UNIVERSITÀ DI ROMA " LA SAPIENZA "

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI

Corso di Laurea in Informatica

TESI DI LAUREA

Grammatiche di sincronizzazione per generare l'esposizione di una fuga

RELATORE — Daniele Gorla

prof.ssa Anna Labella — matr. 11103573

Anno Accademico 1999 – 2000

Indice

INTRODUZIONE

La presente tesi si inserisce nel filone della ricerca informatica e matematica che tenta di modellare vari aspetti del linguaggio musicale con procedimenti formali e algoritmici; in particolare si aggiunge ai vari lavori che nell'ultimo trentennio si sono occupati di generare composizioni secondo forme e regole tradizionali. Un quadro più ampio degli aspetti dell'informatica musicale è data in questa introduzione.

L'informatica musicale fino ad oggi

Negli ultimi quarant'anni la cultura e la ricerca umana hanno subito una radicale trasformazione : è infatti ormai lecito parlare di una rivoluzione tecnologica che ha caratterizzato l'ultimo dopoguerra ma che si è prevalentemente sviluppata nell'ultimo quarto di secolo. Tale sconvolgimento ha toccato più o meno tutti i campi di indagine della nostra tradizione culturale, oltre ad aggiungerne nuovi. Non stupisce quindi che problemi in apparenza lontanissimi dall'ambito tecnologico e formale siano stati riformulati per essere studiati con queste nuove risorse. La musica effettivamente rientra in questo gruppo : ad una prima sommaria analisi sembra una disciplina puramente artistica ed umanistica, mentre in realtà è particolarmente adatta ad un approccio scientifico e formale. Le prossime pagine illustreranno perché e faranno un breve riassunto dei risultati ottenuti negli ultimi decenni.

Musica e linguaggio

I primi importanti esperimenti compiuti verso la formalizzazione di aree apparentemente lontane dall'ambito scientifico sono quelli della linguistica strutturale e generativo– trasformazionale compiuti da vari studiosi negli anni cinquanta ed in particolare da Noam Chomsky. Seguendone i passi, si sono tentati approcci in ambito musicale : effettivamente tale disciplina può essere considerata un linguaggio nel senso che è un mezzo di comunicazione tra più persone in base a precise regole, unendo quindi una sintassi ad un significato. Quindi è logico pensare di seguire i passi fatti dall'informatica teorica nello studio dei vari linguaggi (formali e naturali) e tentare di adattarne le metodologie nell'analisi musicale. Moltissimi lavori sono stati fatti in proposito, sia da matematici che da linguisti : tra questi [Net58] e [Her76] .

Questa idea è inoltre sostenuta da varie somiglianze tra linguaggio musicale e naturale : anzitutto la presenza di una struttura sintattica molto ricca che offre la possibilità di elaborare frasi complesse e varie. Le relative grammatiche condividono inoltre l'alta predisposizione all'ambiguità in fase di analisi : è proprio questo aspetto che le rende così versatili e ricche di possibilità, ma ne è anche uno dei peggiori difetti in ambito formale.

Inoltre sia il fenomeno musicale che quello linguistico rimandano a convenzioni culturali più o meno esplicitamente dichiarate : i suoni emessi sono spesso gli stessi mentre cambiano i rapporti tra di essi e i significati che le diverse successioni assumono. Può capitare infatti che lingue diverse associno a parole uguali significati diversi (i cosiddetti *false–friends* inglesi per esempio) oppure che alcune successioni di accordi siano lecite in una tradizione musicale ed inesistenti in un'altra. Ecco perché in [Bar81] si parla di musica come di un codice, nel senso semiotico del termine (cioè l'associazione

tra strutture sintattiche e semantiche realizzata tramite convenzioni culturalmente accettate), riprendendo argomentazioni già presenti in [Bri71] .

Infine, come con il linguaggio naturale, ci sono due possibili approcci al problema : il primo consiste nell'analizzare frasi (o brani musicali) già esistenti con lo scopo di metterne in evidenza simmetrie e strutture; il secondo consiste nel generare frasi (o brani) corretti secondo le regole imposte dal linguaggio. Tali approcci in informatica vengono definiti *analitico* e *generativo* e servono rispettivamente per analizzare o creare frasi (o brani musicali). Come però fa notare [Bar81] , questi due aspetti sono correlati : un approccio analitico estrapola, analizzando dei testi dati, delle regole strutturali in grado di descriverli; le stesse regole sono poi usate per creare nuovi brani la cui bontà misura il livello di correttezza delle regole così derivate. L'evoluzione di questi esperimenti è contenuta in [B,D,J99] .

Il legame con i linguaggi naturali non è però così profondo : molte sono le differenze, alcune importanti dal nostro punto di vista, altre meno. Un aspetto di grande differenza che ha reso così interessante lo studio formale e scientifico della musica sta nel fatto che essa vive un rapporto molto stretto con i numeri : tutti gli aspetti del fenomeno in questione sono basati su forti considerazioni numeriche, al contrario dei linguaggi naturali. Ecco una breve sintesi di questa situazione.

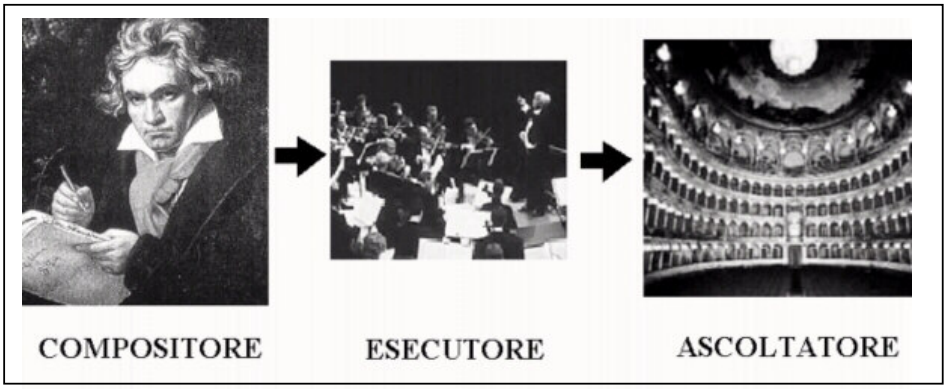
- ***Ritmo :*** le durate delle note sono viste come frazioni di una unità di riferimento (solitamente la semibreve) e, nella divisione regolare, sono organizzate come un albero binario (ogni durata è la metà della durata del livello precedente). Quindi il ritmo di un brano è essenzialmente una successione di frazioni dell'unità opportunamente organizzate (N.B.:

l'organizzazione delle durate in gruppi di uguale valore – le battute – è entrata nella musica occidentale solo dal Rinascimento : il gregoriano riprendeva l'idea della metrica poetica, utilizzando durate brevi e lunghe in base alla declamazione del testo cantato).

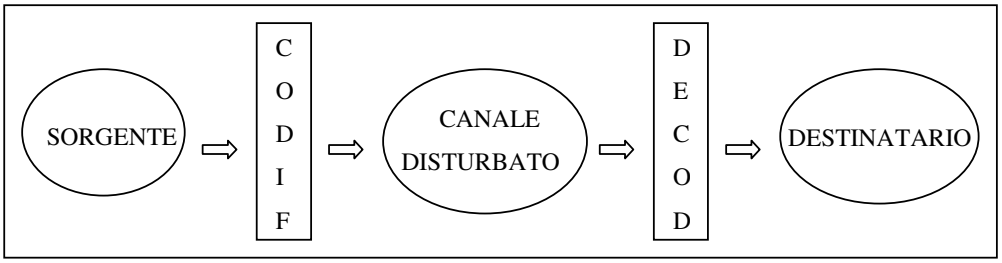
- ***Melodia :*** gli intervalli che una data voce compie sono identificati numericamente (intervalli di terza, di quarta, etc…) e sono misurabili come il numero di semitoni che separano le note di partenza e di arrivo.

- ***Armonia :*** la possibilità di emettere più suoni simultaneamente è regolata da leggi della fisica acustica che trovano il loro fondamento nei rapporti tra le frequenze della nota fondamentale e delle armoniche superiori. La successione di più accordi è regolata da leggi basate sui rapporti tra i vari gradi della scala (tradizionalmente identificati con numeri romani) e sulle possibili armonizzazioni di un dato grado (espresse invece con uno o più numeri arabi).

Un'altra importante differenza che complica il modello di comunicazione cui la musica soggiace sta nel fatto che nel linguaggio comune il rapporto tra sorgente e destinatario è diretto, mentre nella musica c'è bisogno di una figura intermedia (l'esecutore). La situazione tipica è quella riportata nella seguente figura



Il modello di comunicazione prevede un compositore che scrive la musica, un esecutore che la suona ed uno o più ascoltatori che ascoltano. Ovviamente due o addirittura tutte e tre queste figure possono fisicamente coincidere, pur restando concettualmente distinte. Tale modello rispecchia bene quello previsto da Shannon riportato nella seguente figura



La sorgente è chiaramente il compositore che codifica le sue idee con le notazioni musicali usuali. Il canale è l'esecutore ed è "disturbato" nel senso che interviene direttamente sul brano eseguito, mettendone più o meno in evidenza alcuni aspetti, caratterizzandone temi, dinamiche e strutture a seconda del suo gusto e della sua personalità. Il destinatario è l'ascoltatore che compie il processo di decodifica tramite le orecchie (che percepiscono i suoni emessi dall'esecutore) ed il cervello (che elabora, tramite la propria sensibilità e cultura, il segnale acustico che arriva). Questa forte somiglianza, unita al fatto

che la musica è un fenomeno markoviano (nel senso che, nell'ascolto di una composizione, il presente è determinato solo da un passato più o meno prossimo), ha portato negli anni sessanta a compiere vari studi per collegare teoria dell'informazione e musica (per esempio [Mey57] , [Pin56] , [Coh62])

Una terza importante differenza tra linguaggio musicale e naturale sta nel fatto che il secondo è solitamente usato per trasmettere messaggi più o meno semplici, mentre il primo è propriamente usato come forma d'arte. La differenza è grande anche da un punto di vista formale : mentre il creare ed interpretare frasi anche non particolarmente raffinate nel linguaggio naturale è già un grande risultato, creare ed analizzare musiche semplici ed elementari è un obiettivo poco soddisfacente da un punto di vista pratico. In quest'ottica, inoltre, è facile convincersi del perché l'analisi musicale sia stato il primo aspetto analizzato : dal punto di vista artistico è decisamente più semplice analizzare il capolavoro di un grande autore piuttosto che crearne uno dal nulla (e questo vale anche senza l'uso dell'elaboratore). Ciò crea anche una grande differenza con l'approccio informatico ai linguaggi naturali : in questi ultimi l'aspetto più semplice è la generazione di frasi, essendo possibile anche con frasi poco strutturate comunicare moltissimi concetti. Nella musica purtroppo questo non è vero: per esprimere qualcosa di interessante bisogna per forza dar vita a brani articolati e complessi.

Infine è importante notare come esista una profonda differenza tra una grammatica per il linguaggio ed una per la musica : nel primo caso il fulcro del processo generativo risiede nello stabilire quali stringhe appartengono al linguaggio; nel secondo caso questo non è sufficiente. Infatti, essendo la musica una forma d'arte che soggiace a delle precise convenzioni e regole inconsce, non basta che un brano di musica sia corretto sintatticamente : deve

corrispondere al modo di ascoltare tipico della nostra cultura (deve essere cioè coerente con la tradizione musicale occidentale). Questa differenza è motivata dal fatto che un linguaggio naturale è stato creato per esprimere concetti e significati; la musica, al contrario, è una forma d'arte finalizzata a comunicare emozioni in base ad un sistema di convenzioni culturali acquisite nel tempo. E' per questo motivo che spesso, nei lavori che affrontano il problema dello studio della musica tramite grammatiche generative (per esempio [Bar81] e [L,J83]) si incontrano due tipi di regole : quelle di "ben–formatezza" e quelle "preferenziali". Le prime servono a descrivere gli aspetti corretti di un brano musicale, le seconde per descrivere tra le soluzioni corrette, quelle preferibili per coerenza con la nostra tradizione. E' comunque necessario precisare anche in quest'ambito che tale distinzione è un fatto assolutamente non vincolante per un genio : egli spesso viola queste forme preferenziali per dare alla sua musica un elemento di sorpresa, finalizzando questa infrazione all'espressione di una particolare situazione emotiva. La creazione automatica di musica è quindi interessante solo per evidenziare al massimo le regole formali che governano la composizione : fortunatamente è per il momento impossibile immaginare un programma in grado di creare grandi sinfonie o meravigliosi quartetti. Solo l'ingegno e la sensibilità dell'uomo possono fare ciò!

Le possibilità dell' informatica musicale

Lo scopo di questa tesi e dei molti lavori che si sono orientati in questa direzione è quello di mostrare quanto possano essere formalizzate le regole che sovrintendono alla pratica compositiva : l'atto creativo che il compositore compie non è quindi solo frutto di una indefinita "ispirazione" ma è l'applicazione esatta di regole rigorose e razionali (che però possono essere

violate per fini artistici). Ma questo non è che un esempio di come teorie formali ed approcci informatici possano essere applicati in ambito musicale. Infatti, tornando al modello di comunicazione presentato sopra, l'informatica musicale ha trattato nell'ultimo trentennio aspetti di varia natura e situati in vari punti del modello; in particolare

- *dal punto di vista del compositore :* la attività tipiche del compositore sono l'ideazione e la scrittura di un brano musicale. Vari lavori affrontano in maniera più o meno esauriente il primo tema : si hanno infatti sistemi per la creazione di una melodia (es. [L,S73] e [B,D,J99]), per la sua armonizzazione (es. [B,D,J99]), per la creazione di canoni di varia natura e di contrappunti (es. [B,D,J99]), per la generazione di variazioni su un dato tema (es. [Rom97]), per la creazione di musica tramite l'uso di sintetizzatori ed altre apparecchiature elettroniche (es. [Xen71]), etc… Il secondo aspetto si può addirittura considerare risolto, poiché esistono in commercio ormai software estremamente sofisticati per la scrittura di un brano musicale (il più sofisticato è *Finale* della *Coda Music Tecnology*).
- *dal punto di vista dell'esecutore :* un musicista deve anzitutto capire il brano che ha di fronte, per poi interpretarlo secondo la sua sensibilità, in base alla tradizione ed alla cultura ricevuta. Come abbiamo già detto esistono sistemi molto sofisticati e dettagliati che realizzano una precisa analisi armonica e formale di un brano dato; il primo aspetto è quindi ben impostato ed avviato ad una soluzione interessante (il lavoro più importante in merito è [L,J83] e tra breve ne parleremo). Il secondo problema è chiaramente molto più complesso da risolvere : l'esecutore, quando si appresta a studiare un brano, si trova di fronte ad una moltitudine di visioni tutte lecite e coerenti. E' il suo gusto, la sua personalità e la sua sensibilità che lo guidano nella scelta dell'interpretazione : tali aspetti sono

chiaramente estranei ad una macchina. Ma ciononostante anche in questo secondo campo sono stati fatti tentativi con risultati interessanti, ottenuti ovviamente tramite l'identificazione di situazioni standard che l'elaboratore si trova così in grado di gestire (es. [Wid94] e [Wid99]). Ovviamente queste soluzioni sono solo esperimenti che danno vita ad interpretazioni scontate e monotone : di nuovo, solo un essere umano è in grado di affrontare e risolvere il problema in maniera soddisfacente.

- *dal punto di vista dell'ascoltatore :* una persona che ascolta musica anzitutto la percepisce fisicamente, per poi effettuarne un'analisi inconscia e sommaria (potrà riconoscere i temi e la struttura del brano, comprendere l'elaborazione formale del materiale effettuata dal compositore, in alcuni casi avvertirà le modulazioni) ma soprattutto avrà una risposta emotiva da tale ascolto. Il primo aspetto è stato affrontato molto e in varie maniere : si può dire che ormai anche gli elaboratori sono dotati di orecchie in grado di ascoltare note e riconoscerle (es. [Moo77]); il secondo aspetto è simile (anche se molto più superficiale) alla prima fase analizzata per l'esecutore; la terza fase è per il momento irrealizzabile sulle macchine a nostra disposizione (la risposta emotiva di una musica è un evento assolutamente imprevedibile né formalizzabile : potrà quindi essere trattato quando avremo macchine dotate di sensibilità e personalità).

Vogliamo ora riportare alcuni importanti risultati precedenti il presente lavoro che però si inquadrano in un'ottica simile (cioè quella generativa). Prima di entrare nel dettaglio, è bene chiarire cosa si intende per "approccio generativo". Il significato più immediato (suggerito anche dalla parola stessa) è quello di un meccanismo che crea dal nulla un oggetto (nel nostro caso sarà un

brano musicale). Sicuramente questo significato è corretto ma è riduttivo; ciò viene rilevato anche in [L,J83], dove si muovono tre obiezioni a tale proposito :

- l'aspetto più interessante di una grammatica è il descrivere tramite un insieme finito di regole l'insieme (solitamente infinito) delle stringhe appartenenti ad un certo linguaggio (e non generarle tutte)
- data una stringa appartenente al linguaggio generato dalla grammatica, tramite le regole è possibile descriverne in maniera semplice e compatta la struttura presente ai vari livelli
- nell'ambito musicale (ancor più che in quello dei linguaggi naturali) tutta la ricerca è finalizzata alla comprensione dei meccanismi che governano un fenomeno psicologico complesso, la conoscenza

Chiarito questo punto, una teoria generativa può occuparsi dell'ideazione di un brano così come della sua analisi; risulta invece molto meno adatta alla gestione dell'interpretazione e del riconoscimento delle note (compiti meglio realizzati con approcci di intelligenza artificiale, ad esempio).

Il primo è il testo universalmente riconosciuto come fondamentale per ogni approccio generativo alle problematiche musicali : è il testo [L,J83] . Esso parte dai concetti fondamentali dell'analisi musicale proposta da Heinrich Schenker secondo cui un qualsiasi ascoltatore organizza in maniera inconscia i suoni percepiti in una struttura gerarchica coerente, ordinando quindi i vari fenomeni musicali in base ad un grado di importanza. La teoria così prodotta era però estremamente incompleta dal punto di vista formale : essa fu elaborata agli inizi del secolo quando ancora la linguistica non era sviluppata. L'idea alla base era però quella giusta : dare un insieme limitato di principi da cui derivare ricorsivamente un insieme (potenzialmente infinito) di brani. Ciò che mancava era un'adeguata formalizzazione della sua teoria.

