? (substitue 1)
((i)/(i)/(i)/(i 7)/(iv)/(iv)/(i)/(i)/((ii *) m7)/(v
7)/(i)/(i))

 ? (substitue 10)
((i)/(i)/((v *) m7) ((v *) m7)/((iv# *)) ((iv *))/(iv)
((viib *))/(iv)/(i)/(i) ((iv *))/((ii *) m7) ((v *))/((i#
*)) ((i *))/(i) ((iv *))/(i))
```

Lorsque l'on augmente le nombre de règles, le processus de substitution arrive à saturation après une vingtaine de règles environ. Dans l'exemple ci-après, on affiche le nombre de règles effectivement appliquées. Après vingt-trois substitutions, toutes les mesures ont été divisées en deux (sauf la première et la dernière), ce qui rend inopérantes les règles 1 et 2. De plus, tous les accords de septième sont précédés d'un accord marqué * qui ne peut être affecté à la variable *w*, ce qui interdit l'application de la règle 3. La grille obtenue, assez éloignée du blues originel, donne une idée de la richesse maximale que l'on peut obtenir d'une grille après stabilisation du processus :

```
 ? (substitue 100)
1 = Regle3a-11/2 = Regle2-1/3 = Regle3a-21/4 = Regle3a-
21/5 = Regle1-1/6 = Regle1-3/7 = Regle4-11/8 = Regle2-
1/9 = Regle2-1/10 = Regle4-21/11 = Regle2-3/12 = Regle2-
2/13 = Regle2-1/14 = Regle1-1/15 = Regle4-2/16 = Regle1-
2/17 = Regle3b-11/18 = Regle3a-2/19 = Regle3a-
1/20 = Regle4-31/21 = Regle4-21/22 = Regle3b-
11/23 = Regle4-31/
```

```
(((viib *) m7)/((vi *) m7) ((v# *) m7)/((v *) m7) ((v *)
m7)/((v *) 7) ((iv# *))/(iv) (iv)/(iv) ((viib *))/(i)
((iv *))/((vi *) m7) ((ii *))/((ii *)) ((i# *))/((i# *))
((iv# *))/(i) ((iv *))/(i))
```

VIIbm7	VIm7/V#m7	Vm7	V7/IV#
IV	IV/VIIb	I/IV	VIm7/II
II7/I#	I#/IV#	I/IV	I

Tableau 6.3. *Une grille après saturation des règles*

6.4.3. *Arrangement des grilles pour piano*

Pour écouter une grille produite par le programme décrit au paragraphe 6.4.2, il est nécessaire de la jouer sur un instrument, c'est-à-dire d'en faire un *arrangement* à la façon de logiciels comme *Band-in-a-box*. L'utilisateur saisit une grille d'accords sous forme de chiffrages, et le logiciel calcule un fichier midi dans lequel la grille est arrangée pour les instruments fixés (guitare, basse, batterie, etc.) et dans le style choisi. De tels programmes fonctionnent avec une base de séquences musicales préenregistrées au format midi. Dans le programme Common Lisp présenté ici, le moteur d'application de régles décrit précédemment est complété par un arrangeur de grilles, permettant de réaliser les grilles pour piano solo dans le style boogie-woogie. Les règles traitées dans cette section sont une partie des règles 1, 3 et 4. On a supprimé la règle 2, qui a tendance à perturber le rythme harmonique des grilles comme on l'a dit plus haut, et on a éliminé également les parties des autres règles comportant des accords mineur septième, car la table d'arrangements n'en contient pas.

Formellement, cette composante du programme est une *transduction rationnelle*, au sens défini dans la première partie. Elle prend en entrée une séquence d'accords, et calcule en sortie une séquence de mesures arrangées pour piano. Au cours d'un traitement préliminaire, les accords de la grille sont regroupés en mesures. La succession de mesures ainsi obtenue est ensuite traitée selon le mécanisme de lecture suivant :

– si deux mesures consécutives sont occupées par un même unique accord, on cherche dans la base d'arrangements un motif qui dure deux mesures, et on poursuit la lecture deux mesures plus loin ;

– sinon, on cherche un motif pour la première mesure, et on poursuit la lecture en se décalant seulement d'une mesure.

On retrouve le même schéma de fonctionnement que celui de la figure 6.2. L'état 1 est initial. L'état 2 sert à mémoriser l'accord de la dernière mesure lue. Si la mesure suivante a le même accord, on tire un motif de deux mesures et on revient à l'état initial. Sinon, on tire un motif pour la mesure mémorisée précédemment, et on mémorise l'accord de la nouvelle mesure. Il y a en fait *plusieurs* états 2, associés à chacun des accords possibles (comme les états associés aux lettres 2*a* et 2*b* dans la figure 6.2). Dans le programme en Common Lisp, la fonction qui matérialise l'état 2 distingue ces différents états grâce à un argument supplémentaire.

Les deux composantes du programme, moteur d'application des règles et arrangeur de grilles, illustrent le principe de « factorisation » présenté dans la première partie (figure 6.3). Le programme est *factorisé* en deux parties, la partie harmonique réalisée par la grammaire de Steedman qui produit un « squelette » (la grille enrichie par substitutions), et la partie arrangement réalisée par la composante décrite dans ce paragraphe, qui habille ce squelette et produit un fichier midi (la séquence pour piano solo). La situation est la même que dans les programmes de Barbaud et Cope.

Dans le programme en Common Lisp reproduit en annexe, la fonction d'arrangement est appliquée à une progression harmonique engendrée par juxtaposition de plusieurs grilles, dont la première est la grille de base du blues (celle de l'axiome de Steedman), suivie d'enrichissements progressifs de celle-ci en cinq étapes réalisant quatre substitutions chacune. On peut bien sûr imaginer que l'utilisateur paramètre lui-même ce processus, en choisissant le nombre de grilles produites, et le nombre de substitutions passant d'une grille à la suivante, qui mesure la « vitesse » d'enrichissement.