Un importante tentativo di formalizzazione è quello di [Nar77] con la *teoria delle implicazioni – realizzazioni* , ma soprattutto [L,J83] . Anche questa opera trae origine dall'adattamento di lavori linguistici alla musica, ma evidenzia una fondamentale differenza : nelle grammatiche per un linguaggio naturale sono presenti categorie (nomi, verbi, …) e quindi le riscritture rappresentano relazioni di tipo *" Is_a "* , mentre in musica le categorie non sono presenti e le riscritture rappresentano relazioni di tipo *"elaborazione_di"*[1] . Il formalismo usato è quello degli alberi in cui i nodi rappresentano eventi; ogni livello rappresenta fenomeni ad un certo grado della gerarchia musicale; la radice di un sottoalbero rappresenta l'evento più importante tra tutti gli eventi associati ai nodi del sottoalbero; una ramificazione sinistra (o destra) rappresenta l'elaborazione dell'evento precedente (o successivo). Con questo formalismo è rappresentato tutto il materiale teorico presentato nel libro la cui struttura è la seguente :

– anzitutto si divide il brano musicale in *gruppi* così come un ascoltatore lo percepisce in base a somiglianze melodiche, ritmiche, vicinanza di eventi, pause, … Il brano viene così organizzato

[1] In una grammatica per un linguaggio naturale la produzione

$$\text{Frase} \rightarrow \text{Gruppo\_Nominale}\ \ \text{Gruppo\_Verbale}$$

vuol dire che una frase ***è*** la concatenazione di un gruppo nominale e di uno verbale.
In musica, invece, la produzione armonica

$$\text{I V I} \rightarrow \text{I } V_4^6\ V_7\ \text{I}$$

(che sostituisce una cadenza perfetta con una cadenza composta) non afferma l'uguaglianza dei due fenomeni ma afferma solo che si può compiere una sostituzione dell'uno con l'altro, percependo la somiglianza dei due. L'esempio musicale appena riportato è anche importante poiché un lavoro precedente svolto in questo dipartimento ([Rom97]) era basato interamente su riscritture di questo tipo.

gerarchicamente in motivi, sottofrasi, frasi, periodi, gruppi tematici e sezioni.

- indipendentemente associa una *struttura metrica* al brano, esaminando accenti forti e deboli, alternanza regolare di questi, metro del brano, … L'organizzazione così ottenuta rispecchia la diversa importanza dei vari accenti ai vari livelli; il livello fondamentale è quello del cosiddetto *tactus* in cui l'andamento è moderato e l'alternanza dei battiti è estremamente regolare (al contrario dei livelli inferiori e superiori in cui possono esserci molte più irregolarità).
- segmenta il brano in *time – spans* (gruppi di note adiacenti riconoscibili come una entità). A livello più basso tale fenomeno è guidato dalla struttura metrica che, salendo di livello, perde sempre più importanza fino a scomparire ai livelli più alti, in cui entra in gioco solo il raggruppamento del brano.
- *time – span reduction* : è il livello in cui i vari time–spans prodotti sono organizzati gerarchicamente, identificando all'interno di ognuno di essi un evento più importante strutturalmente (*head*) che lo rappresenterà al livello superiore. La head può essere un singolo evento, la fusione di più eventi adiacenti, la combinazione di parti prese da eventi adiacenti o insiemi di più eventi (ad esempio le cadenze). Lo scopo fondamentale è di eliminare ad ogni livello i fenomeni meno significativi così da produrre un'ossatura ritmica, armonica e melodica essenziale del brano. E' quindi necessaria una metrica per identificare l'evento più importante [2] .

[2] Per fare ciò basta decidere ad ogni livello un evento fondamentale ed una misura della distanza da esso (ad esempio la tonica e la scala maggiore o minore, l'accordo di tonica ed il circolo delle quinte, …)

– *prolongational reduction* : è il momento finale dell'analisi in cui si passa a descrivere il "respiro" della musica (cioè l'alternanza tra tensione e rilassamento). Alla creazione della struttura di questo livello concorrono tutte le analisi fatte nei passi precedenti, essendo la creazione dell'albero prolongazionale guidato dalla time–span reduction . Da essi si ricavano le informazioni necessarie per identificare somiglianze, contrasti, attesa, … presenti tra i vari segmenti del brano.

  A questo livello la costruzione dell'albero avviene in modalità top – down (mentre al passo precedente era fatto con procedimenti bottom – up) : questo viene fatto poiché l'evento strutturalmente più importante è solitamente anche quello in cui il brano raggiunge il suo culmine di tensione. Identificato quindi come nucleo fondamentale la sequenza *< inizio_brano , culmine , fine_brano >* , si cerca poi di ricondurre ad esso tutti gli altri eventi.

Tutti questi fenomeni sono organizzati in una gerarchia stretta e descritti tramite alberi. Ciò esprime la cosiddetta ipotesi di riducibilità forte : *"tutti gli eventi di un brano possono essere strutturati in un'unica gerarchia tramite la relazione di subordinazione"* . La relazione di subordinazione è transitiva e pertanto la rappresentazione tramite alberi è ben adeguata al problema.

Tutto il lavoro è volto a raggruppare un insieme coerente e completo di regole e a motivarne la scelta e l'uso; il risultato vede 56 regole, di cui alcune preferenziali, per realizzare le gerarchie descritte sopra. E' importante però notare tre caratteristiche del lavoro :

a) *l'approccio seguito è fortemente analitico* ; non c'è nessun cenno a come tutto il lavoro possa essere usato a scopi generativi propriamente detti.
b) volendo dar vita ad un sistema completo in grado di gestire ogni situazione musicale, *il risultato è molto pesante e complesso* .
c) *non è detto se e come il tutto possa essere implementato in maniera efficiente* . C'è solo un rapido cenno in conclusione al fatto che molte delle regole create richiedono fortemente il ricorso al contesto di applicazione : c'è quindi da pensare che una realizzazione pratica del sistema sia poco pensabile perché genererebbe una grammatica contestuale e quindi poco gestibile.

In conclusione analizzeremo come invece il lavoro presentato in questa tesi non soffra di nessuno dei problemi sopra descritti e qual è stato il prezzo da pagare per ottenere questo risultato.

Passiamo ora ad analizzare un altro risultato che più si avvicina al lavoro di questa tesi : [B,D,J99] . In esso lo scopo è prettamente generativo ed ha come risultato un programma implementato : lo scopo è quello di produrre arie da camera nello stile del compositore barocco Giovanni Legrenzi. Quindi, rispetto al lavoro precedentemente esposto (a cui comunque deve molto), supera due limiti importanti : l'approccio esclusivamente analitico e l'assenza di implementazione efficiente. Anche in questo caso esponiamo in estrema sintesi il lavoro per poter avere strumenti, in fase di conclusione, per un interessante confronto.

Anzitutto il lavoro si struttura in tre parti : una introduzione sul concetto di grammatica per la musica, una seconda dedicata alla descrizione del sistema formale ed implementato per le arie di Legrenzi ed una ultima (a carattere

quasi esclusivamente musicologico) in cui si esamina l'evoluzione della melodia nel corso della tradizione culturale occidentale, allo scopo di meglio inquadrare e motivare le scelte fatte nei passi precedenti. La parte per noi più interessante è ovviamente la seconda anche se vale la pena spendere alcune parole sulla prima.

Gli autori rilevando anzitutto (concordemente con [L,J83]) che anche in musica sia presente in maniera molto forte il concetto di gerarchia : ciò è da loro motivato dicendo che l'apprendimento e la fruizione di un qualsiasi fenomeno risultano più semplici ed efficaci se strutturate in maniera ferrea. Lo scopo di una grammatica è quindi quello di riassumere i processi logici che sovrintendono a questa gerarchia : esse devono pertanto codificare le convenzioni acquisite nel tempo, ma essere pronte a registrare novità (pena la stasi e la conseguente morte del fenomeno analizzato, essendo la musica una realtà in continuo divenire). Alcuni aspetti e regole sono infatti immutabili nel tempo e riguardano fattori sempre presenti in qualunque periodo storico : ciò si ottiene con regole per la chiarezza dell'espressione, l'immediatezza di espressione, la facilità di memorizzazione, … . Ciononostante, esistono sempre dei parametri variabili in maniera casuale [3] : infatti, al momento in cui si presentano più scelte, le modalità di selezione di una strada rispetto ad un'altra risultano di notevole interesse. Spesso, infatti, è proprio la distribuzione di probabilità che governa questi fenomeni a caratterizzare lo stile di un autore o di un'epoca.

Essendo l'approccio seguito più formale del testo precedente, gli autori si soffermano a lungo sui meccanismi formali delle grammatiche usate. Anzitutto affermano che compositore ed ascoltatore usano inconsciamente la

[3] In realtà la scelta non è fatta casualmente da un compositore, ma è guidata da motivazioni di carattere espressivo che però un sistema formale per il momento non può tenere in considerazione.

stessa grammatica, applicandola però in maniera diversa : l'approccio del primo è di tipo top – down (cioè si parte da una cellula fondamentale da specializzare nel corso della composizione), mentre quello del secondo è di tipo bottom – up (cioè parte dal fenomeno realizzato e ne ricostruisce man mano la struttura formale). E' in questo contesto che iniziano a discutere di una importante differenza, poco evidenziata invece in [L,J83] : una cosa è la grammatica (che esprime le regole da rispettare), un'altra cosa è invece la sua modalità d'applicazione (che esprime come applicare le regole presentate). Rileva inoltre come molto spesso si trascuri una descrizione chiara ed esplicita della forma che le regole assumono; è strano però notare che tale leggerezza è commessa anche nel corso della loro opera.

Essendo l'opera stata condotta comunque in un ambito musicologico, si porta un interessante commento sulle possibili relazioni tra analisi musicale tradizionale e apporti dovuti alle nuove frontiere dell'informatica musicale. L'analisi è il supporto fondamentale alla validazione delle regole prodotte; viceversa la grammatica è un grande supporto all'analisi a tutti i livelli formali presenti nel brano in questione. Inoltre una implementazione del sistema prodotto è una verifica immediata della completezza e non contraddittorietà delle regole fornite; si impone quindi la necessità di evidenziare ogni passaggio intermedio, pena un comportamento impreciso (e quindi immediatamente avvertibile) del sistema.

Infine presentano i due possibili approcci ad un problema generativo : il primo consiste nel produrre la musica seguendo una direzione specifica (tipicamente da sinistra a destra, cioè dal suo inizio alla sua fine), il secondo seguendo una tecnica espansiva (cioè partire da un germe iniziale ed evolverlo strutturalmente ai vari livelli, dai più profondi ai più superficiali). L'approccio

seguito dagli autori è di questo secondo tipo, così come molti altri lavori fatti[4] ; il procedimento che invece sarà esposto in questa tesi è invece prevalentemente del primo tipo e sarà quindi interessante in conclusione motivare questa differente scelta.

Andiamo ora a presentare in sintesi il lavoro fatto dagli autori : l'input era un testo poetico da musicare secondo le regole fornite (estratte da un campione di arie di Legrenzi). Come in [L,J83] il lavoro procede su più livelli ed ogni livello ha una sua propria grammatica. In ordine si hanno:

- *elaborazione del testo poetico* : segmentazione dovuta ad accenti, versi, ripetizioni, … che, tramite un parsing bottom – up , struttura il testo a vari livelli (singole parole, frasi, periodi, … , fino ad arrivare alla forma globale del brano). L'identificazione di tale struttura è il passo fondamentale poiché la musica sarà creata in modo da rispettarla ed enfatizzarla.
- *creazione della macroforma* : identifica anzitutto le varie frasi musicali inserendo elementi strutturali che le demarchino (pause, note lunghe, cadenze, …) in base alla segmentazione del testo. Decide poi la successione delle tonalità, con l'assunzione che ogni modulazione avvenga in concomitanza del cambio di frase. Infine, per ogni conclusione, decide il tipo di cadenza (più o meno conclusiva, più o meno sospensiva, …) e la loro successione nell'intero brano in base al testo poetico.

[4] Tra gli altri ci teniamo a segnalare il filone di lavori svolti nel nostro dipartimento da tesisti precedenti che sfruttano questo approccio : [Cer96] , [Rom97] e [Ach98] .

- *costruzione della melodia* : segue un approccio top – down , partendo da una scala nucleare (derivata a sua volta da un intervallo nucleare) ed evolvendola tramite varie strutture di due livelli : uno profondo (ripetizioni, note di volta, …) ed uno superficiale (sincopi, appoggiature, anticipazioni, fioriture, …). La costruzione melodica tiene fortemente in conto il testo poetico : infatti ripetizioni del testo sono realizzate solitamente tramite ripetizioni della frase melodica e la successione di accenti forti e deboli del testo é tipicamente rispettata nell'organizzazione metrica della melodia. In particolare, si parte dalla struttura metrica del testo, si adatta ad essa la scala nucleare e si apportano successive modifiche di quest'ultima allo scopo di verificare la corrispondenza di almeno una nota per ogni sillaba del testo.
- *creazione dell'armonia* : la successione degli accordi realizza l'alternanza di tensione e rilassamento della musica. La struttura armonica globale e le cadenze sono state già decise nella macrostruttura : ciò che bisogna fare è adeguare le armonie stabilite agli accenti metrici e musicali (tipicamente i cambi importanti avvengono su tempi forti) ed aggiungere poi le armonie secondarie (sui tempi deboli).
- *determinazione della linea del basso* : nel modello legrenziano, il basso è una via di mezzo tra una linea melodica autonoma ed un semplice basso continuo : pertanto la sua produzione sarà simile a quella del canto (scala nucleare e sua evoluzione) ma con molti più vincoli e quindi minore elaborazione. Essendo inoltre già stata determinata la successione delle armonie, la scala nucleare scelta dovrà combaciare con tali scelte (nel senso che in corrispondenza di un accordo forte ci dovrà essere una nota della scala); inoltre le frasi

del basso seguono piuttosto fedelmente le frasi del canto e quindi ad esse dovranno essere adattate. Pertanto l'evoluzione del basso sarà piuttosto vincolata e si limiterà ad introdurre qualche rara fioritura e un certo numero di armonie e note di passaggio; nel fare ciò si dovranno comunque rispettare le classiche regole per l'esecuzione di due voci assieme (il cosiddetto contrappunto a due che tra breve affronteremo nella nostra tesi).

Vogliamo infine riassumere alcune caratteristiche di questo lavoro che ci torneranno utili in fase di paragone col lavoro da noi proposto :

a) *le regole sono espresse a parole*, senza nessuna formalizzazione matematica. Come nel testo [L,J83] ciò alimenta il dubbio su di una efficiente realizzazione pratica di esse.
b) nella loro implementazione è stato dichiarato un *frequente ricorso a meccanismi di backtrack* anche tra livelli differenti. Questo è un fatto per noi confortante visto che anche nel nostro sistema questa eventualità può occorrere (con frequenza minore come in dettaglio motiveremo in conclusione)
c) tutto il loro sistema (come poi il nostro) lavora nella *stessa tonalità* : è questo un ulteriore incoraggiamento a non preoccuparci di tale limitazione (visto che è abbastanza diffusa)
d) le composizioni realizzate prevedono l' *uso di due sole voci* (noi ne useremo quattro).

La struttura della tesi

La tesi è strutturata in due parti principali (oltre alla presente introduzione ed una breve conclusione) : la prima si occupa di aspetti puramente teorici e formali fornendo degli strumenti che ben si adattano a modellare l'aspetto musicale trattato, la seconda descrive il problema affrontato e la maniera in cui è stato risolto tramite i meccanismi generativi presentati nella sezione precedente.

In particolare l'approccio seguito è puramente generativo : per questo vengono richiamati all'inizio della prima parte elementi di teoria dei linguaggi formali. Si espone poi un modello di grammatica detto OLIN (abbreviazione di "Only the Last–one Is Non–terminal") che sintatticamente è di tipo contestuale, il cui potere generativo è però molto più ridotto, essendo il linguaggio generato di tipo regolare; l'utilità di tali grammatiche sta nel fatto che tramite esse è molto semplice ed intuitivo formalizzare e risolvere il problema della sincronizzazione di una data sequenza di eventi in base ad un dato insieme di regole di tipo markoviano (tali che, cioè, la scelta al passo i dipende solo dalle scelte fatte nei k passi precedenti ad i , con k costante). Si passa poi a discutere la complessità del problema ed una implementazione efficiente. Questo meccanismo è poi generalizzato per trovare una sincronizzazione con due o più successioni di eventi date. Infine si confronterà tale procedimento di sincronizzazione con un altro procedimento abbastanza di moda come gli Sticker Systems del DNA Computing.

La seconda parte invece utilizza questi strumenti per dare vita all'esposizione di una fuga (la cui struttura è descritta all'inizio della sezione). In estrema sintesi, dato un tema musicale (il *soggetto*), si crea un *controsoggetto* lecito mediante una grammatica di sincronizzazione; poi, con meccanismi probabilistici, si decidono le entrate delle varie voci, caratterizzate

dall'esecuzione del soggetto e controsoggetto in sequenza; infine si colmano i vuoti lasciati inserendo le *parti libere* generate tramite una grammatica di sincronizzazione generalizzata.

In conclusione si analizzano i risultati ottenuti con alcuni altri lavori effettuati nel campo e si espongono eventuali sviluppi futuri, nel tentativo di eliminare i vincoli che un primo approccio al problema necessariamente doveva porre.

In appendice sono riportate due tavole che riassumono le caratteristiche dei più importanti soggetti e controsoggetti di J.S.Bach ed il risultato di una applicazione del sistema implementato, usando un soggetto di W.A.Mozart.

PARTE I

TEORIA DEI LINGUAGGI

FORMALI

Capitolo 1 : Richiami di teoria dei linguaggi formali

Prima di esporre il lavoro fatto nell'ambito dei linguaggi formali, premettiamo un breve richiamo a risultati noti e presenti su tutti i testi di tale disciplina, nonché alcuni elementi di DNA Computing che ci torneranno utili per paragonare i nostri risultati con quelli di tale nuovo campo di indagine.

1.1 La gerarchia di Chomsky

Def. : *una grammatica è una quadrupla* (N, T, S, P) *dove*

- N *è l'insieme dei simboli non terminali*
- T *è l'insieme dei simboli terminali*
- S *è il simbolo iniziale*
- P *è l'insieme delle produzioni*

con le seguenti caratteristiche :

- $N \cap T = \varnothing$
- $S \in N$
- *una produzione è una coppia* (α, β) *(dove* $\alpha \in (T \cup N)^+$ *e* $\beta \in (T \cup N)^*$*. Spesso tale produzione si indicherà con* $\alpha \to \beta$*, nel senso che, applicandola, l'occorrenza della stringa* α *sarà rimpiazzata da una occorrenza della stringa* β

Def. : α *è una forma sentenziale per* G *se* $\alpha \in (T \cup N)^*$ *e* $S \to_G{}^* \alpha$

N.B. : è ragionevole considerare come insieme di produzioni di G l'insieme di tutte e sole le regole che possono essere applicate ad una forma sentenziale di G (cioè se $\exists$ p $\in$ P t.c. $\forall$ *w* forma sentenziale $S \rightarrow_G^* w$ non usando mai p , allora di escluderà p da P).

Def. : *una grammatica è detta*

. *regolare* se $\forall \alpha \rightarrow \beta \in P$ si ha che $\alpha \in N$ e $\beta \in \{\lambda\} \cup T \cup T \cdot N$

. *non contestuale* se $\forall \alpha \rightarrow \beta \in P$ si ha che $\alpha \in N$

. *contestuale* se $\forall \alpha \rightarrow \beta \in P$ si ha che $\alpha = \gamma A \delta$ e $\beta = \gamma \alpha \delta$, dove $\gamma, \delta \in (T \cup N)^*$ e $A \in N$

Def.: *una grammatica è lineare destra se* $\forall \alpha \rightarrow \beta \in P$ *si ha che* $\alpha \in N$ *e* $\beta \in T^* \cdot (N \cup \{ \lambda \})$

Def. : *il linguaggio generato da una grammatica G è*

$$L(G) = \{ \alpha \in T^* : S \rightarrow_G^* \alpha \}$$

N.B.: con il simbolo '$\rightarrow$' indichiamo una produzione, mentre con '$\rightarrow_G$' un passo di derivazione in G

Def. : $REG = \{ L \subseteq T^* : \exists G$ *grammatica regolare t.c.* $L = L(G) \}$

$CF = \{ L \subseteq T^* : \exists G$ *grammatica non contestuale t.c.* $L = L(G) \}$

$CS = \{ L \subseteq T^* : \exists G$ *grammatica contestuale t.c.* $L = L(G) \}$

Prop. : *G è lineare destra* $\Rightarrow$ *esiste G' regolare tale che* $L(G) = L(G')$

Prop. (Chomsky) : $REG \subset CF \subset CS \subset 0K$

1.2 Elementi di teoria di DNA Computing

Una parte dell'informatica teorica che nell'ultimo decennio si è molto sviluppata è quella delle macchine genetiche, tra cui il cosiddetto *DNA Computing* . Quest'ultimo è un meccanismo per la gestione di coppie di stringhe sovrapposte seguendo opportune regole; è quindi un meccanismo generativo il cui scopo è simile a quello delle grammatiche che tra breve introdurremo. E' quindi bene richiamarne i concetti fondamentali per poi effettuare un confronto tra i due metodi.

Sia dato un alfabeto V ed una relazione simmetrica [5] $\rho \subseteq V \times V$, detta complementarità. Considerato ora V^* , il monoide libero su V , si consideri il monoide $V^* \times V^*$: esso sarà indicato più significativamente con $\begin{pmatrix} V^* \\ V^* \end{pmatrix}$, da cui $(x, y) \in V^* \times V^*$ si scriverà $\begin{pmatrix} x \\ y \end{pmatrix}$ e quindi, con questa notazione, la concatenazione di (x, y) e (w, z) sarà indicata $\begin{pmatrix} x\,w \\ y\,z \end{pmatrix}$.

Il motivo di questa notazione risiede nel fatto che nel DNA le due stringhe formanti l'elica molecolare sono sovrapposte e riflettono la complementarità di Watson – Crick : ciò è contenuto nella seguente definizione

[5] La simmetria di ρ non è necessaria nella trattazione seguente ma viene comunque riportata per fedeltà al modello biologico rappresentato : nel DNA la complementarità tra le basi (detta di Watson – Crick) è simmetrica

Def. : $\left[\begin{matrix} V \\ \\ V \end{matrix}\right]_\rho = \{ \begin{pmatrix} a \\ \\ b \end{pmatrix} : a, b \in V , (a, b) \in \rho \}$

$WK_\rho(V) = \left[\begin{matrix} V \\ \\ V \end{matrix}\right]_\rho^*$ *(dominio di Watson – Crick)*

Dell'insieme $WK_\rho(V)$ vale la pena evidenziare due ovvie ma importanti proprietà.

Prop. : *Data* $\begin{pmatrix} x \\ \\ y \end{pmatrix} \in WK_\rho(V)$, *allora*

- $|x| = |y|$
- $\forall i = 1, \ldots, |x|$ *si ha che* $(x_i, y_i) \in \rho$

Si potranno in seguito usare anche "molecole incomplete" definite come segue

Def. : $L_\rho(V) = \left(\begin{pmatrix} \lambda \\ \\ V^* \end{pmatrix} \cup \begin{pmatrix} V^* \\ \\ \lambda \end{pmatrix} \right) \left[\begin{matrix} V \\ \\ V \end{matrix}\right]_\rho^*$

$R_\rho(V) = \left[\begin{matrix} V \\ \\ V \end{matrix}\right]_\rho^* \left(\begin{pmatrix} \lambda \\ \\ V^* \end{pmatrix} \cup \begin{pmatrix} V^* \\ \\ \lambda \end{pmatrix} \right)$

$LR_\rho(V) = \left(\begin{pmatrix} \lambda \\ \\ V^* \end{pmatrix} \cup \begin{pmatrix} V^* \\ \\ \lambda \end{pmatrix} \right) \left[\begin{matrix} V \\ \\ V \end{matrix}\right]_\rho^+ \left(\begin{pmatrix} \lambda \\ \\ V^* \end{pmatrix} \cup \begin{pmatrix} V^* \\ \\ \lambda \end{pmatrix} \right)$

$W_\rho(V) = L_\rho(V) \cup R_\rho(V) \cup LR_\rho(V)$

Gli elementi di $W_\rho(V)$ sono detti *domino* poiché, come i tasselli del domino, giustapponendoli opportunamente (cioè tramite la complementarità Watson – Crick) si possono comporre molecole tramite l'operazione di sticking che andiamo ora a definire.

Def. : *sia* $x = x_1 x_2 x_3 \in W_\rho(V)$ *tale che :* $. x_2 \in WK_\rho(V) - \{ \begin{bmatrix} \lambda \\ \\ \lambda \end{bmatrix} \}$

$$. \; x_1 , x_3 \in \begin{pmatrix} V^* \\ \\ \lambda \end{pmatrix} \cup \begin{pmatrix} \lambda \\ \\ V^* \end{pmatrix}$$

data $y \in W_\rho(V)$ *, si definisce lo* *sticking* *di* x *e* y ($\mu(x, y)$) *se*

1) $x_3 = \begin{pmatrix} u \\ \\ \lambda \end{pmatrix}$, $y = \begin{pmatrix} \lambda \\ \\ v \end{pmatrix}$ y' *t.c.* $\begin{bmatrix} u \\ \\ v \end{bmatrix} \in WK_\rho(V)$ *e* $y' \in R_\rho(V)$

Allora $\mu(x, y) = x_1 \; x_2 \begin{bmatrix} u \\ \\ v \end{bmatrix} y'$

2) $x_3 = \begin{pmatrix} \lambda \\ \\ v \end{pmatrix}$, $y = \begin{pmatrix} u \\ \\ \lambda \end{pmatrix}$ y' *t.c.* $\begin{bmatrix} u \\ \\ v \end{bmatrix} \in WK_\rho(V)$ *e* $y' \in R_\rho(V)$

Allora $\mu(x, y) = x_1 \; x_2 \begin{bmatrix} u \\ \\ v \end{bmatrix} y'$

3) $x_3 = \begin{pmatrix} u_1 \\ \\ \lambda \end{pmatrix}$, $y = \begin{pmatrix} u_2 \\ \\ \lambda \end{pmatrix}$ con $u_1 , u_2 \in V^*$.

Allora $\mu(x , y) = x_1 \; x_2 \begin{pmatrix} u_1 \, u_2 \\ \\ \lambda \end{pmatrix}$

4) $x_3 = \begin{pmatrix} \lambda \\ \\ v_1 \end{pmatrix}$, $y = \begin{pmatrix} \lambda \\ \\ v_2 \end{pmatrix}$ con $v_1 , v_2 \in V^*$.

Allora $\mu(x , y) = x_1 \; x_2 \begin{pmatrix} \lambda \\ \\ v_1 \, v_2 \end{pmatrix}$

5) $x_3 = \begin{pmatrix} u_1 \, u_2 \\ \\ \lambda \end{pmatrix}$, $y = \begin{pmatrix} \lambda \\ \\ v \end{pmatrix}$ t.c. $\begin{bmatrix} u_1 \\ \\ v \end{bmatrix} \in WK_\rho(V)$ e $u_2 \in V^*$

Allora $\mu(x , y) = x_1 \; x_2 \begin{bmatrix} u_1 \\ \\ v \end{bmatrix} \begin{pmatrix} u_2 \\ \\ \lambda \end{pmatrix}$

6) $x_3 = \begin{pmatrix} \lambda \\ \\ v_1 \, v_2 \end{pmatrix}$, $y = \begin{pmatrix} u \\ \\ \lambda \end{pmatrix}$ t.c. $\begin{bmatrix} u \\ \\ v_1 \end{bmatrix} \in WK_\rho(V)$ e $v_2 \in V^*$

Allora $\mu(x , y) = x_1 \; x_2 \begin{bmatrix} u \\ \\ v_1 \end{bmatrix} \begin{pmatrix} \lambda \\ \\ v_2 \end{pmatrix}$

7) $x_3 = \begin{pmatrix} u \\ \\ \lambda \end{pmatrix}$, $y = \begin{pmatrix} \lambda \\ \\ v_1\ v_2 \end{pmatrix}$ *t.c.* $\begin{bmatrix} u \\ \\ v_1 \end{bmatrix} \in WK_\rho(V)$ *e* $v_2 \in V^*$

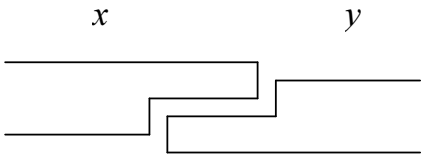
Allora $\mu(x, y) = x_1\ x_2 \begin{bmatrix} u \\ \\ v_1 \end{bmatrix} \begin{pmatrix} \lambda \\ \\ v_2 \end{pmatrix}$

8) $x_3 = \begin{pmatrix} \lambda \\ \\ v \end{pmatrix}$, $y = \begin{pmatrix} u_1\ u_2 \\ \\ \lambda \end{pmatrix}$ *t.c.* $\begin{bmatrix} u_1 \\ \\ v \end{bmatrix} \in WK_\rho(V)$ *e* $u_2 \in V^*$

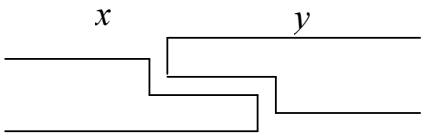
Allora $\mu(x, y) = x_1\ x_2 \begin{bmatrix} u_1 \\ \\ v \end{bmatrix} \begin{pmatrix} u_2 \\ \\ \lambda \end{pmatrix}$

Graficamente i casi appena descritti sono così rappresentabili :

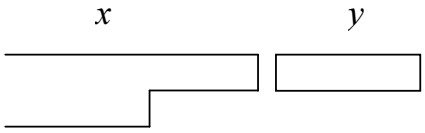
1)



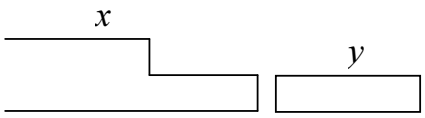
2)



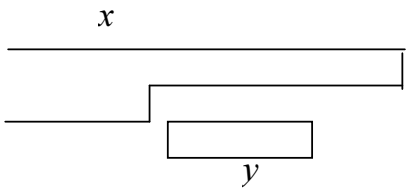
3)



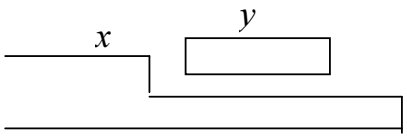
4)



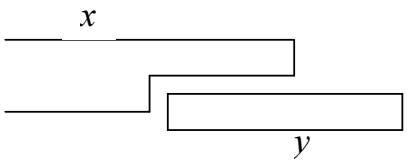
5)



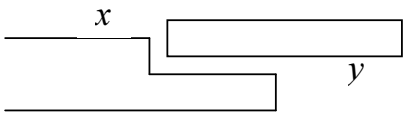
6)



7)



8)



Tramite l'operazione di sticking appena illustrata, si possono creare dei sistemi generativi chiamati sticker systems che andiamo ora a definire

Def. : *uno* *sticker system* *è una quadrupla* $\gamma = (V , \rho , A , D)$ *dove :*

- V *è un alfabeto*
- $\rho \subseteq V \times V$ *una relazione simmetrica*
- $A \subseteq LR_{\rho}(V)$ *finito (gli assiomi)*
- $D \subseteq W_{\rho}(V) \times W_{\rho}(V)$ *finito (i domino di produzione)*

Come ogni meccanismo generativo, andiamo ora ad introdurre i concetti di derivazione e di linguaggio generato.

Def. : $\forall x, y \in LR_\rho(V)$ $x \rightarrow_\gamma y \Leftrightarrow \exists (u, v) \in D$ $y = \mu(u, \mu(x, v))$

N.B. : la sequenzializzazione della derivazione è detta computazione ed è completa se l'ultima molecola derivata è completa

Def. : $LM(\gamma) = \{ y \in WK_\rho(V) : \exists x \in A \text{ t.c. } x \rightarrow_\gamma^* y \}$

(il linguaggio delle molecole)

$$L(\gamma) = \{ u \in V^* : \exists v \in V^* \text{ t.c. } \begin{bmatrix} u \\ v \end{bmatrix} \in LM(\gamma) \}$$

(il linguaggio delle stringhe) [6]

Agli sticker systems così presentati, spesso se ne affiancano alcune varianti allo scopo di meglio caratterizzarne il potere generativo. Di seguito riportiamo solo le definizioni e gli enunciati di proposizioni e teoremi che ci serviranno in seguito; per una trattazione completa e per le dimostrazioni rimandiamo sempre a [P,R,S99].

Def. : *uno sticker system è detto*

– *regolare sse* $\forall (u, v) \in D$ *si ha che* $u = \begin{bmatrix} \lambda \\ \lambda \end{bmatrix}$

[6] Il linguaggio LM è detto *di molecole* perché considera coppie di stringhe sovrapposte (elementi di WK_ρ) mentre L è detto di stringhe poiché considera solo la catena superiore

– <u>*semplice*</u> *sse* $\forall (u,v) \in D$ *si ha che* $u, v \in \begin{pmatrix} V^* \\ \\ \lambda \end{pmatrix}$ *o* $u, v \in \begin{pmatrix} \lambda \\ \\ V^* \end{pmatrix}$

I linguaggi di stringhe generati da uno sticker system regolare , semplice e regolare e semplice sono indicati rispettivamente con $RSL(\gamma)$, $SSL(\gamma)$ *e* $SRSL(\gamma)$

Teor. : $SRSL(\gamma) \subset REG$

Capitolo 2 : Le grammatiche OLIN

2.1 Definizione e caratterizzazione

Def. : *una grammatica è detta OLIN (Only the Last–one Is Non–terminal) se*

$\forall x \to y \in P$ *si ha che* $x = \alpha A$ *e* $y = \alpha\beta$ *dove*

- $\alpha \in T^*$
- $A \in N$
- $\beta \in T^* \cdot (N \cup \{\lambda\})$

Per esempio la seguente produzione è OLIN

$$\alpha_1 \dots \alpha_k A \to \alpha_1 \dots \alpha_k \beta_1 \dots \beta_t B$$

mentre non lo sono

$$A \to \alpha_1 \dots \alpha_k B C$$

$$A \to \alpha_1 \dots \alpha_k D \beta_1 \dots \beta_t \quad (\text{con } t \geq 1)$$

La prima infatti ha due non terminali (B e C), mentre la seconda ha un non terminale (D) che non sta all'ultimo posto nella parte destra della produzione.

Teor.(caratterizzazione) : *G è OLIN sse ogni sua forma sentenziale w ha al più un non terminale che è l'ultimo simbolo di w*

Dim.

$\Rightarrow$ *(per induzione su n, il numero di produzioni applicate per arrivare a w)*

- Base (n = 0) : $w = S \Rightarrow$ banale

– Induzione (tesi vera per n , da dim. per n+1) : sia $S \rightarrow_G^n w' \rightarrow_G w$ la derivazione che ha portato a w . Per induzione w' ha un solo non terminale (se non ne avesse nessuno allora non si potrebbe più applicargli nessuna produzione) che è l'ultimo suo simbolo. L'ultima produzione, essendo G OLIN , sarà del tipo

$$t\ A \rightarrow t\ t'\ n$$

con $t , t' \in T^*$, $A \in N$ e $n \in N \cup \{\lambda\}$, e si avrà che

$$w' = x\ t\ A \qquad \text{e} \qquad w = x\ t\ t'\ n$$

con $x \in T^*$. Quindi anche w soddisfa la tesi

$\Leftarrow$ *(per assurdo)*

Diciamo che esista in P una produzione $x \rightarrow y$ che renda G non OLIN. Allora si deve verificare almeno una delle seguenti condizioni

(i) $x \in T^* \cdot N \cdot T^+$: in tal caso questa produzione non sarà mai applicabile in G poiché ogni sua forma sentenziale per ipotesi appartiene a T^* o a $T^* \cdot N$; quindi tale produzione non sarà considerata come una produzione di G

(ii) x ha più di un non terminale : come in (i) la produzione non sarà mai applicabile e quindi ignorata

(iii) $y \in T^* \cdot N \cdot T^+$: o tale regola non è mai applicabile e allora torniamo ai casi precedenti oppure tale regola sarà l'ultima ad essere applicata nel derivare una certa forma sentenziale w , cioè $S \rightarrow_G^* w' \rightarrow_G w$, con $w' = s\ x$ e $w = s\ y$. Ma allora w non termina con un non terminale, in contraddizione con l'ipotesi

(iv) y ha più di un non terminale : come in (iii) tale regola, se applicabile, porterebbe ad avere una forma sentenziale con più di un terminale, in contraddizione con l'ipotesi.

c.v.d.

2.2 Equivalenza con linguaggi regolari

Teor.1 : *G OLIN e non contestuale* $\Rightarrow \exists$ *G' regolare equivalente a G*

Dim.

Allora G è lineare destra e quindi esiste una grammatica regolare equivalente[7]

c.v.d.

Il teorema appena mostrato ci garantisce che le parti destre delle produzioni non creano problemi, poiché possono assumere la forma imposta dalle grammatiche regolari introducendo nuovi simboli non terminali. Quindi, nel mostrare l'equivalenza con i linguaggi regolari, trascureremo le parti destre, concentrandoci solo su quelle sinistre. Verrà dato un algoritmo che sostituisce le regole contestuali con nuove produzioni la cui parte sinistra è formata solo da un non terminale; il corretto funzionamento è garantito dalla seguente

Lemma : *Sia G una grammatica olin e sia*

$$\beta_0 \ (= S) \rightarrow_G \beta_1 \rightarrow_G \beta_2 \rightarrow_G \ldots \rightarrow_G \beta_n \ (= \beta) \rightarrow_G \beta_{n+1} \ (= \gamma)$$

una derivazione tale che, nel passare da β *a* γ*, usa la produzione* $x \rightarrow y$*. Allora* $\exists\ i < n$ *t.c.* $S \rightarrow_G^i \beta_i$ *e* $\beta_i \rightarrow_G \beta_{i+1}$ *con la produzione* $w \rightarrow z$ *t.c.* $z \rightarrow_G^* \alpha x$ *(con* $\alpha \in T^*$*).*

---

[7] Infatti presa la generica $x \rightarrow y \in P$ se $y \in \{\lambda\} \cup T \cup T \cdot N$ la produzione è regolare; altrimenti si ha che $y = t_1 \ldots t_k\ n$ con $k \geq 2$, $t_i \in T$ per $i = 1,\ldots,k$ e $n \in N \cup \{\lambda\}$. Allora sostituisco $x \rightarrow y$ con

$$\{ x \rightarrow t_1 A_1 ,\ A_1 \rightarrow t_2 A_2 , \ldots , A_{k-2} \rightarrow t_{k-1} A_{k-1} , A_{k-1} \rightarrow t_k\ n \}$$

dove $A_1 , A_2 , \ldots , A_{k-1}$ sono non terminali nuovi. La grammatica così ottenuta è chiaramente regolare e con una semplice doppia inclusione si mostra che il linguaggio generato coincide con L(G) .