La séquence harmonique obtenue est ensuite traitée par la transduction qui effectue des tirages dans une table d'arrangements contenant des motifs de piano, qui ont été échantillonnés en les jouant sur un clavier midi. Leur longueur est de une ou deux mesures. Dans la table, les motifs sont codés au format de l'environnement *Open Music* développé à l'Ircam par l'équipe de Gérard Assayag [ASS 97]. Les motifs sont représentés par quatre listes : hauteurs, dates absolues, durées, vélocités, les hauteurs et vélocités étant exprimées en valeurs midi, les dates et durées en multiples de la croche ternaire prise comme unité. Le générateur produit un quadruplet de quatre listes donnant les hauteurs, dates, durées et vélocités de l'ensemble de la séquence calculée. Pour la jouer dans *Open Music*, il suffit d'introduire ces quatre listes dans les entrées d'une boîte *ChordSeq* après avoir effectué quelques réglages :

– les hauteurs sont multipliées par 100 (midicents) ;

– les dates et durées sont converties en millisecondes par multiplication par un facteur 138 (correspondant à la durée de la croche ternaire à un tempo de 145 à la noire pointée : 138 = 60 000/(145 * 3)).

Les motifs échantillonnés dans la table d'arrangement peuvent subir deux transformations : transposition de hauteur et diminution de vélocité. La première permet d'adapter les motifs aux degrés de la grille, chaque motif étant enregistré avec un degré fixe indiqué dans la table. Les motifs correspondant à une mesure divisée en deux peuvent subir une transposition différente pour chaque moitié. La seconde transformation concernant les vélocités permet de rendre l'interprétation moins mécanique, plus vivante.

La table est conçue pour ne contenir qu'un nombre minimal de motifs. La grille est jouée dans la tonalité de Mi majeur. Tous les accords (majeur ou septième de dominante) sont réalisés comme septième de dominante (ce qui correspond au chiffrage 7' de Steedman évoqué plus haut). On a défini deux positions d'accords, l'une avec la tierce en haut pour les degrés V# à I (de Do à Mi), l'autre avec la septième en haut pour les degrés I# à V (de Fa à Si). La distinction de ces deux registres permet de réaliser l'harmonie de manière satisfaisante tout en gardant la table très compacte. Pour chaque accord, on choisit dans la table celui qui appartient au même registre, et on transpose les données midi en conséquence. La figure 6.8 présente en notation musicale une réalisation d'un chorus de blues.

6.4.4. *Analyse par automate fini avec Lex*

On s'intéresse dans ce paragraphe à l'implémentation de la grammaire de Steedman dans le sens de l'*analyse*, c'est-à-dire comme un programme capable de prendre une grille en entrée, et de décider si c'est un blues au sens de Steedman ou non. Le problème est que la grammaire de Steedman n'est ni algébrique, ni régulière. Exceptées les règles 1 et 2, toutes les autres règles comportent plusieurs symboles en partie gauche, contrairement à la condition définissant les règles de type 2 (et de type 3). Or, on a vu que seules les grammaires de type 2 et 3 sont équivalentes à des classes d'automates facilement implémentables (à pile ou finis).

Le problème peut être contourné par une analyse plus fine de la forme de cette grammaire, et par quelques hypothèse restrictives. Les règles contextuelles de la grammaire de Steedman (règles 3 et 4) ont une caractéristique essentielle : elles ont toujours *le même nombre de symboles* à gauche et à droite. Elles ont donc la propriété de conserver la longueur des séquences. Appliquées à un ensemble fini de séquences dont la longueur est bornée, elles produisent un ensemble dont la longueur est également bornée, donc ayant, lui aussi, un nombre fini d'éléments. Cette propriété de finitude montre que l'on peut décrire cet ensemble par une grammaire *régulière*.

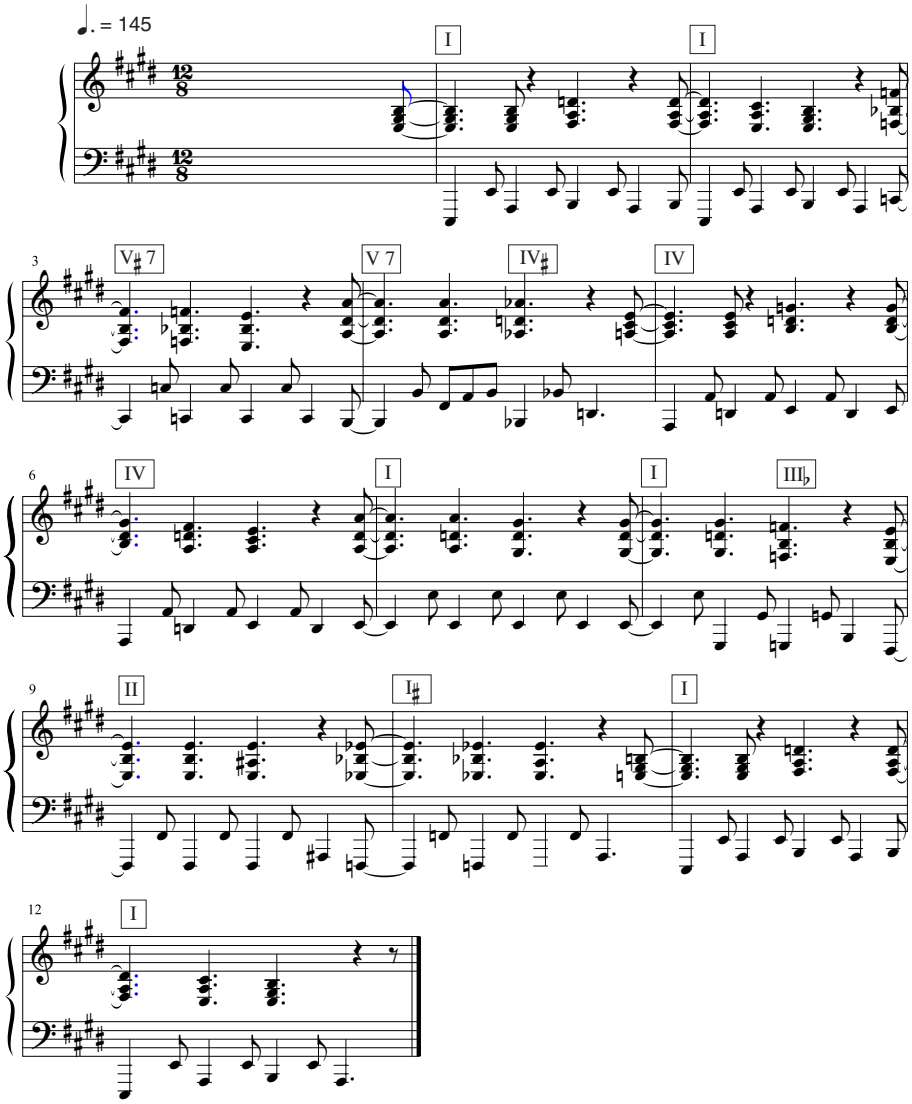

Figure 6.8. *Arrangement pour piano solo d'un chorus de blues*

Il suffit alors de restreindre l'action des autres règles (règles 1 et 2) pour que la grammaire de Steedman puisse être remplacée par une grammaire régulière. Cette restriction appliquée aux règles 1 et 2 peut être justifiée par le fait que les grilles sont rarement divisées au-delà de la demi-mesure. Plus précisément, la grille du

blues n'ayant que douze mesures, une dérivation qui n'introduit pas de quart de mesure ne peut pas comporter plus de douze applications des règles 1 ou 2.

On peut donc construire un automate fini reconnaissant un sous-ensemble du langage engendré par la grammaire de Steedman, ce sous-ensemble étant plus précisément l'intersection du langage engendré par cette grammaire avec le langage des grilles sans division en quart de mesures. On présente ici la construction d'un tel automate avec la commande Lex du système UNIX, en introduisant quelques simplifications supplémentaires dans les restrictions imposées aux règles 1 et 2. On suppose que la règle 1 n'est appliquée qu'en début de dérivation, aux six symboles donnés par l'axiome (pour fabriquer des mesures), puis aux quatre première mesures (pour fabriquer des demi-mesures uniquement dans la première ligne de la grille). Cela revient à limiter la génération aux séquences obtenues en appliquant les autres règles (2, 3 et 4) à la séquence :

I I/I I/I I/I I7/IV/IV/I/I/V/V7/I/I

Quant à la règle 2, nécessaire pour expliquer certains enchaînements de m7 à 7 (comme on le verra ci-après dans l'analyse de *Blues For Alice*), on limite son action en supposant qu'elle n'intervient qu'en fin de dérivation, seulement sous forme de réécriture d'un accord mineur septième.

La construction de l'automate s'effectue de proche en proche, à partir d'une chaîne reconnaissant la séquence ci-dessus. Les états sont les accords, et les flèches indiquent une transition d'un accord à un autre. Pour toute règle applicable à cette séquence (la règle 3a par exemple, I I7 $\rightarrow$ Vm7 I7, applicable à la quatrième mesure avec x = I), on rajoute dans l'automate les états nécessaires pour que la nouvelle séquence obtenue après substitution soit reconnue par l'automate. Ici, il suffit de rajouter un état étiqueté par l'accord Vm7, avec une flèche partant du I précédent, et une autre allant vers le I7 suivant. Lorsque l'on itère cette construction, il arrive un moment où aucune règle ne donne de nouvelles séquences non déjà reconnues par l'automate, puisqu'il n'y a qu'un nombre fini de séquences. A ce stade, la construction s'arrête.

L'automate est complété par quelques transitions supplémentaires. D'une part dans les quatre premières mesures, on relie un accord sur deux afin de reconnaître les grilles dont ces mesures ne sont pas divisées. D'autre part, on ajoute à la fin de la grille quelques *turnarounds* usuels, car dans la plupart des blues, la grille ne se termine pas sur l'accord I, mais sur l'accord V7 ou sur un enchaînement d'accords semi-cadentiel.

L'implémentation de cet automate avec Lex utilise de nombreuses variables auxiliaires pour représenter les différents accords possibles, ainsi que les étapes de la

construction. Les variables représentant les accords mineur septième peuvent se réécrire par la règle 2, en fin de dérivation. La construction de l'automate procède « à reculons » à partir des accords I7 et V7 des quatrième et dixième mesures comme une sorte de préparation amplifiée de ces deux accords cadentiels. Les variables matérialisant les étapes sont groupées en deux sous-séries, correspondant aux deux portions de la grilles traitées indépendamment (mesures un à quatre, mesures cinq à dix).

Une fois la commande Lex exécutée, et le code C compilé, on obtient une fonction réalisant l'automate. Cette fonction lit une grille, et affiche « reconnu », si c'est un blues reconnu par l'automate, et « non reconnu » dans le cas contraire. Elle est utilisée ici pour analyser quelques-uns des vingt-trois blues du *Charlie Parker Omnibook* (qui contient soixante thèmes de Charlie Parker). Ces blues du répertoire bop conviennent bien au style harmonique défini par la grammaire de Steedman, mais on va voir que malgré l'homogénéité de ce corpus, il apparaît une frontière entre des grilles reconnues par la grammaire et d'autres qui ne le sont pas.

Voici d'abord deux blues simples, *K.C. Blues* et *Now's The Time,* reconnus par la fonction :

```
$ blues
> I7/I7/I7/I7/IV7/IV7/I7/I7/IIm7/V7/I7/I7
Reconnu

$ blues
> I7/I7/I7/I7/IV7/IV7/I7/VI7/IIm7/V7/I7/I7
Reconnu
```

En revanche, la grammaire ne reconnaît pas *Barbados,* qui est pourtant un autre blues aussi simple. Le problème vient de ce que les règles de Steedman n'enrichissent la grille qu'en procédant de proche en proche, à partir des cadences des quatrième et dixième mesures. Le turnaround IIm7 V7 de la deuxième mesure ne peut pas s'expliquer, car il est isolé dans un contexte d'accords non cadentiels, et il ne peut être relié par dérivation à l'accord cadentiel I7 de la quatrième mesure.