Dim.

La dimostrazione è banale : infatti, poiché $\beta_n \rightarrow_G \beta_{n+1}$ con $x \rightarrow y$, si avrà che

$$\beta = s\,x \qquad \text{e} \qquad \gamma = s\,y$$

Quindi deve esistere $i < n$ (sicuramente almeno $i = 0$) tale che

$$\beta_i = \delta\, w_i \quad , \qquad \beta_{i+1} = \delta\, z_i \quad , \qquad w_i \rightarrow z_i \in P$$

$$\text{e} \qquad z_i \rightarrow_G^{n-i-1} \alpha\, x$$

dove $\alpha , \delta \in T^*$ e $\delta \cdot \alpha = s$ (nel caso di $i = 0$ si ha $\delta = \lambda$, $w_i = S$ e $z_i = \beta_1$) .

c.v.d.

La proposizione ci garantisce che, se ad un certo passo di una derivazione si può usare la produzione $x \rightarrow y$, allora ci deve essere una produzione $w \rightarrow z$ precedentemente usata che $z \rightarrow_G^* \alpha\, x$. Passiamo quindi a dare l'algoritmo che elimina tutte le produzioni con parte sinistra contestuale, sostituendole con produzioni non contestuali. In esso si supporrà che tutte le produzioni con parte sinistra uguale siano raggruppate (cioè se $x \rightarrow y_1$, … , $x \rightarrow y_m \in P$ allora le sostituisce con $x \rightarrow y_1 \mid \ldots \mid y_m$)

1) $\forall\ x \rightarrow y_1 \mid \ldots \mid y_m \in P$ t.c. $x \in T^+ \cdot N$
2) $\quad \forall\ w \rightarrow z \in P$
3) $\quad\quad$ se $\exists\, \alpha \in T^* : z \rightarrow_G^* \alpha\, x$
4) $\quad\quad$ allora $P = P \cup \{\, w \rightarrow \alpha\, y_1 \mid \ldots \mid \alpha\, y_m \,\}$
5) $\quad P = P - \{\, x \rightarrow y_1 \mid \ldots \mid y_m \,\}$

L'algoritmo considera tutte le parti sinistre di produzioni contestuali [riga 1]. Per la generica x , passa in rassegna P [riga 2], selezionando le regole dalla cui parte destra si può derivare una stringa con suffisso x [riga 3]; detta $w \to z$ una tale produzione, aggiunge m nuove produzioni che fanno derivare da w i terminali precedenti x nella stringa derivata da z (α) concatenati con tutto e solo ciò che si derivava da x (y_1 , … , y_m) [riga 4] ; dopo aver esaminato tutte le produzioni a disposizione, elimina $x \to y_1 \mid \ldots \mid y_m$ [riga 5] e itera finché P non ha più produzioni contestuali.

Teor.2 (correttezza) : *Sia G una grammatica OLIN contestuale e sia*

$$G' = (N , T , S , P')$$

la grammatica ottenuta applicando a G l'algoritmo ; allora G' è non contestuale , OLIN ed equivalente a G.

Dim.

a) Sia $x \to y$ una generica produzione di G; se è non contestuale, essa non verrà considerata nell'algoritmo; altrimenti, l'algoritmo dedicherà ad essa una sua iterazione al termine della quale la eliminerà, avendo aggiunto o nuove produzioni non contestuali (se $w \in$ N) o nuove produzioni contestuali ma con parte sinistra già presente in P (w) e non ancora considerata dall'algoritmo (da qui l'utilità di raggruppare le produzioni in base alla parte sinistra). Si procede così finché esistono parti sinistre contestuali; poiché esse sono di numero finito ed ogni iterazione ne elimina una senza introdurne nessun'altra nuova, l'algoritmo terminerà in un numero finito di passi (lineare nel numero di produzioni) creando una grammatica non contestuale.

b) G' è OLIN poiché :

- le produzioni di P lasciate in P' sono di una grammatica OLIN
- le nuove produzioni sono OLIN poiché

$$w \in T^* \cdot N \quad \text{e} \quad y_1 , \dots , y_m \in T^* \cdot (N \cup \{\lambda\})$$

essendo G OLIN , da cui $\alpha y_1 , \dots , \alpha y_m \in T^* \cdot (N \cup \{\lambda\})$ essendo $\alpha \in T^*$.

c) Mostreremo che $L(G) = L(G')$ per doppia inclusione

♦ $L(G) \subseteq L(G')$

Sia $w \in L(G)$. Allora

- $S = w_0 \rightarrow_G w_1 \rightarrow_G \dots \rightarrow_G w_k = w$ per $k \geq 0$
- $\forall\, i = 0 , \dots , k-1$ la produzione usata per $w_i \rightarrow_G w_{i+1}$ può essere :

– *non contestuale :* allora $w_i \rightarrow_{G'} w_{i+1}$ poiché le produzioni non contestuali non sono eliminate dall'algoritmo e sono quindi tutte presenti in P'.

– *contestuale :* sia $x \rightarrow y$ la produzione usata, cioè

$$w_i = \alpha x \quad \text{e} \quad w_{i+1} = \alpha y$$

Per il Lemma precedente si ha che $\exists\, j < i$ t.c.

$w_j = \beta \gamma$, $w_{j+1} = \beta \mu$, $\gamma \rightarrow \mu \in P$, $\mu \rightarrow_G^* \delta x$ e $\gamma \in N$ con $\beta \cdot \delta = \alpha$ (sicuramente almeno per $j = 0$, nel qual caso $\beta = \lambda$, $\gamma = S$ e $\mu = w_1$) . Ma quindi nell'iterazione dell'algoritmo dedicata a $x \rightarrow y$ si considererà anche $\gamma \rightarrow \mu$ e quindi in P' si avrà anche $\gamma \rightarrow \delta y$. Perciò

$$w_j = \beta \gamma \quad \rightarrow_{G'} \quad \beta \delta y \;=\; \alpha y \;=\; w_{i+1}$$

Abbiamo quindi mostrato che

$$\forall\, i = 0 , \ldots , k\text{-}1 \;\; \exists\, j \le i \;\text{ t.c. }\; w_j \rightarrow_{G'} w_{i+1}$$

Quindi possiamo concludere che

$$\exists\, k_1 < k \;\text{ t.c. }\; w_{k1} \rightarrow_{G'} w_k = w$$

$$\exists\, k_2 < k_1 \;\text{ t.c. }\; w_{k2} \rightarrow_{G'} w_{k1}$$

$$\ldots$$

$$\exists\, k_d < k_{d-1} \;\text{ t.c. }\; w_{kd} \rightarrow_{G'} w_{kd-1} \;\text{ AND }\; w_{kd} = w_0 = S$$

ovvero che $S \rightarrow_{G'}{}^* w$, cioè $w \in L(G')$.

- $L(G') \subseteq L(G)$

Sia $w \in L(G')$. Allora

- $S = w_0 \rightarrow_{G'} w_1 \rightarrow_{G'} \ldots \rightarrow_{G'} w_k = w$ per $k \ge 0$
- $\forall\, i = 0 , \ldots , k\text{-}1$ la produzione usata per $w_i \rightarrow_{G'} w_{i+1}$ può essere :

– *inclusa in* $P' \cap P$ *:* allora banalmente $w_i \rightarrow_G w_{i+1}$

– *inclusa in* $P' - P$ *:* allora sarà stata inclusa dall'algoritmo e sarà della forma

$$A \rightarrow \alpha\, y \qquad \text{con } A \in N \;,\; \alpha \in T^* \;\text{ e }\; y \in T^* \cdot (N \cup \{\lambda\})$$

avendo quindi che

$$w_i = s\, A \qquad \text{e} \qquad w_{i+1} = s\, \alpha\, y \qquad \text{con } s \in T^*$$

Ma se tale produzione è in P' – P , ciò vuol dire che è stata introdotta per eliminare da P la produzione contestuale $x \rightarrow y$; allora, per come procede l'algoritmo, si deve avere che

$$A \rightarrow \gamma \in P \qquad \text{e} \qquad \gamma \rightarrow_G{}^* \alpha\, x$$

In conclusione abbiamo che

$$w_i = s\,A \rightarrow_G s\,\gamma \rightarrow_G^* s\,\alpha\,x \rightarrow_G s\,\alpha\,y = w_{i+1}$$

(avendo usato in sequenza le riscritture $A \rightarrow \gamma$, $\gamma \rightarrow_G^* \alpha\,x$ e $x \rightarrow y$) cioè

$$w_i \rightarrow_G^* w_{i+1}$$

Da ciò segue quindi che

$$S = w_0 \rightarrow_G^* w_1 \rightarrow_G^* \ldots \rightarrow_G^* w_k = w$$

cioè $w \in L(G)$

c.v.d.

Coroll.1 : *G è OLIN* $\Rightarrow \exists$ *G' regolare t.c. L(G) = L(G')*

Dim.

Basta applicare l'algoritmo che elimina le produzioni contestuali, dando vita ad una G_1 non contestuale equivalente a G che è ancora OLIN (e quindi lineare destra); infine si applica il procedimento di Nota 7 per portare le parti destre di G_1 in forma regolare, ottenendo così una G_2 t.c. $G_2 \equiv G_1 \equiv G$.

c.v.d.

I risultati appena mostrati ci assicurano che grammatiche di tipo OLIN hanno un potere generativo pari a quello delle regolari, ed abbiamo mostrato come simulare le prime con le seconde. Ci si potrebbe quindi chiedere perché introdurre ed usare questa nuova famiglia di grammatiche. La risposta sta nel fatto che, come vedremo tra breve, alcune situazioni si possono formulare in maniera molto compatta e leggibile con questo formalismo. Essendo questi (come detto nell'introduzione) i punti di forza e gli scopi di un approccio generativo ad un qualsiasi problema, il lavoro svolto nel presente capitolo ci sembra pienamente giustificato.

Capitolo 3 : Grammatiche di sincronizzazione

Questo capitolo si occupa del seguente problema : dato un insieme A di eventi elementari (che d'ora in poi sarà considerato come un alfabeto) e data una particolare sequenza di n eventi *s* su A (che sarà quindi una stringa di A^n), trovare un'altra sequenza di n eventi *c* che possano avvenire contemporaneamente con *s* (cioè *s* e *c* devono essere sincronizzabili evento per evento) in base ad un insieme di regole R tali che la sincronizzazione di due eventi elementari al passo i dipenda solo dalle scelte fatte nei k passi precedenti (dove k è una costante). La soluzione verrà data tramite una grammatica generativa di tipo OLIN , e quindi il linguaggio in cui scegliere *c* sarà regolare.

3.1 Il problema della sincronizzazione con regole markoviane

Def. : *una regola markoviana di offset d è una funzione r tale che*

- $r : A^* \times A^* \rightarrow \{ \textbf{true} , \textbf{false} \}$
- $\forall x , y \in A^d \;\; \forall w , w' \in A^* \;$ *si ha che* $\; r (w x , w' y) = r (x , y)$
- $\exists x , y \in A^{d-1} \;\; \exists a , a' \in A \;$ *tale che* $\; r (a x , a' y) \neq r (x , y)$

Def. : *R è un insieme di regole markoviane di offset k se*

- $\forall r \in R \;\; r$ *è una regola markoviana di offset* d *con* $d \leq k$
- $\exists r \in R$ *t.c. r è una regola markoviana di offset* k

OSS.1 : in realtà si può assumere che tutte le regole abbiano offset k . Ciò non cambia il problema poiché, se $r \in R$ ha offset d ($< k$) , allora può tranquillamente essere trattata come una regola di offset k nel senso che

$$r(w\,x , w'\,y) = r(x , y) \quad \forall\, x , y \in A^{d} \;\; \forall\, w , w' \in A^{k-d}$$

(come espresso dalla definizione di regola markoviana di offset d) . Risulta cioè ininfluente considerare i k–d simboli precedenti le stringhe in questione.

Def. : *$s , c \in A^n$ sono sincronizzabili secondo un insieme di regole markoviane di offset k R se*

*$\forall r \in R \;\; \forall i = 1 , \ldots , n$ si ha che $r (s_{i-k+1} \ldots s_i , c_{i-k+1} \ldots c_i) =$ **true***

(avendo posto $s_h = c_h = \lambda$ per $h \leq 0$)

Def. : *$\forall a \in A \;\; S_R(a) = \{ a' \in A :$ a e a' sono sincronizzabili secondo R $\}$*

$\forall s \in A^n \;\; S_s^R = \{ c \in A^n :$ s e c sono sincronizzabili secondo R $\}$

N.B. : $S_R(a)$ è formato dai caratteri di A sincronizzabili con a ed è quindi costruito considerando solo le produzioni di R con offset 1 .

OSS.2 : il problema posto è sempre decidibile; infatti, essendo al più $|A|^n$ il numero dei possibili sincronizzati con *s* ed essendo sia |A| che n numeri finiti, basterà esaminare una per una tutte queste stringhe per vedere se esiste o meno un sincronizzabile.

3.2 Definizione della grammatica

Def. : *La grammatica di sincronizzazione (SG = Synchronizing Grammar) per* $s \in A^n$ *secondo l'insieme* R *di regole markoviane di offset* k *è la quadrupla* $G_s^R = (N , T , S , P)$ *dove*

- $N = \bigcup_{i = 1 , \dots , n} \{ i \} \times \wp (S_R(s_i))$
- $T = \left(\bigcup_{i = 1 , \dots , n} \bigcup_{t \in S_R(s_i)} (s_i , t) \right) \cup \{ ERROR \}$

  (con $ERROR \notin A$ *)*
- $S = (1 , S_R(s_1))$
- $P = \{ (i , \varnothing) \rightarrow ERROR : i = 1 , \dots , n \}$

  $\cup \{ (n , I) \rightarrow (s_n , a) : I \subseteq S_R(s_n) , a \in I \}$

  $\cup \{ \alpha (i , I) \rightarrow \alpha (s_i , a) \left(i+1 , S_R(s_{i+1}) - ILL(i+1 , a_{i-k+2} \dots a_{i-1} \, a) \right)$

  tale che

  . $i = 1 , \dots , n-1$

  . $\alpha = (s_{i-k+2} , a_{i-k+2}) \dots (s_{i-1} , a_{i-1})$

  . $a_j \in S_R(s_j)$ *per* $j = i - k + 2 , \dots , i - 1$

  . $I \subseteq S_R(s_i)$

  . $a \in I$

  }

dove

$$ILL(i , \beta) = \{ b \in S_R(s_i) : \exists r \in R : r (\gamma s_i , \beta b) = \textbf{false} \}$$

avendo posto . $\beta = a_{i-k+1} \dots a_{i-1}$

. $\gamma = s_{i-k+1} \dots s_{i-1}$

. $s_h = a_h = \lambda$ *per* $h \leq 0$

. $(\lambda , \lambda) = \lambda$

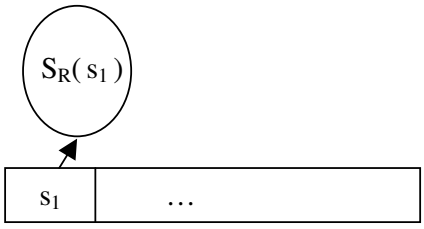
OSS. 3 : una SG è una grammatica OLIN ; infatti le prime due classi di produzioni sono regolari (e quindi OLIN), mentre l'ultima è formata da produzioni contestuali ma tali da contenere solo un non terminale che è l'ultimo simbolo sia delle parti destre che sinistre di ogni produzione.

Def. : *il linguaggio generato dalla SG per* $s \in A^n$ *secondo R è*

$$L(G_s^R) = \{ c \in A^n : S \rightarrow_G^* (s_1, c_1) \dots (s_n, c_n) \}$$

L'idea che sta dietro ad una SG così definita è semplice : data la sequenza di n eventi s , si esamina tale stringa da sinistra a destra.

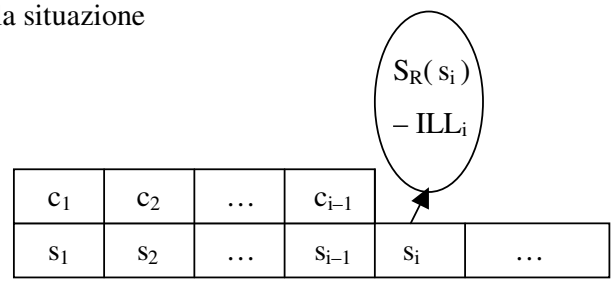
1) Inizialmente si considerano tutti i simboli che possono essere sincronizzati col primo evento



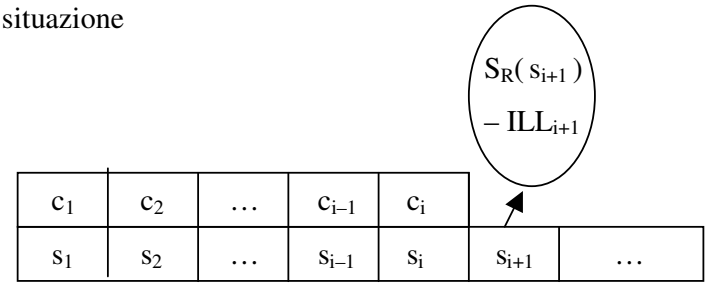
2) Al generico passo i (per $i \geq 1$)

. si sceglie un elemento da sincronizzare con s_i

. si costruisce l'insieme di tutti i simboli di A sincronizzabili con s_{i+1} da cui si tolgono tutti gli eventi che non sono più sincronizzabili a causa delle scelte fatte negli al più k passi precedenti (l'insieme ILL). Graficamente, si passa dalla situazione



alla situazione



Se ad un certo punto si ottiene un insieme vuoto, si decreta un errore (cioè l'impossibilità, in base alle scelte fatte, di continuare a sincronizzare s). E' possibile, quindi, che a causa di un insieme di regole troppo restrittivo e/o di una serie di scelte sfortunate effettuate, si giunga ad un punto in cui qualunque scelta porta a violare almeno una regola di R ; in tal caso la derivazione termina applicando la produzione che introduce il terminale di errore e si ferma rifiutando la stringa prodotta (infatti nella definizione del linguaggio generato dalla SG non sono contenute stringhe col simbolo ERROR). Se si vuole ottenere tutto $L(G_s^R)$ questo aspetto non crea problemi : non deterministicamente si percorrono tutte le possibili derivazioni della SG e si includono nel linguaggio generato tutte e sole le parole ottenute concatenando le seconde componenti delle stringhe derivabili da S. Se invece si vuol ottenere una qualsiasi stringa che sia sincronizzabile con s , allora la possibilità di commettere un errore può dar vita ad un numero non polinomiale di iterazioni del processo di derivazione; bisognerà quindi, a seconda delle applicazioni in cui si userà la SG, valutare quanto il rischio di andare nell'esponenziale sia elevato e quanto ciò possa esser tollerato. Anticipiamo intanto che una soluzione polinomiale è possibile ma altamente inefficiente : pertanto il rischio di backtrack (che peraltro in fase di testing del sistema implementato in ambito musicale si è dimostrato davvero molto basso) può essere preferito in alcune circostanze. Le discuteremo in dettaglio nei prossimi paragrafi mostrando

quanto sia inefficiente in certi ambiti la soluzione polinomiale e di come invece risulti migliore una soluzione con grammatiche di sincronizzazione efficientemente implementate.