```
$ blues
> I7/IIm7 V7/I7/Vm7 I7/IV7/IV7/I7/I/IIm7/V7/I7/IIm7 V7
Non reconnu
Mais la grille devient conforme aux règles si on le
supprime :
$ blues
> I7/I7/I7/Vm7 I7/IV7/IV7/I7/I/IIm7/V7/I7/IIm7 V7
Reconnu
```

Certains problèmes apparaissent du fait de restrictions trop grandes imposées dans notre analyseur à la règle 1, qui ne peut s'appliquer qu'aux quatre premières mesures. Ainsi, la grille de *Au Privave* n'est pas reconnue, alors qu'elle est pourtant conforme aux règles de Steedman. Il suffirait en effet d'appliquer la règle 1 à la huitième mesure pour la diviser en deux moitiés, ce qui permettrait ensuite d'expliquer l'enchaînement IIIm7 VI7, dont le rythme harmonique est plus resserré, grâce à la règle 3 :

```
$ blues
> I7/I7/I7/I7/IV7/IV7/I7/IIIm7 VI7/IIm7/V7/I7/IIm7 V7
Non reconnu
```

Les blues complexes, comme le célèbre *Blues For Alice,* ou *Laird Baird* dont la grille est assez proche, sont reconnus par la fonction, et confirment de ce fait la pertinence de la grammaire de Steedman. Ces deux blues utilisent la longue cascade de substitutions II-V qui prépare l'accord I7 de la quatrième mesure. Le second ne diffère du premier que par les accords des sixième et huitième mesures qui n'ont pas été réécrits par la règle 2 (*Blues For Alice* apparaît ainsi comme une dérivation de *Laird Baird*) :

```
$ blues
> I7/VIIm7 III7/VIm7 II7/Vm7 I7/IV7/IVm7
VIIb7/IIIm7/IIIbm7 VIb7/IIm7/V7/I7/IIm7 V7
Reconnu

$ blues
> I7/VIIm7 III7/VIm7 II7/Vm7
I7/IV7/IVm7/IIIm7/IIIbm7/IIm7/V7/I7 VI7/IIm7 V7
Reconnu
```

En revanche, la grille de *Bloomdigo* n'est pas reconnue, alors qu'il s'agit d'une simplification des deux grilles précédentes (la cascade des quatre premières mesures est supprimée). L'accord IVm7 de la sixième mesure ne peut pas s'expliquer s'il n'est pas inséré dans une suite d'accords mineur septième descendant chromatiquement comme dans les deux exemples précédents (il manque l'accord IIIm7, remplacé par I7, ce qui empêche de relier IVm7 et IIIbm7) :

```
$ blues
> I7/I7/I7/I7/IV7/IVm7/I7/IIIbm7/IIm7/V7/I7/IIm7 V7
Non reconnu
```

On voit que la grammaire de Steedman reconnaît une bonne proportion des grilles de blues de Charlie Parker, mais ne fournit pas un modèle de ce corpus complètement adéquat.

6.5. Automate sériel

6.5.1. *Régularité du langage sériel*

Cette dernière partie est consacrée à la modélisation de la règle de base de la musique sérielle, et à la construction d'un analyseur de séquences sérielles. Elle apporte un complément intéressant aux exemples précédents (Barbaud, Cope, Steedman), dans la mesure où le modèle d'automate fini sous-jacent au phénomène musical étudié ici n'est pas directement apparent. Alors que les substitutions des grilles de jazz s'exprimaient naturellement comme des règles de réécriture, la règle de base de la musique sérielle ne se ramène pas immédiatement aux modèles de grammaires et d'automates envisagés dans ce chapitre ; on va voir dans ce paragraphe que pour y parvenir, on est obligé de procéder à une formalisation plus poussée.

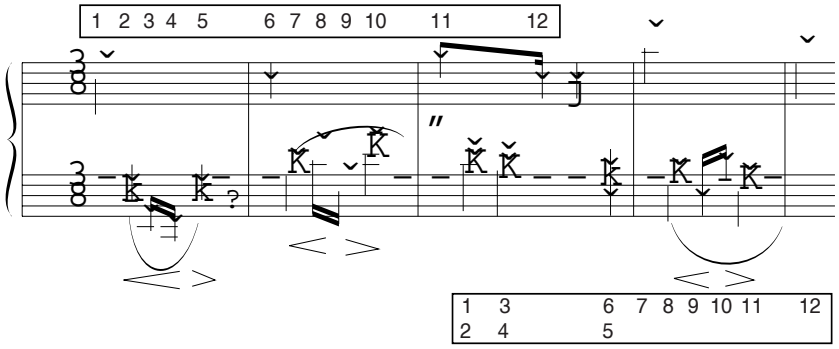

Figure 6.9. *Valse de l'opus 23 de Schoenberg, mesures 29 à 33*

La figure 6.9 est un exemple emprunté à Schoenberg permettant de rappeler la règle sérielle. La pièce est basée sur la série *u = do# la si sol lab fa# la# ré mi mib do fa*, et l'analyse consiste à numéroter les notes dans l'ordre où elles apparaissent dans *u*. Un premier décompte permet de mettre en évidence une première occurrence de la série, correspondant aux numéros 1 à 12 inscrits au-dessus de la portée. On recommence ensuite le même processus avec les notes restantes (notons que l'on peut le cas échéant réutiliser certaines notes déjà numérotées, mais ce n'est pas nécessaire dans cet exemple). Cette deuxième phase montre que les notes restantes constituent à elles seules une deuxième occurrence de la série, associée aux numéros 1 à 12 inscrits au-dessous de la portée. Dire que la séquence est « sérielle », cela revient à dire que l'on peut réitérer le processus de numérotation jusqu'à épuisement de toutes les notes contenues dans la séquence étudiée.