3.3 Alcuni semplici esempi

Per meglio spiegare il funzionamento di questo meccanismo generativo, portiamo alcuni semplici esempi che ben rispecchiano però le possibili situazioni reali (che saranno ovviamente molto più complesse).

Prendiamo come alfabeto { a , b , c }

a) una derivazione che ha successo

Prendiamo come regole

R = { r_1 = "Non si può sincronizzare un evento con se stesso" ,

r_2 = "Non si possono avere due eventi uguali di fila" }

e come stringa da sincronizzare la sequenza *a b c b* . Una possibile applicazione della grammatica è

$$(1 , \{ b , c \}) \rightarrow (a , b) (2 , \{ a , c \}) \quad [8]$$
$$\rightarrow (a , b) (b , a) (3 , \{ b \}) \quad [9]$$
$$\rightarrow (a , b) (b , a) (c , b) (4 , \{ a , c \})$$
$$\rightarrow (a , b) (b , a) (c , b) (b , a)$$

[8] Per la regola r_1 i sincronizzabili con a e b sono rispettivamente {b,c} e {a,c}. Per la scelta fatta, ILL(2 , b) = ∅

[9] I sincronizzabili con c sono (sempre per r_1) {a,b}. Ora ILL(3 , b a) = { a } poiché r_2 afferma che non possono esserci due simboli uguali di fila ed al passo precedente era stato scelto a .

ed effettivamente la stringa $b\ a\ b\ a$ è sincronizzabile con $a\ b\ c\ b$ rispettando le regole di R .

b) una derivazione che non ha successo a causa di una scelta infelice

Prendiamo ora l'insieme di regole

$$R' = R \cup \{ r_3 = \text{"la sequenza } a\ b \text{ è vietata"} \}$$

e come stringa $c\ b\ c\ b$. Una possibile applicazione della grammatica è

$$\begin{aligned} (1 , \{ a , b \}) &\rightarrow (c , b) (2 , \{ a , c \}) \\ &\rightarrow (c , b) (b , a) (3 , \varnothing) \quad [10] \\ &\rightarrow (a , b) (b , a) \text{ ERROR} \end{aligned}$$

Notiamo che, se avessimo scelto come secondo simbolo c invece che a , ci sarebbero state tre soluzioni ($b\ c\ a\ c$, $b\ c\ b\ a$ e $b\ c\ b\ c$) .

c) una stringa non sincronizzabile con un insieme di regole

Prendiamo ora l'insieme di regole

$$R'' = R' \cup \{ r_4 = \text{" a deve essere sincronizzato con c "} \}$$

e come stringa $a\ b\ c$. Possiamo facilmente convincerci che non esistono sincronizzazioni possibili poiché :

- il primo carattere della stringa da sincronizzare deve essere c (per r_4)
- quindi il secondo carattere deve essere a (per r_1 e r_2)
- quindi il terzo carattere non può essere una c (per r_1) , non può essere una a (per r_2) e non può essere una b (per r_3) .

[10] I sincronizzabili con c sono (sempre per r_1) {a,b}. Ora ILL(3 , b a) = {a,b} poiché

- r_2 impedisce la presenza di a (altrimenti avremmo due a consecutive)
- r_3 impedisce la presenza di b (altrimenti si avrebbe la sequenza $a\ b$)

Notiamo che l'insieme di regole non è contraddittorio poiché esistono stringhe che sincronizzabili secondo R'' (ad esempio $b\ c\ a$ è sincronizzabile con $c\ a\ c$) .

3.4 Dimostrazione della correttezza

Teor. (correttezza) : $L(G_s^R) \equiv S_s^R$

Dim. (Per doppia inclusione)

- $L(G_s^R) \subseteq S_s^R$

Sia $c \in L(G_s^R)$. Per definizione avremo che

$$S \rightarrow_G^* (s_1 , c_1) \dots (s_n , c_n)$$

da cui

$$\forall\ i = 1 , \dots , n \quad c_i \notin ILL(i , c_{i-k+1} \dots c_{i-1})$$

perché al momento di passare da s_{i-1} a s_i , dai possibili candidati per sincronizzarsi con s_i sono stati eliminati tutti quelli proibiti, contenuti nell'insieme ILL . Quindi, per definizione di ILL , si ha che

$$\forall\ r \in R \quad r (s_{i-k+1} \dots s_{i-1}\ s_i , c_{i-k+1} \dots c_{i-1}\ c_i) = \textbf{\textit{true}}$$

da cui

$$\forall\ r \in R\ \ \forall\ i = 1 , \dots , n \quad r (s_{i-k+1} \dots s_i , c_{i-k+1} \dots c_i) = \textbf{\textit{true}}$$

cioè s e c sono sincronizzabili (cioè $c \in S_s^R$) .

♦ $S_s^R \subseteq L(G_s^R)$

Sia $c \in S_s^R$. Allora per definizione si avrà che

$$\forall r \in R \ \forall i = 1 , \dots , n \quad r (s_{i-k+1} \dots s_i , c_{i-k+1} \dots c_i) = \textbf{\textit{true}}$$

$$\Rightarrow \forall i = 1 , \dots , n \ \big(c_i \in S_R(s_i) \ \wedge \ \forall r \in R \quad r (s_{i-k+1} \dots s_i , c_{i-k+1} \dots c_i) \big)$$

$$\Rightarrow \forall i = 1 , \dots , n \ \big(c_i \in S_R(s_i) \ \wedge c_i \notin ILL(i , c_{i-k+1} \dots c_{i-1}) \big)$$

$$\Rightarrow \forall i = 1 , \dots , n \quad c_i \in S_R(s_i) - ILL(i , c_{i-k+1} \dots c_{i-1})$$

Quindi $\forall I \subseteq S_R(s_{i-1})$ t.c. $c_{i-1} \in I$ avremo in P la produzione

$$\alpha (i-1 , I) \rightarrow \alpha (s_{i-1} , c_{i-1}) (i , S_R(s_i) - ILL(i , c_{i-k+1} \dots c_{i-1}))$$

dove $\alpha = (s_{i-k} , c_{i-k}) \dots (s_{i-2} , c_{i-2})$.

Perciò si può avere in G_s^R la derivazione

$$\begin{aligned}
S &= (1 , S_R(s_1)) \\
&\rightarrow_G (s_1 , c_1) (2 , S_R(s_2) - ILL(2, c_1)) \\
&\rightarrow_G (s_1 , c_1) (s_2 , c_2) (3 , S_R(s_3) - ILL(3, c_1 c_2)) \\
&\qquad \dots \\
&\rightarrow_G (s_1 , c_1) \dots (s_{n-1} , c_{n-1}) (n , S_R(s_n) - ILL(n , c_{n-k+1} \dots c_{n-1})) \\
&\rightarrow_G (s_1 , c_1) \dots (s_{n-1} , c_{n-1}) (s_n , c_n)
\end{aligned}$$

cioè $c \in L(G_s^R)$

<u>*c.v.d.*</u>

3.5 Complessità del problema

Per discutere la complessità del problema, lo riformuliamo nei suoi tratti essenziali.

Siano dati :

- A un alfabeto finito
- R un insieme finito di funzioni del tipo

$$r : A^k \times A^k \rightarrow \{ 0 , 1 \}$$

(dove k è una costante)

- $s \in A^n$

Si determini $c \in A^n$ tale che

$$\forall\, i = 1 , \ldots , n\text{-}k+1 \quad \forall\, r \in R \quad \text{si ha} \quad r(s_i \ldots s_{i+k-1} , c_i \ldots c_{i+k-1}) = 1$$

N.B. : per l'OSS.1 di questo capitolo il problema risulta equivalente a quello sopra formulato anche considerando tutte le regole di stesso offset.

Così formulato il problema può essere ridotto ad una visita in profondità di un grafo di dimensione costante in n (cioè la lunghezza della stringa data) con nodi etichettati da stringhe binarie lineari in n (il cui scopo sarà chiaro tra breve), e quindi risulta risolvibile in tempo polinomiale. Vediamo come procede questa costruzione.

Consideriamo come vertici del grafo tutte le stringhe di A^k . Andiamo ora a calcolarci per ognuna di esse una etichetta nella maniera seguente

$$\forall\, c \in A^k \quad label(c) = \alpha \in \{ 0 , 1 \}^{n-k+1} \text{ t.c.}$$

$$\forall\, i = 1 , \ldots , n\text{–}k+1 \quad \alpha_i = \prod_{r \in R} r (s_i \ldots s_{i+k-1} , c_i \ldots c_{i+k-1})$$

(cioè l'etichetta del nodo corrispondente alla stringa c contiene in posizione i il risultato della congiunzione di tutte le regole di R calcolate su s e c a partire dal carattere i). Tali etichette servono per creare l'insieme degli archi del grafo $E \subseteq V \times V$. La sua costruzione avviene in maniera iterativa, con la procedura portata nel riquadro sottostante; in essa si fa uso del concetto di shifted string che andiamo ora a definire

Def. : *sia* $\delta \in A^k$ *; la shifted string di* δ *tramite il carattere* c $(\in A)$ *è la stringa*

$$\sigma_c(\delta) = \delta_2 \ldots \delta_k\, c$$

essendo $\delta = \delta_1\, \delta_2 \ldots \delta_k$

```
1)  E = ∅
2)  U1 = { ν ∈ V  t.c.  label( ν )1 = 1 }
3)  for  i = 2  to  n – k + 1
4)       Ui = ∅
5)       ∀ ν ∈ Ui–1  ∀ υ ∈ { σc( ν ) : c ∈ A }
6)            if  label( υ )i = 1  then
7)                 Ui = Ui ∪ { υ }
8)                 E = E ∪ { < ν , υ > }
9)  return E
```

In pratica, si parte dai nodi la cui etichetta ha 1 in prima posizione (cioè dai nodi la cui stringa associata soddisfa con s tutte le regole di R applicate ai primi k caratteri) [riga 2]. Dopodiché si entra nel ciclo di [riga 3] in cui si analizzano tutti i nodi selezionati al passo precedente e si cercano tutte le shifted string della stringa associata al nodo in esame [riga 5] ; per ognuna di esse si vede se l'etichetta del nodo associato ha 1 nella posizione corrente

[riga 6] ed in caso positivo aggiunge quest'ultimo nell'insieme corrente di vertici e crea un arco tra il nodo precedente ed il corrente [righe 7 e 8].

Teor.1 : *Sia* $t \geq k$ *; allora* $c_1 \ldots c_t \in A^t$ *è sincronizzabile con* $s_1 \ldots s_t$ *secondo* R *se e solo se* $c_1 \ldots c_k \rightsquigarrow c_{t-k+1} \ldots c_t$ *è un cammino da* U_1 *a* U_{t-k+1} *nel grafo costruito con l'algoritmo precedente.*

Dim.

($\Rightarrow$) Se $c_1 \ldots c_t$ è sincronizzabile con $s_1 \ldots s_t$ allora :

- $\forall\, r \in R \quad r(s_1 \ldots s_k , c_1 \ldots c_k) = 1$
- $\forall\, r \in R \quad r(s_2 \ldots s_{k+1} , c_2 \ldots c_{k+1}) = 1$

  …
- $\forall\, r \in R \quad r(s_{t-k+1} \ldots s_t , c_{t-k+1} \ldots c_t) = 1$

da cui (per definizione dell'etichetta di un nodo e per il funzionamento dell'algoritmo) :

- $c_1 \ldots c_k \in U_1$
- $c_2 \ldots c_{k+1} \in U_2$ e $< c_1 \ldots c_k , c_2 \ldots c_{k+1} > \in E$

  …
- $c_{t-k+1} \ldots c_t \in U_{t-k+1}$ e $< c_{t-k} \ldots c_{t-1} , c_{t-k+1} \ldots c_t > \in E$

($\Leftarrow$) Basta invertire il ruolo di antecedenti e conseguenti nell'argomentazione precedente.

c.v.d.

Coroll.1 : *Se l'algoritmo termina con* $U_{n-k+1} \neq \varnothing$ *allora ogni cammino da un nodo di* U_1 *ad uno di* U_{n-k+1} *costituisce una soluzione del problema . Se invece l'algoritmo termina con* $U_{n-k+1} = \varnothing$ *allora* s *non è sincronizzabile secondo* R *.*

Il problema della sincronizzazione può dunque esser posto in termini di visita del grafo così costruito. Tale grafo ha un numero di archi e vertici che non dipende dalla dimensione di s (essi dipendono solo da R che può essere considerato fissato una volta per tutte); da tale lunghezza invece dipende la lunghezza delle etichette che è lineare in n. La costruzione del grafo pertanto può essere fatta in tempo polinomiale in n, così come la sua visita.

Purtroppo però le costanti nascoste possono essere enormi. Nell'applicazione musicale in seguito trattata valori tipici dei dati del problema sono :

- A sarà l'insieme delle note disponibili ($|A| \approx 20$)
- R sarà formato dalle regole contrappuntistiche ($|R| \approx 20$)
- $k = 7$
- n sarà molto piccolo (raramente $n > 6$)

In questo caso l'applicazione dell'algoritmo esposto richiederebbe uno spazio di memoria di parecchi MegaByte dovendo memorizzare un grafo con $|A|^k$ ($\approx 10^9$) vertici e quindi tempi di elaborazione lunghi. Pertanto risulta sicuramente meglio l'applicazione di una SG soprattutto con l'implementazione efficiente che esporremo nel prossimo paragrafo : nonostante ci sia il rischio di backtrack, le prestazioni medie sono di gran lunga migliori, considerando soprattutto il fatto che il ricorso al backtrack ad ogni passo non è molto probabile (tipicamente si può sempre trovare una soluzione corretta – anche se musicalmente non entusiasmante – in quasi tutte le situazioni).

3.6 Una implementazione efficiente

Come per il grafo del paragrafo precedente, anche la costruzione della SG per una stringa $s \in A^n$ è O(n) con costanti moltiplicative nascoste enormi. Quindi un algoritmo che prima costruisce la SG e poi la applica è sicuramente un algoritmo polinomiale ma è altamente inefficiente (come l'algoritmo precedente sul grafo); basti pensare poi che la quasi totalità delle produzioni risulteranno inutilizzate. Infatti è inutile introdurre la produzione

$$\alpha (i , I) \rightarrow \alpha (s_i , a) \big(i+1 , S_R(s_{i+1}) - \mathrm{ILL}(i+1 , a_{i-k+2} \ldots a_{i-1}\, a) \big)$$

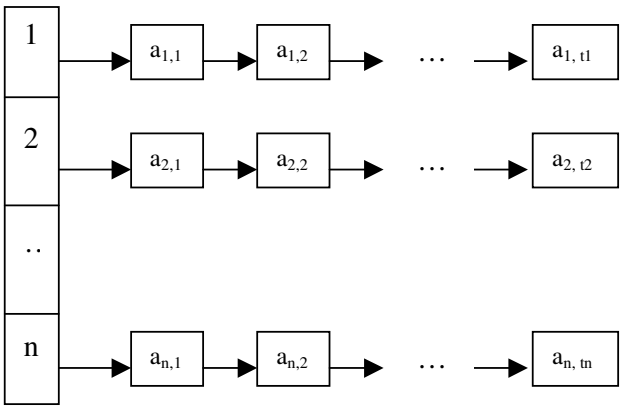
(con $\alpha = (s_{i-k+2} , a_{i-k+2}) \ldots (s_{i-1} , a_{i-1})$) se al passo i non è stato selezionato il carattere a ma invece è stato selezionato a' .

Sarebbe quindi meglio costruire la grammatica di pari passo con l'esplorazione di *s* . Inoltre, in caso di errore al passo i , non è necessario ricominciare tutta la derivazione da capo : basterebbe risalire (tramite backtrack) al primo carattere precedente s_i che era sincronizzabile con più di un carattere ed effettuare un'altra scelta. Questi sono gli scopi del presente paragrafo; l'esposizione finora fatta è comunque importante perché ci garantisce che il procedimento che sta dietro alla realizzazione che proporremo è in realtà l'iterazione di una grammatica regolare e quindi un meccanismo computazionale elegante e semplice.

Strutture dati

Sia la stringa da sincronizzare *s* che la stringa sincronizzata *c* sono rappresentati da array di n caratteri scelti da A che chiameremo S e C (con indicizzazione Pascal , da 1 a n).

Per poter realizzare backtrack basta avere un array di n liste contenenti per ogni indice j dell'array i caratteri sincronizzabili con s_j , cioè



dove $\{ a_{i,1} , \ldots , a_{i,ti} \} = S_R(s_i)$. Tale struttura dati sarà chiamata L .

Procedure

Le procedure sono due, una per la costruzione on–line delle produzioni della SG , l'altra per la realizzazione del backtrack : entrambe condividono S , C e L . Eccole in dettaglio.

a) Costruzione delle produzioni

Al generico passo i, la procedura deve essenzialmente calcolarsi ILL (accedendo alle scelte fatte nei k passi precedenti in C e ai corrispondenti caratteri di S), togliere il risultato dalla locazione i–esima di L (ottenendo così $S_R(s_i) - ILL$) e tra gli elementi restanti sceglierne uno e porlo in C alla locazione i. Eccone lo pseudo–codice :

```
Genera_carattere ( i )

∀ a ∈ L[i]
    ∀ r ∈ R
        if NOT r ( S[i-k+1] … S[i] , C[i-k+1] … C[i-1] a )
        then L[i] . cancella( a )
             break
if L[i] == ∅ then ERROR
else C[i] = L[i] . scegli_elemento( )
```

In questo modo sono state create solo le produzioni necessarie per aggiungere il carattere i–esimo in C in base alle scelte effettivamente compiute nei k passi precedenti, evitando le costruzione di un numero di produzioni esponenziale in k. Così come abbiamo supposto $s_h = \lambda$ per $h \leq 0$ nella definizione delle SG, così qui supponiamo l'esistenza di un controllo sull'indicizzazione degli array S e C che restituisca λ in caso di accesso a locazioni non positive (che omettiamo data la sua semplicità). Supponiamo inoltre che le regole di R siano o calcolabili in maniera efficiente tramite una opportuna procedura oppure siano date

come una tabella che, presi come input $s_{i-k+1} \dots s_i$ e $c_{i-k+1} \dots c_i$ restituisce $r (s_{i-k+1} \dots s_i , c_{i-k+1} \dots c_i)$.

b) *Gestione del backtrack*

Come abbiamo visto, nella procedura precedente si può entrare in una situazione di errore quando l'insieme $S_R(s_i) - ILL$ risulta vuoto (cioè non si ha nessun evento sincronizzabile con s_i senza violare regole di R) . In tal caso bisogna tornare alla prima locazione di L precedente la locazione i in cui si possano compiere scelte diverse (chiamiamola j); tale locazione sarà tale che $| L[j] | > 1$ (cioè conterrà l'elemento messo in C[j] e qualche altro elemento tra cui compiere la nuova scelta). Ovviamente tutte le locazioni tra la i (compresa) e la j (esclusa) dovranno venir resettate al loro valore originale (cioè $L[t] = S_R(s_t) \quad \forall\ t = j+1 , \dots , i$) . Questo è necessario perché le eliminazioni fatte in precedenza erano dovute anche alla scelta compiuta al passo j ; cambiata tale scelta tali eliminazioni non è detto che siano ancora necessarie (anzi in generale non lo saranno).

```
Backtrack ( i )

for  j = i  downto  1
     if   | L[j] | >  1
     then    L[j] . cancella( C[j] )
             C[j]  =  L[j] . scegli_elemento( )
             return  j
     else    L[j]  =  S_R( s_j )
return  0
```

E' interessante notare che se, procedendo a ritroso, si arriva alla prima locazione e si rileva che anche qui non c'era rimasta che una sola possibilità (fatto che avviene se si esegue per intero il FOR senza cioè uscire per il RETURN contenuto nel THEN) allora vuol dire che tutti i tentativi possibili per sincronizzare *s* sono stati compiuti e che sono falliti tutti (cioè *s* non è sincronizzabile in base ad R).