Cette règle de répartition des sons dans l'ordre fixé par une série apparaît historiquement pour la première fois dans cette *Valse* de Schoenberg. A l'époque, celui-ci voulait, entre autres choses, structurer l'univers harmonique atonal qui s'était développé durant les années dix. L'application de la règle sérielle tend en effet, à favoriser certaines configurations d'intervalles qui déterminent la couleur harmonique de la séquence. Plus précisément, les notes simultanées d'une séquence sérielle sont principalement des notes consécutives de la série, ce qui donne à celle-ci un rôle que l'on pourrait qualifier de « réservoir harmonique ». Par exemple, dans la série précédente, les tierces majeures sont favorisées au début de la série (*do# la*) et (*si sol*), puis au milieu sous la forme (*fa# la# ré*), et enfin en bouclant la fin avec le début (*fa la do#*). L'une des configurations préférées des musiciens sériels est l'accord quarte augmentée/quarte juste, qui a été repris abondamment par les pianistes de jazz depuis les années soixante (Herbie Hancock, Chick Corea). La série *fa# fa la sol sib sol# ré do# do mi ré# si* utilisée par Webern dans les *Variations pour piano* op. 27 permet cet accord « fétiche » sous la forme (*ré sol# do#*).

Il est possible de montrer que le langage des séquences construites selon la règle sérielle énoncée ci-dessus est *reconnaissable par automate fini*. Ce résultat a été démontré dans [CHE 90], puis développé dans [CHE 02]. Il ne conduit pas directement à une construction de l'automate qui soit utilisable dans la pratique, mais il repose sur une formalisation de la notion de séquence sérielle (dans sa version simpliste, restreinte à l'énoncé de la règle élémentaire illustrée figure 6.9) à partir de laquelle on peut déduire certaines propriétés caractéristiques. La principale propriété est qu'une séquence est sérielle dans ce sens restreint si et seulement si *toute note apparaissant dans cette séquence est précédée et suivie des notes qui la précèdent et la suivent dans la série*. Cette propriété traduit l'idée intuitive de « plénitude chromatique », qui est implicitement recherchée dans l'utilisation systématique d'une même permutation de douze notes. On peut également formuler cette propriété de manière négative, en disant qu'une séquence n'est pas sérielle si et seulement si il existe une paire de notes (x, y) telle que x précède y dans la série, et telle que l'une des deux conditions suivantes soient vérifiées :

(1) x apparaît dans la séquence, et y n'apparaît pas après x ;

(2) y apparaît dans la séquence, et x n'apparaît pas avant y.

En notant A_x l'ensemble des agrégats qui contiennent la note x, la propriété précédente se traduit sous la forme d'une *expression rationnelle* définissant l'ensemble des séquences non sérielles. En effet, celui-ci est égal à une union d'ensembles qui sont eux-mêmes définis par des expressions rationnelles. Plus précisément, l'ensemble des séquences non sérielles est égal à l'union pour toutes les paires de notes (x, y) telles que x précède y dans la série, des ensembles de la forme $P(x, y) \cup S(x, y)$, où P et Q correspondent respectivement aux conditions (1) et (2) ci-dessus, et sont définis par les expressions rationnelles suivantes :

$$P(x,y) = A^*\left(A_x \cap \overline{A}_y\right)\overline{A}_y\,^*$$

$$S(x,y) = \overline{A}_x\,^*\left(A_y \cap \overline{A}_x\right)A^*$$

L'expression rationnelle ainsi obtenue permet de construire un automate reconnaissant l'ensemble des séquences sérielles. Effectuons l'union partielle des ensembles $P(x, y)$, en faisant varier x pour y fixé. On obtient l'égalité suivante, où P_y désigne l'ensemble des agrégats qui ne contiennent pas y, mais qui contiennent un prédécesseur de y dans la série :

$$\bigcup_x P(x,y) = A^*\left(\left(\bigcup_x A_x\right)\cap \overline{A}_y\right)\overline{A}_y\,^* = A^* P_y \overline{A}_y\,^*$$

De la même façon, effectuons l'union partielle des ensembles $S(x, y)$, en faisant varier y pour x fixé. On obtient une égalité similaire, où S_x désigne l'ensemble des agrégats qui ne contiennent pas x, mais qui contiennent un successeur de x dans la série :

$$\bigcup_y S(x,y) = \overline{A}_x\,^*\left(\left(\bigcup_y A_y\right)\cap \overline{A}_x\right)A^* = \overline{A}_x\,^* S_x A^*$$

Finalement, l'ensemble des séquences non sérielles est égal à :

$$\left(\bigcup_y A^* P_y \overline{A}_y\,^*\right)\cup\bigcup_x \overline{A}_x\,^* S_x A^*$$

Pour chaque terme apparaissant dans l'expression ci-dessus, il est facile de faire un automate reconnaissant le langage correspondant. Dès lors, on peut en déduire un automate reconnaissant le complémentaire de chacun de ces termes, puis construire l'automate de l'intersection de ces langages, qui donne finalement un automate pour le langage sériel lui-même.

6.5.2. *Programme d'analyse par automate fini*

Dans ce paragraphe, on présente un analyseur capable de détecter si une séquence est sérielle ou non (on suppose ici que la série est donnée, la recherche d'une série associée à une séquence supposée sérielle étant un problème beaucoup plus complexe). Comme on l'a vu au paragraphe 6.5.1, cet analyseur peut être décrit

comme une combinaison d'automates, et il reconnaît une séquence comme sérielle dès lors que celle-ci est acceptée par chacun des automates intervenant dans cette combinaison.