Funzionamento globale

Siamo quindi in grado di descrivere il funzionamento globale dell'algoritmo che genera una stringa sincronizzabile con la stringa *s* data.

```
Sincronizza ( s )

n = | s |
for i = 1 to n
    S[i] = s_i
    C[i] = λ
    L[i] = S_R( s_i )
i = 1
while i ≤ n
    Genera_carattere ( i )
    if ERROR then i = Backtrack ( i )
                 else i = i+1
    if i == 0 then ABORT //impossibile sincronizzare s
return concatena( C[1] , … , C[n] )
```

3.7 Confronto tra Sticker Systems e Synchronizing Grammars

Vogliamo ora mettere in luce i punti di contatto e le differenze tra i due meccanismi presentati. Anzitutto entrambi lavorano su coppie di stringhe "appaiate" [11] e con una relazione di accoppiamento che opera sui simboli dell'alfabeto dato [12] : una prima differenza sta nel fatto che nel DNA Computing tale relazione è simmetrica (anche se nella trattazione precedente ciò non è stato usato), mentre nel caso delle SG ciò non è vero [13] . Comunque le due proprietà della proposizione del paragrafo 1.2 valgono anche per le SG.

Il meccanismo di generazione di una SG può essere paragonato a quello di uno sticker system semplice e regolare con una regolarità molto forte poiché

$$\forall\ (u, v) \in D \text{ si ha che } u = \begin{bmatrix} \lambda \\ \\ \lambda \end{bmatrix} \text{ (sticker system semplice) e } v \in \begin{pmatrix} V^* \\ \\ \lambda \end{pmatrix}$$

[11] Usiamo questo termine per indicare il fatto che ad ogni simbolo della prima ne corrisponde uno della seconda.

[12] N.B.: nel caso del DNA Computing tale alfabeto è stato chiamato V mentre nel contesto delle SG era chiamato A .

[13] Notiamo che nel DNA Computing l'accoppiamento era stato definito come $\rho \subseteq V \times V$ mentre nelle SG si aveva che $S_R : A \to \wp(A)$. Questo modo di definire l'accoppiamento è però identico poiché :

a) data ρ definiamo $f_\rho : V \to \wp(V)$ t.c. $f_\rho(v) = \{ v' \in V : (v, v') \in \rho \}$

b) data S_R definiamo $\sigma_R \subseteq A \times A$ t.c. $\forall\ a' \in S_R(a)$ si ha che $(a, a') \in \sigma_R$

e quindi l'operazione di sticking è sempre del tipo (6) con $x1 = \lambda$. E' importate notare, però, che nel meccanismo generativo degli sticker systems non è possibile tenere conto di dipendenze markoviane : ogni simbolo della catena superiore è determinato solo in base al corrispondente simbolo della catena inferiore, indipendentemente dal passato più o meno recente.

Se vogliamo ora confrontare i rispettivi poteri generativi, nonostante l'impossibilità di realizzare con sticker systems le dipendenze markoviane, dobbiamo comunque notare che ogni SG può essere simulata con uno sticker system, mentre il viceversa non è vero; ciò è dovuto al fatto che le prime generano comunque un insieme di stringhe finito e tutte della stessa lunghezza, mentre i secondi possono procedere all'infinito, generando quindi stringhe di lunghezze differenti. Più precisamente, data la SG

$$G_s^R = (N , T , S , P)$$

per l'insieme di regole R e per la stringa $s \in A^n$, si può creare lo sticker system

$$\gamma = (A , S_R , X , \varnothing)$$

dove X è l'insieme

$$X = \{ \begin{bmatrix} c \\ \\ s \end{bmatrix} : c \in L(G_s^R) \}$$

Ci costruiamo cioè uno sticker system che ha come assiomi le molecole ottenute ponendo nella catena inferiore s ed in quella superiore tutte le stringhe $c \in A^n$ prodotte dalla grammatica. Non essendoci regole di derivazione (l'ultimo componente di γ è l'insieme vuoto), si hanno banalmente le seguenti uguaglianze che mostrano l'equivalenza tra G_s^R e γ

$$LM(\gamma) = X$$
$$L(\gamma) = L(G_s^R)$$

Viceversa basta prendere un qualunque sticker system che produca stringhe di lunghezza diversa (a tal scopo basta prenderne uno che generi un linguaggio infinito) per non essere in grado di simularlo con una grammatica di sincronizzazione. Esso potrebbe essere simulato con un'infinità numerabile di grammatiche di sincronizzazione G_1 , G_2 , … G_n , … (una per ogni lunghezza) con regole di sincronizzazione di offset 1 date da ρ . Più precisamente

- $R = \bigcup_{a \in A} r_a$

  con $r_a : A \times A \rightarrow \{ \textbf{\textit{true}} , \textbf{\textit{false}} \}$ t.c.

  $$r_a (a , b) = \textbf{\textit{true}} \quad \text{sse} \quad (a , b) \in \rho$$

- $\forall \ (x , y) \in LM(\gamma) \quad y \in L (G_{|x|}^R (x))$

  (poiché la grammatica applica semplicemente la complementarietà data da ρ)

In realtà siamo propensi a credere che possa esistere una formulazione che usi un numero finito di grammatiche (una per ogni domino di produzione). A tale trattazione sarà dedicato un lavoro autonomo essendoci molti aspetti da valutare che, ai fini della presente tesi, risulterebbero comunque non applicati.

Capitolo 4 : Grammatiche di sincronizzazione generalizzate

Questo capitolo si occupa di generalizzare leggermente il problema della sincronizzazione : infatti, sono date ora m particolari sequenza di n eventi $s^{(1)}, \ldots, s^{(m)}$ su A e si deve trovare un'altra sequenza di n eventi c che possano avvenire contemporaneamente con le stringhe date in base ad un insieme R di regole markoviane di offset k. La soluzione sarà uguale a quella per le SG, così come i risultati ottenuti e la possibile implementazione, tranne che per una lieve complicazione dovuta alla presenza di m stringhe date. Per semplificare le notazioni (che sono l'unica cosa che cambia) introduciamo le seguenti convenzioni :

- per indicare la m–pla $s^{(1)}, \ldots, s^{(m)}$ scriveremo $\underline{s}$
- per indicare il prodotto cartesiano di A^n con se stesso per m volte (cioè per indicare l'insieme delle m–ple formate da stringhe lunghe n su A) scriveremo $A^{n \times m}$ (quindi, poiché $s^{(i)} \in A^n$, si ha che $\underline{s} \in A^{n \times m}$)
- per indicare i caratteri dal k–esimo al j–esimo inclusi di $s^{(i)}$ scriveremo $s^{(i)}{}_{<k \ldots j>}$
- per indicare la m–pla $s^{(1)}{}_{<k \ldots j>}, \ldots, s^{(m)}{}_{<k \ldots j>}$ scriveremo $\underline{s}_{<k \ldots j>}$
- per indicare la m–pla ottenuta concatenando una stringa $w \in A^*$ con ogni elemento della m–pla $\underline{x}$ scriveremo $w\,\underline{x}$

Con tali convenzioni le definizioni date nel corso del capitolo precedente restano formalmente molto simili e quindi leggibili e comprensibili.

Def. : *una regola markoviana di offset d e di arità m+1 è una funzione r tale che*

- $r : A^*_{\times m} \times A^* \rightarrow \{ \textbf{true} , \textbf{false} \}$
- $\forall \underline{x} \in A^d_{\times m} \; \forall \underline{y} \in A^d \;\; \forall w , w' \in A^* \;$ *si ha* $r (w \underline{x} , w' \underline{y}) = r (\underline{x} , \underline{y})$
- $\exists \underline{x} \in A^{d-1}_{\times m} \; \exists \underline{y} \in A^{d-1} \;\; \exists a , a' \in A \;$ *tale che* $r (a \underline{x} , a' \underline{y}) \neq r (\underline{x} , \underline{y})$

OSS.1 : tipicamente, dato un insieme di regole markoviane di arità 2 (cioè le regole usate per il problema della sincronizzazione nella formulazione del capitolo precedente), esso potrà essere generalizzato in un insieme di regole con arità m+1 . In particolare, data r di offset d e arità 2 , si ottiene una regola r' sempre di offset d ma di arità m+1 ponendo

$$r'(x_1 , \ldots , x_m , y) = r(x_1 , y) \wedge \ldots \wedge r(x_m , y)$$

Chiaramente il viceversa non è vero (cioè non tutte le regole di arità m+1 possono essere scritte come congiunzione di m regole di arità 2).

Def. : *R è un insieme di regole markoviane di offset k e arità m+1 se*

- $\forall r \in R$ *r è una regola markoviana di offset d e arità m+1 con* $d \leq k$
- $\exists r \in R$ *t.c. r è una regola markoviana di offset k e arità m+1*

Def. : $\underline{s} \in A^n_{\times m}$ *e* $c \in A^n$ *sono sincronizzabili secondo un insieme R di regole markoviane di offset k e arità m+1 se*

$\forall r \in R \;\; \forall i = 1 , \ldots , n$ *si ha* $r (\underline{s}_{< i-k+1 \ldots i >} , c_{< i-k+1 \ldots i >}) = \textbf{true}$

(mantenendo sempre la convenzione che $s^{(i)}_h = c_h = \lambda$ *per* $h \leq 0$ *)*

Def. : $\forall \underline{a} \in A^m \quad S_R(\underline{a}) = \{ c \in A :$ $\underline{a}$ *e c sono sincronizzabili secondo R* $\}$

$\forall \underline{s} \in A^n_{\times m} \quad S_{\underline{s}}^R = \{ c \in A^n :$ $\underline{s}$ *e c sono sincronizzabili secondo R* $\}$

OSS.2 : chiaramente $S_{R'}(\underline{a}) = \bigcap_{i = 1, \dots, m} S_R(a_i)$ dove R è un insieme di regole markoviane di arità 2 e R' è il corrispondente insieme di regole di arità m+1 (ottenuto come da OSS.1)

OSS.3 : se R è un insieme di regole markoviane di arità 2 , R' è il corrispondente insieme di arità m+1 e $R' \subseteq R''$, allora

$$S_{R''}(\underline{a}) \subseteq \bigcap_{i = 1, \dots, m} S_R(a_i)$$

Def. : *La grammatica di sincronizzazione generalizzata (GSG = Generalized Synchronizing Grammar) per* $\underline{s} \in A^{n \times m}$ *secondo l'insieme R di regole markoviane di offset k e arità m+1 è la quadrupla* $G_{\underline{s}}^{R} = (N , T , S , P)$ *dove*

- $N = \bigcup_{i = 1, \dots, n} \{ i \} \times \wp (S_R(\underline{s}_{<i>}))$
- $T = \left(\bigcup_{i = 1, \dots, n} \bigcup_{t \in S_R(\underline{s}_{<i>})} (\underline{s}_{<i>} , t) \right) \cup \{ ERROR \}$
- $S = (1 , S_R(\underline{s}_{<1>}))$
- $P = \{ (i , \varnothing) \rightarrow ERROR : i = 1 , \dots , n \}$
  $\cup \{ (n , I) \rightarrow (\underline{s}_{<n>} , a) : I \subseteq S_R(\underline{s}_{<n>}) , a \in I \}$
  $\cup \{ \alpha (i,I) \rightarrow \alpha (\underline{s}_{<i>} , a) \left(i+1, S_R(\underline{s}_{<i+1>}) - ILL(i+1 , a_{i-k+2} \dots a_{i-1} a) \right)$
  tale che
  . $i = 1 , \dots , n-1$
  . $\alpha = (\underline{s}_{<i-k+2>} , a_{i-k+2}) \dots (\underline{s}_{<i-1>} , a_{i-1})$
  . $a_j \in S_R(\underline{s}_{<j>})$ *per* $j = i-k+2 , \dots , i-1$
  . $I \subseteq S_R(\underline{s}_{<i>})$
  . $a \in I$
  }

dove

$ILL(i, \beta) = \{ b \in S_R(\underline{s}_{<i>}) : \exists r \in R : r(\underline{s}_{<i-k+1 \ldots i>}, \beta b) = \textbf{false} \}$

con $\beta = a_{i-k+1} \ldots a_{i-1}$ *e la solita convenzione* $(\lambda, \lambda) = \lambda$

Def. : *il linguaggio generato dalla GSG per* $\underline{s} \in A^n{}_{\times m}$ *secondo R è*

$$L(G_{\underline{s}}^{R}) = \{ c \in A^n : S \rightarrow_G^* (\underline{s}_{<1>}, c_1) \ldots (\underline{s}_{<n>}, c_n) \}$$

Avendo questo formalismo praticamente identico, è inutile riportare dimostrazioni e implementazioni di una GSG : basta riportare nel paragrafo precedente quei piccoli cambi di notazione fatti in questo paragrafo per ottenere risultati identici.

PARTE II

APPLICAZIONE MUSICALE

Capitolo 5 : Richiami musicali

5.1 La struttura di una fuga

La fuga è la forma musicale tipica del barocco musicale : nasce nella seconda metà del 1600 e trova il suo massimo compimento nelle opere di Johann Sebastian Bach, compositore attivo in Germania nella prima metà del 1700. Le sue caratteristiche fondamentali sono due : è una forma *polifonica* (nel senso che ci sono più linee melodiche – o voci – che vengono eseguite contemporaneamente) e *monotematica* (basata cioè su di un unico tema – il soggetto). Tale motivo viene trattato con complessi procedimenti contrappuntistici che consistono nell'eseguire il soggetto (o alcune sue parti) opportunamente elaborato tramite procedimenti imitativi tra le varie voci.

In realtà tali caratteristiche sono riscontrabili anche in composizioni ampiamente precedenti (già nel 1300 e 1400 si scrivevano composizioni simili): ciò che invece caratterizza la fuga e le dà vita autonoma è il complesso insieme di regole che governano lo sviluppo della composizione. In estrema sintesi la struttura è la seguente :

– *esposizione :* è l'inizio della fuga ed il suo scopo è di presentare nella maniera più chiara possibile per l'ascoltatore il soggetto; per una trattazione più dettagliata vedi oltre.

– *divertimenti :* lo scopo è quello di preparare il ritorno del tema (con le conseguenti imitazioni) in tonalità diverse; hanno quindi una funzione

modulante. Tipicamente sono costruiti usando parti del soggetto o del controsoggetto (vedi oltre) combinate con procedimenti imitativi.

– *stretto :* ritorno del soggetto nella tonalità iniziale con la caratteristica che le entrate in imitazione avvengono prima della fine dell'esecuzione completa del soggetto da parte della voce precedente (sono cioè ravvicinate, da cui il nome stretto). Questo crea un grande senso di concitazione per il finale imminente.

– *pedale :* conclude la fuga tenendo fissa al basso la tonica della tonalità d'impianto (la nota "più importante" della composizione) e su cui le voci superiori ripropongono per l'ultima volta il soggetto.

Non è chiaramente questo il luogo adatto ad approfondire ulteriormente questo argomento : da tre secoli a questa parte teorici e compositori hanno dato vita a decine di manuali e trattati che tentano di descrivere tutti i dettagli di questa forma così ricca e complessa (tra i più importanti [Dub] e [Ged]; da essi sono tratti tutti gli elementi di teoria di contrappunto e fuga che presenteremo). In conclusione, vale la pena di rilevare l'importanza storica questa forma : essa è infatti la sintesi delle manifestazioni musicali dal gregoriano (IV – V sec. d.C.) fino al periodo barocco. In essa confluiscono infatti caratteristiche come la monotematicità (proveniente dalla tradizione gregoriana), la polifonia (introdotta attorno al 1200 e mai scomparsa), i complessi procedimenti imitativi (sviluppati al massimo dai fiamminghi del 1400) nonché una forte caratterizzazione armonica e tonale (tipica del 1600 e 1700). E' quindi la sintesi di quasi 1500 anni di musica occidentale ed anche per questo è una delle forme più studiate ed usate fino ai giorni nostri.

5.2 L'esposizione di una fuga

Come già detto in precedenza, l'esposizione è la parte iniziale della fuga il cui compito è quello di esporre il materiale tematico su cui la composizione sarà costruita. Essa è formata da tre elementi fondamentali :

- ***il soggetto :*** è il tema su cui si basa tutta la composizione; tipicamente viene eseguito da solo (senza alcun accompagnamento) da una voce all'inizio dell'esposizione affinché sia chiaramente udito e memorizzato dagli ascoltatori.

- ***la risposta :*** è la trasposizione del soggetto nella tonalità della dominante. Questa trasposizione non è così semplice come sembra : infatti nel portare il soggetto dalla tonalità d'impianto a quella della relativa dominante bisogna fare attenzione a non uscire dall'ambito della tonalità originale, poiché l'esposizione deve tassativamente svolgersi nell'ambito della tonalità d'impianto. Si avranno quindi due casi :
  - i) se il soggetto non modula in dominante (*fuga reale*) $\Rightarrow$ basta trasportare il soggetto nel relativo V grado
  - ii) se il soggetto modula in dominante (*fuga tonale*) $\Rightarrow$ nel trasportare il soggetto nel relativo V grado bisognerà evitare la sensazione di modulare nella dominante del V grado, altrimenti l'impressione sarebbe quella di trovarsi nella tonalità di dominante e non più di tonica. Saranno quindi necessarie delle mutazioni del soggetto finalizzate ad evitare ciò.

- ***il controsoggetto :*** è l' "accompagnamento ufficiale" del soggetto : ogni volta che una voce eseguirà il soggetto, un'altra voce eseguirà il

controsoggetto assieme ad essa. Il controsoggetto deve pertanto essere distinguibile dal soggetto e quindi dovrà avere caratteristiche musicali differenti (N.B.: non è però corretto pensare che la fuga sia una forma bitematica : infatti il controsoggetto non ha una vita autonoma ma vive solo in funzione del soggetto). Ovviamente anche il controsoggetto potrà fornire materiale tematico utile per i divertimenti e potrà subire gli stessi processi contrappuntistici del soggetto (andamento per moto contrario o retrogrado, aumentazione, diminuzione, …) ; è inoltre possibile, anche se poco frequente, avere fughe con più controsoggetti che si alternano nell'accompagnare il soggetto.

La fuga nella sua formulazione più tradizionale è a quattro voci (soprano, contralto, tenore e basso) : è questa la forma che adotteremo. In realtà sono comuni (soprattutto in ambito strumentale) fughe a 2, 3, 5 o più voci. La loro trattazione è però ottenuta adattando il modello classico a tali nuove esigenze e pertanto non sarà qui considerato.