On considère figure 6.10 l'automate reconnaissant le complémentaire du langage défini par l'expression rationnelle suivante, pour une note *y* donnée :

$$A^* P_y \overline{A}_y ^*$$

Pour obtenir un automate déterministe, on introduit le complémentaire M_y de l'union de A_y et P_y, de telle sorte que l'ensemble de tous les agrégats soit égal à l'union de M_y, A_y et P_y. L'automate vérifie que toute note précédent *y* dans la série est bien suivie de *y* dans une séquence sérielle. Il a deux états et fonctionne de la manière suivante :

– tant que la séquence lue ne contient pas de note précédent *y*, on reste dans l'état 1 ;

– si on lit une note précédent *y* sans lire en même temps *y*, on passe dans l'état 2 ;

– dans l'état 2 :

- si on lit *y,* condition nécessaire pour que la séquence soit sérielle, on revient à l'état 1 qui est terminal ;

- si l'on ne lit aucun *y* jusqu'à la fin de la séquence, en bouclant sur l'état 2, celle-ci n'est pas sérielle.

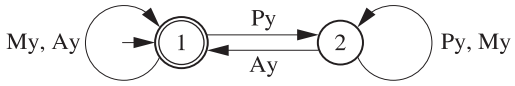

Figure 6.10. *Automate fini contrôlant les notes qui précèdent y dans la série (*P_y*)*

On considère maintenant l'automate représenté figure 6.11, reconnaissant le complémentaire du langage défini par l'expression rationnelle suivante, pour une note *x* donnée :

$$\overline{A}_x ^* S_x A^*$$

Comme précédemment, on introduit le complémentaire N_x de l'union de A_x et S_x, de telle sorte que l'ensemble de tous les agrégats soit égal à l'union de N_x, A_x et S_x, ce qui permet d'obtenir un automate déterministe. Théoriquement, l'automate vérifie

que toute note suivant *x* dans la série est bien précédée de *x* dans une séquence sérielle. Les états 1 et 2 devraient donc être terminaux, mais si l'on ajoute une condition supplémentaire selon laquelle une séquence sérielle contient au moins une fois chacune des notes *x*, cela revient à rendre l'état 1 non terminal. Notons que les états 2 et 3 sont des « puits » dont on ne peut sortir après y être entré, et dans lesquels on peut arrêter la lecture. Le fonctionnement est le suivant :

– tant que la séquence lue ne contient ni *x*, ni une note suivant *x*, on reste dans l'état 1 ;

– si on lit *x*, la lecture s'arrête car toute note apparaissant ultérieurement sera précédée de *x* (en particulier celles qui suivent *x* dans la série, condition pour que la séquence soit sérielle) ;

– si on lit une note suivant *x* sans avoir lu *x*, la lecture s'arrête également, car la séquence n'est pas sérielle.

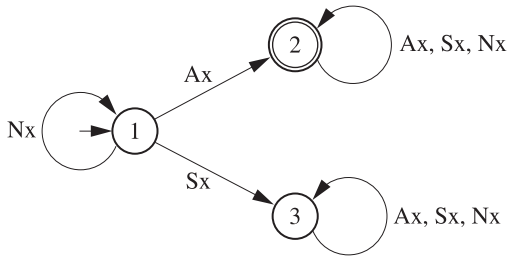

Figure 6.11. *Automate fini contrôlant les notes qui suivent x dans la série (*S_x*)*

L'analyseur reconnaît une séquence comme sérielle si et seulement si pour chaque note de la série, cette séquence est acceptée par les automates des figures 6.10 et 6.11 (on fait l'intersection des langages correspondants). Malheureusement, on ne peut construire avec Lex l'automate résultant de cette combinaison d'automates associés aux douze notes de la série, car cela supposerait un calcul explicite de la table de transition. Or, celle-ci est trop grande pour que le calcul soit envisageable, le nombre d'agrégats possibles (donc d'étiquettes sur les flèches) dépassant les capacités de Lex. On est ainsi obligé d'implémenter l'automate de façon dynamique, en faisant fonctionner séparément chacun des automates associés aux douze notes de la série. C'est ce qui est réalisé ici dans un programme en Common Lisp utilisé pour analyser deux passages comportant des anomalies, dans la *Valse* de l'opus 23 de Schoenberg, construite sur la série indiquée plus haut *u = do# la si sol lab fa# la# ré mi mib do fa*.

L'analyseur détecte une anomalie dans la partition aux mesures 13 à 17, où il manque un *ré* dans le deuxième énoncé de la série, avant l'accord (*mib mi*) :

```
 ? (seriel ? '(reb la si sol lab fa# sib re mi mib (reb
do) fa la si (lab sol) solb sib (mib mi) do fa))
Il manque un ré en partant de la fin
nil
```

Remarquons que l'anomalie n'est plus détectée si l'on prolonge l'extrait analysé par un troisième énoncé de la série jusqu'à la mesure 21. Cet énoncé supplémentaire apporte le *ré* manquant, et la séquence devient sérielle :

```
 ? (seriel ? '(reb la si sol lab fa# sib re mi mib (reb
do) fa la (re si) (lab sol) solb sib (mib mi) do fa do#
la si sol lab solb sib re mi mib (fa do) do# la (sol si
lab solb) (sib re) (mi mib) (fa do)))
La séquence est sérielle
t
```

Il s'agit là d'une faiblesse de l'analyseur, qui n'est pas capable de contrôler la « proximité » relative des notes constituant un énoncé de la série. On ne peut donc analyser la pièce avec un seul traitement. Pour que l'analyseur fonctionne de manière pertinente, il faut l'appliquer à des segments relativement courts, regroupant des énoncés apparemment complets de la série. L'analyseur se charge alors de vérifier qu'aucune note ne manque dans ces énoncés, et qu'elles sont bien disposées dans l'ordre requis.