In ambito scolastico, il soggetto è tipicamente dato; è invece compito dello studente crearvi uno (o più) controsoggetti leciti, nonché costruire una possibile risposta (N.B.: la complessità di tale operazione è testimoniata dal fatto che in una fuga tonale ci sono spesso più possibili risposte, tutte lecite e logiche). Una volta fatto ciò, organizzare l'esposizione risulta piuttosto semplice : basta infatti decidere l'ordine d'entrata delle voci e creare le cosiddette parti libere.

Le entrate seguono le seguenti regole :

- soggetto e risposta devono sempre alternarsi
- la voce che ha appena terminato l'esposizione del soggetto (o della risposta) è quella che eseguirà il controsoggetto in concomitanza dell'entrata successiva
- la risposta deve entrare immediatamente dopo la fine del soggetto, tranne nel caso in cui tale ingresso immediato modifichi la struttura metrica del soggetto : in tal caso una breve coda alla fine del soggetto è lecita
- la prima e la terza entrata, nonché la seconda e la quarta, devono avvenire in voci di tessitura corrispondente (soprano – tenore e contralto – basso)

Le parti libere sono invece la prosecuzione del controsoggetto e servono a far proseguire una voce che ha appena esposto soggetto e controsoggetto mentre le altre voci fanno lo stesso. Schematicamente l'esposizione può avere le seguenti forme

Sopr	SOGGETTO	CONTROSOGG	*Parte libera*	*Parte libera*
Contr		RISPOSTA	CONTROSOGG	*Parte libera*
Ten			SOGGETTO	CONTROSOGG
Basso				RISPOSTA

Sopr				RISPOSTA
Contr			SOGGETTO	CONTROSOGG
Ten		RISPOSTA	CONTROSOGG	*Parte libera*
Basso	SOGGETTO	CONTROSOGG	*Parte libera*	*Parte libera*

Sopr				RISPOSTA
Contr	SOGGETTO	CONTROSOGG	*Parte libera*	*Parte libera*
Ten		RISPOSTA	CONTROSOGG	*Parte libera*
Basso			SOGGETTO	CONTROSOGG

Sopr			SOGGETTO	CONTROSOGG
Contr		RISPOSTA	CONTROSOGG	*Parte libera*
Ten	SOGGETTO	CONTROSOGG	*Parte libera*	*Parte libera*
Basso				RISPOSTA

Una trattazione completa delle regole e dei modi di realizzare controsoggetto e parti libere è contenuta nella prosecuzione di questa sezione della tesi. Per quanto riguarda invece la scelta dell'ordine di entrata basta invece estrarre a sorte una delle quattro possibilità appena esposte e completarle con soggetto, controsoggetto e parti libere che saranno create. La problematica della risposta è invece stata tralasciata (per il momento cioè si analizzeranno solo fughe reali) : questa scelta è motivata dal fatto che una trattazione completa ed esauriente della risposta richiederebbe un lavoro molto grande e complesso che in prima istanza può essere tranquillamente tralasciato, per venir ripreso in un lavoro successivo e specificamente orientato alla soluzione di questo problema. Pertanto la risposta sarà d'ora in poi intesa come la semplice trasposizione del soggetto nella tonalità di dominante.

Capitolo 6 : La generazione del controsoggetto

6.1 Regole del contrappunto doppio di prima specie

Dato il soggetto, la prima operazione da compiere è determinarne un controsoggetto lecito rispetto alle regole del contrappunto doppio. Il ***contrappunto doppio*** è un caso particolare del ***contrappunto a due*** : quest'ultimo è quella parte della composizione musicale che si occupa, dato un tema, di trovarne un altro che possa essere eseguito contemporaneamente al primo (il nome deriva dal fatto che sono coinvolte solo due voci, una per il primo tema, una per il secondo). La differenza tra contrappunto a due e contrappunto doppio sta nel fatto che il secondo cerca un tema che possa essere indifferentemente eseguito da una voce superiore o inferiore a quella che esegue il tema dato (mentre il contrappunto a due non consente questa operazione, tecnicamente chiamata *rivolto delle parti* : in tale ambito è infatti rilevante quale voce tiene il canto dato e quale realizza il contrappunto ed in generale le due parti non possono essere scambiate).

La specie dipende dai rapporti ritmici tra tema e contrappunto : quella che analizzeremo prevede che ad ogni nota del soggetto ne corrisponda una nel controsoggetto. E' quindi pensata per dar vita a contrappunti per note tutte della stessa durata : questa scelta esclude la problematica ritmica che all'interno della fuga è molto rilevante (infatti tipicamente soggetto e controsoggetto sono chiaramente distinti per il diverso ritmo che li caratterizza). Questa scelta porta ad un forte limite nella trattazione seguente in termini di vicinanza ad una fuga reale : si tratta però di una semplificazione

accettabile in quanto il presente lavoro vuole essere una prima approssimazione di un sistema formale che possa descrivere compiutamente l'intero processo compositivo di una fuga. Pertanto la problematica ritmica può essere affrontata in un secondo momento, come arricchimento della presente trattazione.

Esistono vari tipi di contrappunto doppio, in base all'intervallo massimo che intercorre tra tema dato e contrappunto : quello che scolasticamente (ed anche in pratica) è il più usato è quello *all'ottava* [14] , anche se si hanno frequenti esempi (soprattutto nella musica strumentale) di contrappunti alla decima, all'undicesima ed alla dodicesima, mentre più raramente usati sono quelli alla tredicesima ed alla quindicesima [15]. Pertanto la scelta fatta in questa tesi è quella di usare solo il *contrappunto doppio di prima specie all'ottava* per la realizzazione del controsoggetto per un dato soggetto. Le regole che lo compongono sono in gran parte prese dal contrappunto a due e sono riportate nella tabella seguente. In particolare : le prime quattro regole sono tipiche della costruzione melodica; le seguenti tre sono invece comuni a tutte le forme di contrappunto; la regola (viii) è quella che caratterizza un contrappunto di 1^ specie; le regole dalla (ix) alla (xiii) sono invece del contrappunto a due; la regola (xiv) caratterizza il contrappunto doppio; infine la regola (xv) è usata nell'ambito della creazione del controsoggetto (solitamente infatti sono concesse modulazioni, ma solo ai toni vicini; quando invece si crea un controsoggetto, come è già stato detto parlando dell'esposizione, bisogna restare sempre nella tonalità d'impianto). In realtà la regola (xiii) sarebbe inutile : essa può infatti essere dedotta dalle regole (i) e

[14] Sempre tenendo conto del fatto che scolasticamente la fuga è pensata come fuga corale e non strumentale.

[15] Si veda in appendice le tabelle con i soggetti e relativi controsoggetti di J.S.Bach (N.B.: tratti dal repertorio strumentale)

(x) . Per rispetto alla tradizione, essa viene però esplicitamente aggiunta, essendo inoltre una piccola ottimizzazione.

i) se nel contrappunto compare la sensibile, essa deve risolvere sulla tonica
ii) nel contrappunto una stessa nota può essere ribattuta al più due volte di seguito
iii) il contrappunto deve essere il più melodico possibile : pertanto si privilegeranno intervalli melodici piccoli (soprattutto seconde e terze, quando possibile)
iv) il contrappunto non può compiere intervalli melodici illeciti, neanche se composti
v) si deve sempre evitare qualunque combinazione producente la sensazione dell'accordo di quarta e sesta
vi) non sono tollerati incroci tra le voci
vii) sono preferibili i moti obliquo e contrario a quello retto
viii) ad ogni nota del canto dato ne deve corrispondere una del contrappunto
ix) l'unisono tra canto dato e contrappunto è consentito solo in corrispondenza della prima e dell'ultima nota del canto dato
x) canto dato e contrappunto non possono formare due ottave consecutive
xi) canto dato e contrappunto possono formare al più tre terze consecutive
xii) canto dato e contrappunto possono formare al più tre seste consecutive
xiii) se nel canto dato compare la sensibile, essa non può essere raddoppiata nel contrappunto
xiv) le uniche note che possono essere usate nel contrappunto sono l'unisono, l'ottava, la quinta, la terza e la sesta (maggiori o minori a seconda del modo della fuga) con la corrispondente nota del canto dato
xv) non sono accettate modulazioni

6.2 Grammatica di sincronizzazione per il controsoggetto

Per creare una grammatica di sincronizzazione dobbiamo anzitutto dare l'alfabeto degli eventi elementari N , cioè le note musicali. La notazione usata per rappresentarle è quella latina, ancora oggi usata nei paesi anglo–sassoni, che chiama le note musicali con le lettere dell'alfabeto [16] . Per distinguere le note con stesso nome ma appartenenti ad ottave diverse, viene concatenato al nome un numero indicante l'ottava di appartenenza [17] ; sempre per tradizione le ottave vanno dal do al si immediatamente superiore . Pertanto l'insieme di tutte le note musicali sarà

$$NOTE = \{ C_1 , D_1 , \dots , B_1 , C_2 , \dots , B_2 , \dots , C_5 , \dots , B_5 , C_6 \}$$

essendo 5 ottave sufficienti per comporre una fuga (in realtà per comporre una fuga corale ne bastano anche meno). L'insieme NOTE è totalmente ordinato con l'ordinamento usato nell'enumerazione appena fatta; più algoritmicamente tale ordinamento si ottiene con i seguenti due passi :

- $B \succ A \succ G \succ F \succ E \succ D \succ C$
  (ordinamento decrescente delle note di un'ottava)
- $\forall x , y \in \{ A , B , C , D , E , F , G \}$ $\forall n , m \in \{ 1 , 2 , 3 , 4 , 5 , 6 \}$ si ha che $x_n \succ y_m$ sse
  a) $n > m$, oppure
  b) $n = m$ AND $x \succ y$, oppure
  c) $n = m$ AND $x = y$

[16] Con la corrispondenza do ↔ C , re ↔ D , mi ↔ E , fa ↔ F , sol ↔ G , la ↔ A , si ↔ B

[17] Il la a 440 Hz (dato dal diapason) è convenzionalmente indicato LA_3 , e quindi nel nostro alfabeto sarà A_3

In questa fase del lavoro, si lavorerà con un sottoinsieme di NOTE : basterà infatti avere a disposizione le note racchiuse in due ottave, poiché il soggetto è tipicamente contenuto in un'ottava ed il controsoggetto non può distanziarsi da esso più di un'ottava, essendo il contrappunto usato quello doppio all'ottava. Pertanto, se il soggetto è $s = s_1 \dots s_n$ e la sua nota più bassa è $s_j = x_m$ allora avremo che l'insieme A di eventi elementari che useremo nella grammatica di sincronizzazione sarà

$$A = \{ t \in \text{NOTE} : x_{m+2} \blacktriangleright t \blacktriangleright x_m \}$$ [18]

Definiamo ora quattro funzioni che serviranno per costruire gli insiemi S_R per le varie note di A

- $\upsilon : A \to A$ t.c. $\upsilon(t) =$ l'unisono con t [19]
- $\tau_+ : A \to A$ t.c. $\tau(t) =$ la terza superiore a t
- $\sigma_+ : A \to A$ t.c. $\sigma(t) =$ la sesta superiore a t
- $o_+ : A \to A$ t.c. $o(t) =$ l'ottava superiore a t

Si ha quindi che

$$S_R(s_1) = \{ \upsilon(s_1), \tau_+(s_1), \sigma_+(s_1), o_+(s_1) \}$$
$$S_R(s_n) = \{ \upsilon(s_n), \tau_+(s_n), \sigma_+(s_n), o_+(s_n) \}$$
$$\forall i = 2, \dots, n-1 \quad S_R(s_i) = \{ \tau_+(s_i), \sigma_+(s_i), o_+(s_i) \}$$

Per completare la grammatica manca solo la formalizzazione delle regole del contrappunto doppio di 1^ specie all'ottava come regole markoviane di offset costante. Alcune di esse in realtà sono già state realizzate nei paragrafi

[18] N.B.: il controsoggetto verrà costruito come contrappunto superiore del soggetto; essendo scritto in contrappunto doppio sappiamo che esso ne è anche un contrappunto inferiore

[19] Cioè $\upsilon(t) = t$

precedenti e pertanto non corrisponderanno ad alcuna regola markoviana; in particolare :

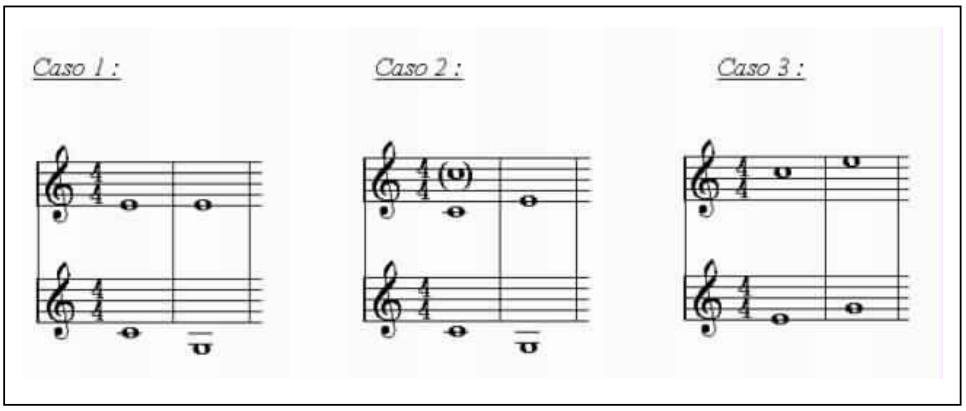
- la regola (vi) è realizzata ponendo il controsoggetto sempre al di sopra del soggetto
- la regola (viii) è realizzata dalla grammatica di sincronizzazione che per come è stata definita associa ad ogni evento elementare di *s* un evento elementare in *c*
- le regole (ix) e (xiv) sono state già realizzate nella definizione di S_R
- la regola (xv) si realizza non includendo in NOTE alcuna alterazione (pertanto si resterà sempre in do maggiore)
- la regola (v) è realizzata in parte non includendo la quarta in S_R (che esclude la possibilità di un secondo rivolto dell'accordo), in parte tramite le seguenti regole markoviane

$$r_{(v)}^{(1)}(u\,v, x\,y) = \begin{cases} \textbf{\textit{false}} & \text{se} \quad \text{è\_dominante}(v, u) \ \text{AND} \\ & \quad x = y = \tau_+(u) \\ \textbf{\textit{true}} & \text{altrimenti} \end{cases}$$

$$r_{(v)}^{(2)}(u\,v, x\,y) = \begin{cases} \textbf{\textit{false}} & \text{se} \quad \text{è\_dominante}(v, u) \ \text{AND} \\ & \quad x \in \{u, o_+(u)\} \ \text{AND} \\ & \quad y = \sigma_+(v) \\ \textbf{\textit{true}} & \text{altrimenti} \end{cases}$$

$$r_{(v)}^{(3)}(u\,v, x\,y) = \begin{cases} \textbf{\textit{false}} & \text{se} \quad \text{è\_dominante}(v, x) \ \text{AND} \\ & \quad x = \sigma_+(u) \ \text{AND} \ y = \sigma_+(v) \\ \textbf{\textit{true}} & \text{altrimenti} \end{cases}$$

che escludono una sensazione di accordo di quarta e sesta che si verrebbe a creare nelle seguenti situazioni rispettivamente



Inoltre bisogna includere anche altre tre regole per considerare la situazione al rivolto (cioè la situazione in cui i ruoli di u , v e di x , y siano invertiti nelle regole appena esposte). Eccone la formulazione esplicita

$$r_{(v)}^{(4)}(u\,v\,,x\,y) = \begin{cases} \textbf{\textit{false}} & \text{se } \text{è\_dominante}(y,x) \text{ AND } u = v = \sigma_{-}(u) \\ \textbf{\textit{true}} & \text{altrimenti} \end{cases}$$

$$r_{(v)}^{(5)}(u\,v\,,x\,y) = \begin{cases} \textbf{\textit{false}} & \text{se } \text{è\_dominante}(y,x) \text{ AND } u \in \{x, o_{-}(x)\} \text{ AND } v = \tau_{-}(y) \\ \textbf{\textit{true}} & \text{altrimenti} \end{cases}$$

$$r_{(v)}^{(6)}(u\,v\,,x\,y) = \begin{cases} \textbf{\textit{false}} & \text{se } \text{è\_dominante}(y,u) \text{ AND } x = \tau_{+}(u) \text{ AND } y = \tau_{+}(v) \\ \textbf{\textit{true}} & \text{altrimenti} \end{cases}$$

- la regola (i) è realizzata con la seguente

$$r_{(i)}(u\,v\,,\,x\,y) = \begin{cases} \textbf{\textit{false}} & \text{se} \quad x = \text{`}B_n\text{'} \ \text{AND} \\ & \quad\ y \neq \text{`}C_{n+1}\text{'} \ \text{ con } n \in \{1, \ldots, 5\} \\ \textbf{\textit{true}} & \text{altrimenti} \end{cases}$$

cioè, se x è una sensibile (siamo in do maggiore, quindi la sensibile è un si della generica ottava n) allora y deve essere una tonica (cioè il do dell'ottava successiva)

- la regola (ii) è data dalle seguente

$$r_{(ii)}(u\,v\,w\,,\,x\,y\,z) = \begin{cases} \textbf{\textit{false}} & \text{se} \quad x = y = z \\ \textbf{\textit{true}} & \text{altrimenti} \end{cases}$$

- la regola (x) si ottiene con

$$r_{(x)}(u\,v\,,\,x\,y) = \begin{cases} \textbf{\textit{false}} & \text{se} \quad x = o_+(u) \ \text{AND} \ y = o_+(v) \\ \textbf{\textit{true}} & \text{altrimenti} \end{cases}$$

- la regola (xi) è data da

$$r_{(xi)}(u\,v\,w\,,\,x\,y\,z) = \begin{cases} \textbf{\textit{false}} & \text{se} \quad x = \tau_+(u) \ \text{AND} \ y = \tau_+(v) \\ & \quad\ \text{AND} \ z = \tau_+(w) \\ \textbf{\textit{true}} & \text{altrimenti} \end{cases}$$

ed analoga è la regola (xii) con σ_+ al posto di τ_+

- la regola (xiii) si ottiene con

$$r_{(xiii)}(u\,,\,x) = \begin{cases} \textbf{\textit{false}} & \text{se} \quad u = \text{`}B_n\text{'} \ \text{AND} \\ & \quad\ x = \text{`}B_{n+1}\text{'} \ \text{ con } n \in \{1, \ldots, 5\} \\ \textbf{\textit{true}} & \text{altrimenti} \end{cases}$$

- le regole (iii) e (vii) non sono regole coercitive : sono semplicemente regole preferenziali. Possono quindi essere tralasciate (cioè non essere

incluse in R) e, in un modello un po' più evoluto, possono essere usate in fase di scelta (cioè, se dopo aver eliminato da S_R gli eventi illeciti con le regole di R , l'insieme risultante ha più di un elemento si possono usare tali regole come criterio di scelta [20]).