Une autre anomalie de la partition de Schoenberg se trouve dans l'extrait qui va du deuxième accord de la mesure 63 jusqu'au dernier accord de la mesure 65. Mais dans la segmentation ainsi définie, l'anomalie est masquée par le dernier énoncé de la série qui est tronqué. A la mesure 65, en effet, l'extrait se termine par les deux premières notes d'un énoncé incomplet interrompu avant le *si* :

```
 ? (seriel ? '((la do#) (sol# si sol) (re fa# la# mi)
(do# fa do mib) (la si sol) (lab re fa# sib) (mib fa dob
mi) (la do# sol si) (lab re fa# sib) (la reb fa do mib
mi)))
Il manque un si en partant de la fin
nil
```

Pour faire apparaître l'anomalie, il faut éliminer cet énoncé incomplet, en supprimant les trois derniers accords, de telle sorte que l'extrait se termine exactement à la fin de la série. Dans ce nouvel extrait, délimitant en principe deux occurrences de la série, l'analyseur détecte qu'il faut un *do* à la place du *dob* dans le dernier accord (*mib fa dob mi*) :

```
 ? (seriel ? '((la do#) (sol# si sol) (re fa# la# mi)
(do# fa do mib) (la si sol) (lab re fa# sib) (mib fa dob
mi)))
Il manque un do en partant de la fin
Nil
```

Cet analyseur rudimentaire basé sur des automates finis réalise correctement la tâche consistant à détecter si une séquence est sérielle dans certains cas simples comme ceux correspondant aux deux anomalies étudiées précédemment dans des segments découpés de façon *ad hoc*. C'est le problème de cet analyseur, qui ne fonctionne correctement que sur le plan local, avec des extraits courts comportant des énoncés supposés complets de la série. Cette limite vient de la formalisation de la règle sérielle proposée initialement, qui ne comporte aucun contrôle de la proximité relative des notes constituant un énoncé de la série. La prise en compte de ces conditions de proximité serait nécessaire pour parvenir à une analyse sérielle plus fine. Mais il ne semble pas que les théories analytiques développées autour des musiques atonales et sérielles (notamment dans la musicologie américaine) n'aient jamais résolu de façon satisfaisante ces questions de segmentation et de proximité des notes formant une unité structurelle. Et d'ailleurs, si elles y parvenaient en fournissant une procédure de segmentation explicite basée sur des critères précis, il est probable que leur implémentation nécessiterait un modèle de calcul plus complexe que celui des automates finis.

6.6. Bibliographie

[AHO 89] AHO A., SETHI R., ULLMAN J., *Compilateurs. Principes, techniques et outils*, 1986, InterEditions, Paris, 1989.

[ASS 97] ASSAYAG G., AGON C., Logiciel OpenMusic, Cdrom Forum Ircam, 1997.

[BAR 65] BARBAUD P., *Introduction à la composition musicale automatique*, Dunod, Paris, 1965.

[BAR 68] BARBAUD P., *La musique, discipline scientifique*, Dunod, Paris, 1968.

[BAR 75] BARBAUD P., Ludus margaritis vitreis, rapport Inria, 1975.

[BAR 93] BARBAUD P., *Vademecum de l'ingénieur en musique*, Springer, Paris, 1993.

[BAR 98] BARBAUD P., *Schoenberg*, Editions Main d'Œuvre, Nice, 1998.

[BAR 84] BARONI M., CALLEGARI L. (DIR.), *Musical grammars and computer analysis*, Leo S. Olschki, Firenze, 1984.

[BEC 79] BECKER A., BECKER J., « A Grammar of the Musical Genre Srepegan », *Journal of Music Theory*, vol. 23, 1979, repris dans *Asian Music*, vol. 14, n° 1, 1983.

[BEL 92] BEL B., KIPPEN J., « Bol Processor Grammars », M. Balaban, K. Ebcioglu, O. Laske (dir.), *Understanding Music with AI*, AAAI Press, p. 366-401, 1992.

[BRO 98] BROWN F., « Les trois âges de la musique par ordinateur tels que nous les avons vécus », dans M. Chemillier, F. Pachet (dir.), *Recherches et applications en informatique musicale*, Hermès, Paris, 1998.

[CHE 87] CHEMILLIER M., « Monoïde libre et musique », *RAIRO Informatique théorique*, vol. 21, n° 3 et n° 4, 341-371, 379-417, 1987.

[CHE 88] CHEMILLIER M., TIMIS D., « Toward a theory of formal musical languages », *Proceedings of the ICMC 88*, Cologne, p. 175-183, 1988.

[CHE 90] CHEMILLIER M., Structure et méthode algébriques en informatique musicale, Thèse, Université Paris 7, 1990.

[CHE 90b] CHEMILLIER M., « Langages musicaux et automates : la rationalité du langage sériel », *Actes du colloque Musique et assistance informatique*, Marseille, p. 302-318, 1990.

[CHE 92] CHEMILLIER M., « Automata and music », *Proceedings of the ICMC 92*, San José, p. 370-371, 1992.

[CHE 98] CHEMILLIER M., PACHET F. (dir.), *Recherches et applications en informatique musicale*, Hermès, Paris, 1998.

[CHE 99] CHEMILLIER M., « Générateurs musicaux et singularités », *JIM 99, 6e journées d'informatique musicale*, CNET-CEMAMu, Issy-les-Moulineaux, p. 167-177, 1999.

[CHE 02] CHEMILLIER M., Synchronisation of musical words*, Theoretical Computer Science*, à paraître.

[CHE 03] CHEMILLIER M., Grammaires de substitutions de jazz, à paraître.

[CHO 56] CHOMSKY N., « Three models for the description of language », *IRE Transactions on information theory*, IT-2, 1956.

[CHO 57] CHOMSKY N., *Structures syntaxiques*, 1957, trad. coll. Points, Seuil, Paris, 1969.

[COK 64] COKER J., *Improvising Jazz*, 1964, rééd. Fireside, New York, 1987.

[COP 91] COPE D., *Computers and Musical Style*, A-R Editions, Madison, 1991.

[COP 96] COPE D., *Experiments in Musical Intelligence*, A-R Editions, Madison, 1996.

[COP 99] COPE D., *The Algorithmic Composer*, A-R Editions, Madison, 1999.

[CRO 91] CROSS I., HOWELL P., WEST R. (DIR.), *Representing Musical Structure*, Academic Press, Londres, 1991.

[EIL 74] EILENBERG E., *Automata, languages and machines*, Academic Press, Londres, 1974.

[GRE 73] GREUSSAY P., Modèles de descriptions symboliques en analyse musicale, Thèse de doctorat, Université Paris 8, 1973.

[GRO 69] GROSS M., LENTIN A., *Notions sur les grammaires formelles*, Gauthier-Villars, Paris, 1969.

[GRO 89] GROSS M., PERRIN D. (DIR.), *Electronic Dictionnaries and Automata in Computational Linguistics*, Springer, Berlin, 1989.

[HUG 91] HUGHES D.W., « Grammars of Non-Western Music: A Selective Survey », I. Cross, P. Howell, R. West (dir.), *Representing Musical Structure*, Academic Press, Londres, 327-362, 1991.

[JOH 91] JOHNSON-LAIRD P., « Jazz Improvisation: A Theory at the Computational Level », dans I. Cross, P. Howell, R. West (dir.), *Representing Musical Structure*, Academic Press, Londres, p. 291-326, 1991.

[KLE 56] KLEENE S.C., « Representation of events in nerve nets and finite automata », dans C.E. Shannon, J.M. McCarthy (dir.), *Automata Studies*, Princeton University Press, Princeton, p. 3-42, 1956.

[LAS 73] LASKE O.E., « In Search of a Generative Grammar for Music », *Perspectives of New Music*, p. 351-378, 1973.

[LER 83] LERDAHL F., JACKENDOFF R., *A Generative Theory of Tonal Music*, MIT Press, Cambridge, 1983.

[MAR 13] MARKOV A.A., « Essai d'une recherche statistique sur le texte du roman *Eugène Oneguine* », *Bulletin de l'Académie Impériale des Sciences de Saint-Petersbourg*, vol. 7, 1913.

[MOO 72] MOORER J.A., « Music and Computer Composition », *Communications of the ACM*, vol. 15, n° 2, p. 104-113, 1972.

[MOU 95] MOUTON R., Outils intelligents pour les musicologues, Thèse de doctorat, Université Paris 1, 1995.

[NAT 75] NATTIEZ J.-J., *Fondements d'une sémiologie de la musique*, 10/18, Christian Bourgois, Paris, 1975.

[PAC 98a] PACHET F., « Computer Analysis of Jazz Chord Sequences: Is Solar a Blues ? », dans E.R. MIRANDA (dir.), *Readings in Music and AI*, Harwood, Paris, 1998.

[PAC 98b] PACHET F., CARRIVE J., « Intervalles temporels circulaires et application à l'analyse harmonique », dans M. Chemillier, F. Pachet (dir.), *Recherches et applications en informatique musicale*, Hermès, Paris, p.17-30, 1998.

[PAC 98c] PACHET F., « Sur la structure algébrique des séquences d'accords de jazz », *JIM 98, 5e journées d'informatique musicale*, LMA Marseille, La Londe-les-Maures, 1998.

[PAC 99] PACHET F., « Surprising Harmonies », *JIM 99, 6e journées d'informatique musicale*, CNET-CEMAMu, Issy-les-Moulineaux, p. 187-206, 1999.

[PAR 78] PARKER C., *Charlie Parker Omnibook*, Atlantic Music Corp., New York, 1978.

[PER 93] PERRIN D., « Les débuts de la théorie des automates », *Séminaire Philosophie et Mathématiques*, ENS, janvier 1993.

[RAH 93] RAHN J., « Le compositeur et ses AMIs », *Les cahiers de l'Ircam*, n° 3, 1993.

[RIO 79] RIOTTE A., Formalisation des structures musicales, Polycopié, Université Paris 8, 1979.

[ROA 79] ROADS C., « Grammars as Representations for Music », *Computer Music Journal*, vol. 3, n° 1, 1979, p. 48-55, repris dans C. Roads, J.Strawn (dir.), *Foundations of Computer Music*, MIT Press, Cambridge, 1985.

[ROA 84] ROADS C., « An overview of music representations », dans M. Baroni, L. Callegari (dir.), *Musical grammars and computer analysis*, Leo S. Olschki, Firenze, p. 7-37, 1984.

[ROA 96] ROADS C., *The Computer Music Tutorial*, MIT Press, Cambridge, 1996.

[RUW 71] RUWET N., *Langage, musique, poésie*, Seuil, Paris, 1971.

[SCH 93] SCHWANAUER S., LEVITT D. (DIR.), *Machine Models of Music*, MIT Press, Cambridge, 1993.

[SNE 85] SNELL J.L., « Musical Grammars and Computer Analysis: A Review », *Perspectives of New Music*, vol. 23, n° 2, p. 220-234, 1985.

[STE 84] STEEDMAN M.J., « A Generative Grammar for Jazz Chord Sequences », *Music Perception*, vol. 2, n° 1, p. 52-77, 1984.

[STE 96] STEEDMAN M.J., « The Blues and the Abstract Truth: Music and Mentals Models », dans A. Garnham, J. Oakhill (dir.), *Mentals Models in Cognitive Science*, Erlbaum, Mahwah, 1996.

[SUN 70] SUNDBERG J., LINDBLOM B., « Towards a generative theory of melody », *Swedish Journal of Musicology*, vol. 52, p. 71-88, 1970.

[SUN 91] SUNDBERG J., LINDBLOM B., « Generative Theories for Describing Musical Structure », dans I. Cross, P. Howell, R. West (dir.), *Representing Musical Structure*, Academic Press, Londres, p. 245-272, 1991.

[WIN 68] WINOGRAD T., « Linguistics and the Computer Analysis of Tonal Harmony », *Journal of Music Theory*, 1968, repris dans S. Schwanauer, D. Levitt (dir.), *Machine Models of Music*, MIT Press, Cambridge, 1993.

[XEN 63] XENAKIS I., « Musique stochastique markovienne », dans *Musiques formelles*, 1963, Stock, Paris, p. 61-131, 1981.