- la regola più complessa e che fa salire l'offset delle regole da 3 (come è stato finora) a 7 è la (iv) . Eccone la formulazione esplicita

$$r_{(iv)}(u_1 \dots u_7 , x_1 \dots x_7) = \begin{cases} \textbf{\textit{false}} & \text{se } \exists j = 2, \dots, 7 \ \exists k = 1, \dots, j-1 \\ & \text{tali che} \\ & \quad \text{stesso\_moto}(< x_k , x_{k+1} >, \\ & \qquad\qquad < x_{k+1} , x_{k+2} >, \dots, \\ & \qquad\qquad < x_{j-1} , x_j >) \\ & \text{AND} \\ & \quad \text{NOT è\_lecito}(\text{intervallo}(x_k , x_j)) \\ \textbf{\textit{true}} & \text{altrimenti} \end{cases}$$

Essa vuol dire che, se dalla nota x_k a x_j si procede sempre per moto ascendente o discendente e se l'intervallo formato da tali note è illecito, allora la regola dichiara un errore. Basta fermarsi a stringhe di 7 note consecutive; infatti il seguente è il caso peggiore :

$$C_3 – D_3 – E_3 – F_3 – G_3 – A_3$$

se la nota successiva è x_n con $A_3 \blacktriangleright x_n$ allora non ci sono problemi poiché il moto si interrompe; altrimenti qualunque y_m (con $y_m \blacktriangleright A_3$) si pone di seguito formerà un intervallo illecito (per esempio B_3 formerà una

[20] Rifacendoci allo pseudo-codice dell'algoritmo *genera_carattere* del paragrafo 3.6 , la funzione *scegli-carattere* potrebbe essere realizzata proprio con queste due regole, invece che con una semplice scelta casuale (in realtà è proprio ciò che facciamo nel sistema implemetato SEBASTIAN che presenteremo nel capitolo 8) .

settima con C_3 e una quarta aumentata con F_3, C_4 una settima con D_3, etc.) [21].

Quindi l'insieme di regole markoviane di offset 7 che la SG per il controsoggetto adotterà è

$$R = \{ r_{(i)} , r_{(ii)} , r_{(iv)} , r_{(v)}^{(1)}, r_{(v)}^{(2)}, r_{(v)}^{(3)}, r_{(v)}^{(4)}, r_{(v)}^{(5)}, r_{(v)}^{(6)}, r_{(x)} , r_{(xi)} , r_{(xii)} , r_{(xiii)} \}$$

Gli elementi costituenti la SG sono ora totalmente definiti; la grammatica può quindi essere creata come in precedenza descritto.

[21] Il caso citato è il peggiore perché è quello in cui ad ogni passo si sale di un tono alla volta : salendo di più toni alla volta si raggiungerebbe con al più sei note il limite dell'ottava che costituisce il più ampio intervallo che una voce può compiere (quindi la settima nota per essere lecita deve comunque cambiare moto).

Capitolo 7 : La generazione delle parti libere

Fino ad ora la nostra trattazione si è occupata di generare un possibile controsoggetto per un soggetto dato; fatta questa operazione, il passo successivo consiste nel decidere una possibile organizzazione del materiale tematico per dar vita alla nostra esposizione. Ciò può esser fatto in quattro modi diversi, come presentato nei richiami di teoria musicale nel paragrafo 5.2: chiaramente sceglierne uno in maniera casuale è compito piuttosto semplice per un elaboratore. Fatta tale scelta, nella griglia rappresentante l'esposizione sono state sistemate tutte le entrate del soggetto (la prima e la terza in tonalità di impianto, la seconda e la quarta nella relativa dominante [22]) con i relativi controsoggetti (anche qui il secondo e il quarto in tonalità di dominante, il terzo in tonalità d'impianto). Stabiliti quindi questi cardini, restano da riempire i vuoti restanti, cioè creare le parti libere; è questo il compito del presente capitolo.

7.1 Regole del contrappunto a tre e a quattro di prima specie

In analogia con il contrappunto a due, il contrappunto a tre e a quattro è così chiamato perché coinvolge rispettivamente tre e quattro voci; inoltre è

[22] E' da notare che, avendo supposto di lavorare solo con soggetti reali, il passaggio alla relativa dominante risulta semplice : basta trasportare soggetto e controsoggetto una quinta sopra (o analogamente una quarta sotto)

mantenuta anche qui la limitazione di parti ad andamento omoritmico (1^ specie). Cambia invece il fatto che per creare le parti libere non è necessario ricorrere al contrappunto triplo e quadruplo, non essendo richiesta la rivoltabilità delle parti [23] . La quasi totalità delle regole che sovrintendono alla scrittura di tali melodie sono le stesse di quelle esaminate nel caso del contrappunto doppio, tranne per il fatto che bisogna tenere conto della presenza di due o tre stringhe date, invece che di una sola . Prima di passare all'esposizione delle regole e della relativa GSG , ci sembra importante ribadire l'importanza del fatto che la rivoltabilità qui non è presente : in fase di generazione del controsoggetto, la situazione si presentava abbastanza semplice poiché decidevamo di creare il controsoggetto come contrappunto superiore al soggetto, sapendo che, grazie alla rivoltabilità, ciò non avrebbe fatto perdere la generalità del nostro approccio. Qui la situazione è diversa : poiché non siamo in regime di rivoltabilità, è fondamentale sapere l'esatta disposizione delle voci che tengono le varie parti e questo creerà qualche problema nella determinazione dell'insieme $S_R(\underline{a})$. Il problema sta nel fatto che per il controsoggetto l'insieme $S_R(a)$ era formato solo dall'unisono, la terza, la sesta e l'ottava superiori ad a ; ora ciò è più complesso poiché la parte libera può essere più alta di una voce data ma più bassa di un'altra. Le regole che modellano questa situazione sono le seguenti :

[23] Questo è anche abbastanza intuitivo poiché, non avendo queste parti una forte valenza tematica (essendo la prosecuzione di un discorso già iniziato), esse saranno transitorie e da non ripetere in altre voci (altrimenti assumerebbero un'importanza che non appartiene loro). Al contrario, come già detto, soggetto e controsoggetto sono i cardini dell'intera composizione e quindi deve essere loro garantita la possibilità di apparire in qualunque combinazione di voci possibile, e ciò può esser fatto solo tramite la rivoltabilità delle parti

(I) l'i–esima nota della parte libera deve formare una consonanza con l' i–esima nota di tutte le parti date

(II) deve essere comunque evitata qualsiasi sensazione di un accordo di quarta e sesta

(III) non sono tollerati incroci tra le voci

(IV) . se la parte libera è superiore a tutte le parti date, essa non potrà scostarsi dalla più alta di queste per più di un'ottava.

. se la parte libera è inferiore a tutte le parti date, essa non potrà scostarsi dalla più bassa di queste per più di un'ottava.

(V) non sono accettate modulazioni

Queste cinque regole determinano l'insieme $S_R(\underline{a})$ contenente le note che possono essere emesse assieme ad $\underline{a}$ in una parte libera tenuta da una voce compresa tra la voce superiore V_s e la voce inferiore V_i [24].

Restano inoltre molte delle regole già presentate per il contrappunto doppio, qui lievemente modificate (in alcuni casi).

(VI) se nella parte libera compare la sensibile essa deve risolvere salendo alla tonica

(VII) nella parte libera una stessa nota può essere ribattuta al più due volte di seguito

[24] Come già detto, qui è fondamentale l'esatta posizione della parte libera che si sta creando : a tale scopo vengono dati gli indici delle voci immediatamente superiori ed inferiori. Quindi la sequenza verticale delle voci dalla più bassa alla più alta sarà

$$< V_1 , \dots , V_i , \text{parte libera} , V_S , \dots , V_{NUM\_VOCI} >$$

Può essere che non ci sia la voce superiore (o inferiore) : in tal caso $s = NUM\_VOCI + 1$ (oppure $i = -1$)

(VIII) la parte libera deve essere il più melodica possibile : pertanto si privilegeranno intervalli melodici piccoli (soprattutto seconde e terze, quando possibile)

(IX) la parte libera non può compiere intervalli melodici illeciti, neanche se composti

(X) ad ogni nota delle parti date ne deve corrispondere una della parte libera

(XI) la parte libera non può formare due ottave consecutive con nessuna delle parti date

(XII) la parte libera non può formare due quinte consecutive con nessuna delle parti date

(XIII) la parte libera può formare al più tre terze consecutive con ogni parte data

(XIV) la parte libera può formare al più tre seste consecutive con ogni parte data

(XV) se in una delle parti date compare la sensibile, essa non può essere raddoppiata nella parte libera

Infine si introducono tre importanti regole proprie del contrappunto a più di due voci

(XVI) non si può arrivare in unisono per moto retto

(XVII) non si può arrivare su una consonanza perfetta per moto retto tra parti estreme [25]

(XVIII) in una triade non si può raddoppiare né triplicare la terza

[25] Per parti estreme si intende la voce più alta e più bassa che in quel momento sono in azione

7.2 Grammatica di sincronizzazione generalizzata per le parti libere

Iniziamo con la creazione dell'insieme $S_R(\underline{a})$ per dar vita ad una parte libera racchiusa tra le voci V_i e V_s (sempre con la convenzione che i = –1 se la parte libera è tenuta dalla voce più bassa, e s = NUM_VOCI + 1 se è tenuta dalla più alta); per far ciò serve un algoritmo complesso, al contrario della semplicità richiesta nel caso del controsoggetto.

L'algoritmo esamina tutte le note date (tutte le componenti del vettore $\underline{a}$) [righe 1 e 8]; per la generica a_j (con $j \in \{ 1, \dots, i, s, \dots, NUM\_VOCI \}$) considera ogni nota n che può essere eseguita contemporaneamente ad a_j (cioè l'unisono, la terza, la quarta, la quinta, la sesta e l'ottava superiori ad a_j per $j \le i$ – date rispettivamente dalle funzioni υ , τ_+ , q_+ , Q_+ , σ_+, o_+ – oppure l'unisono, la terza, la quarta, la quinta, la sesta e l'ottava inferiori ad a_j per $j \ge s$ – date rispettivamente dalle funzioni υ , τ_- , q_- , Q_- , σ_- , o_-) [26] [righe 3 e 10]; dopodiché le porta nell'ottava corretta (cioè quella superiore ad a_i per $j \le i$ o a quella inferiore ad a_s per $j \ge s$) [righe 5, 6 e 12, 13] e le inserisce nell'insieme I_j contenente tutte le note eseguibili assieme ad a_j poste nella corretta ottava [righe 7 e 16] . Alla fine restituisce l'intersezione di tutti questi insiemi I_j così costruiti (che appunto conterrà tutte le note eseguibili assieme alla note di $\underline{a}$ poste nelle loro posizioni corrette).

[26] In questo caso, contrariamente al contrappunto doppio, è lecito porre una quinta (inferiore o superiore) nella parte libera : ciò è dovuto al fatto che, non essendo in regime di rivoltabilità, non si presenterà mai la situazione di una quinta rivoltata (cioè una quarta) che produce un'armonia di quarta e sesta. E' inoltre possibile mettere anche una quarta, purché non si produca un'armonia di quarta e sesta; di ciò tra breve si discuterà.

Crea_sincronizzabili (a , i , s)

1) for j = 1 to i
2) $I_j = \varnothing$
3) $\forall$ n $\in$ { $\upsilon(a_j)$, $\tau_+(a_j)$, $q_+(a_j)$, $Q_+(a_j)$, $\sigma_+(a_j)$, $o_+(a_j)$ }
4) sia l il nome ed o l'ottava della nota n (cioè n = ' l_o ')
5) k = o
6) while a_i ➤ ' l_k ' do k++
7) if intervallo (a_1 , ' l_k ') ≠ QUARTA then $I_j = I_j \cup$ { ' l_k ' }

8) for j = s to NUM_VOCI
9) $I_j = \varnothing$
10) $\forall$ n $\in$ { $\upsilon(a_j)$, $\tau_-(a_j)$, $q_-(a_j)$, $Q_-(a_j)$, $\sigma_-(a_j)$, $o_-(a_j)$ }
11) sia l il nome ed o l'ottava della nota n (cioè n = ' l_o ')
12) k = o
13) while ' l_k ' ➤ a_s do k – –
14) if ((i > 0 AND intervallo (a_1 , ' l_k ') ≠ QUARTA) OR
15) (i == –1 AND intervallo (' l_k ' , a_j) ≠ QUARTA))
16) then $I_j = I_j \cup$ { ' l_k ' }

17) return $\cap_{j = 1, \ldots, m} I_j$

Un discorso un po' delicato va fatto per motivare la condizione posta alle [righe 7 e 14,15] : essa serve a realizzare la regola (II) . La nota ' l_k ' può essere inserita nell'insieme I_j solo se non forma una quarta con la nota più

bassa eseguita in quel momento da tutte le voci, inclusa quella che terrà la parte libera. Possono presentarsi due situazioni :

– la nota più bassa è tenuta dalla prima voce (cioè V_1 è la voce più bassa e quindi $i \geq 1$) : in tal caso ([righe 7 e 14]) si deve verificare che l'intervallo formato da a_1 e ' l_k ' non sia una quarta [27] . Solo in tal caso si potrà inserire ' l_k ' in I_j

– la nota più bassa sarà tenuta dalla voce che esegue la parte libera che si sta creando ($i = -1$) : anzitutto vale la pena osservare che questo caso non è considerato nella condizione di [riga 7] : ciò è dovuto al fatto che il ciclo di [riga 1] non viene neanche avviato nel caso in cui $i = -1$ e quindi in tal caso la condizione di [riga 7] non sarà mai valutata. E' invece possibile che la nota che si sta considerando (' l_k ') formi con a_j (per $j \geq s$) un intervallo di quarta (il test a [riga 15]) : in tal caso si formerebbe un accordo di quarta e sesta e perciò la nota ' l_k ' non viene inclusa nell'insieme I_j

La regola (II) è quindi realizzata dalle condizioni poste alle [righe 7 e 14,15] ; la regola (I) è realizzata considerando a [righe 3 e 10] solo le consonanze perfette (unisono, quarta, quinta, ottava) ed imperfette (terza e sesta) ; la regola (III) è realizzata intersecando a [riga 17] tutti gli insiemi I_j ottenuti (cioè la nota k–esima della parte libera sarà al di sopra di tutte le a_j

[27] N.B.: non basta verificare a [riga 7] che ' l k ' $\neq q_+(a_1)$ (ed analogamente con q_- a [riga 14,15]) . Questo poiché anche gli intervalli di 11^ o di 18^ (cioè delle quarte a cui sono state aggiunte una o due ottave) produrrebbero la stessa sensazione di quarta e sesta. Per questo motivo nell'algoritmo si è introdotta la costante QUARTA che raccoglie tutti gli intervalli ottenuti aggiungendo ad una quarta nessuna, una o più ottave.

per $j \leq i$ ed al di sotto di tutte le a_j per $j \geq s$, cioè la voce che tiene la parte libera che si sta creando non incrocerà nessun'altra voce) ; la regola (IV) viene realizzata considerando come massimo intervallo l'ottava inferiore alla voce più bassa (nel caso $i = -1$, [riga 10] , iterazione per $j = 1$) oppure l'ottava superiore alla voce più alta (nel caso $s = NUM\_VOCI+1$, [riga 3] , iterazione per $j = 1$); infine la regola (V) si realizza facendo in modo che le funzioni υ , τ_+ , q_+ , Q_+ , σ_+, o_+ , τ_- , q_- , Q_- , σ_- , o_- restituiscano note appartenenti alla tonalità d'impianto (nel nostro caso il do maggiore).

Bisogna ora esprimere le restanti regole del contrappunto a tre e a quattro come regole markoviane; per quanto riguarda quelle appartenenti già al contrappunto doppio (e quindi già formalizzate nel capitolo precedente) usiamo l' OSS.1 del capitolo 4 . Esse saranno pertanto così espresse :

- $r_{(VI)} (\underline{u}\ \underline{v} , x\ y) = r_{(i)} (u_1\ v_1 , x\ y)$
- $r_{(VII)} (\underline{u}\ \underline{v}\ \underline{w} , x\ y\ z) = r_{(ii)} (u_1\ v_1\ w_1 , x\ y\ z)$
- $r_{(IX)} (\underline{u}_1 \dots \underline{u}_7 , x_1 \dots x_7) = r_{(iv)} (u_{11} \dots u_{71} , x_1 \dots x_7)$
- $r_{(XI)} (\underline{u}\ \underline{v} , x\ y) = \bigwedge_{i = 1 , \dots , m} r_{(x)} (u_i\ v_i , x\ y)$
- $r_{(XIII)} (\underline{u}\ \underline{v}\ \underline{w} , x\ y\ z) = \bigwedge_{i = 1 , \dots , m} r_{(xi)} (u_i\ v_i\ w_i , x\ y\ z)$
- $r_{(XIV)} (\underline{u}\ \underline{v}\ \underline{w} , x\ y\ z) = \bigwedge_{i = 1 , \dots , m} r_{(xii)} (u_i\ v_i\ w_i , x\ y\ z)$
- $r_{(XV)} (\underline{u} , x) = \bigwedge_{i = 1 , \dots , m} r_{(xiii)} (u_i , x)$

E' importante spiegare il modo in cui sono state definite $r_{(VI)}$, $r_{(VII)}$ e $r_{(IX)}$. Esse sono state definite usando regole corrispondenti del contrappunto doppio, con la particolarità rispetto alle altre che si è considerata solo la prima

componente (cioè la prima voce) delle parti date : ciò è dovuto al fatto che le regole (VI) , (VII) e (IX) riguardano solo l'andamento melodico della parte libera che si sta costruendo. Andando a vedere infatti la realizzazione di $r_{(i)}$, $r_{(ii)}$ e $r_{(iv)}$ si nota che i primi elementi (quelli cioè del canto dato) non servono a nulla : essi sono tenuti solo per omologare le regole che si stanno creando alla definizione di regola markoviana. Quindi in questa sede si seleziona una voce qualsiasi (abbiamo scelto la prima ma potevamo sceglierne una qualunque) tanto l'importante è solo la linea melodica della parte libera che si sta creando.

Non possiamo però dichiarare completamente realizzata ancora la regola (II) : infatti, come nel contrappunto doppio, bisognerà evitare non solo le armonizzazioni di 4^ e 6^ ma anche le sensazioni di tale rivolto (si vedano gli esempi per la regola (v) nel paragrafo 6.2). In quel caso avevamo introdotto le regole $r_{(v)}^{(1)}$, $r_{(v)}^{(2)}$, $r_{(v)}^{(3)}$, $r_{(v)}^{(4)}$, $r_{(v)}^{(5)}$, $r_{(v)}^{(6)}$. In questo contesto le ultime tre non si useranno, non trovandoci in regime di rivoltabilità. Le prime tre verranno invece applicate solo tra la parte libera e la voce più bassa, se questa è diversa dalla voce che tiene la parte libera in questione; verranno invece applicate alla parte libera e tutte le altre voci se la voce più bassa è quella che eseguirà la parte libera.

Infine, analogamente con il caso del contrappunto doppio :

- la regola (X) è realizzata automaticamente per come è stata definita la GSG (vedi la regola (viii) del capitolo precedente)
- la regola (VIII) è anche qui utilizzabile come funzione di scelta al momento in cui l'insieme $S_R(\underline{a})$ contiene più elementi (vedi le regole (iii) e (vii))

Manca solo la definizione della regola proprie del contrappunto a più voci. Iniziamo con la regola (XII) : per fare ciò introduciamo una regola markoviana di offset 2 per poi usarla come nei casi sopra portati. E' importante notare che questa regola non era presente nel contrappunto doppio poiché lì, essendo in regime di rivoltabilità, non era possibile sovrapporre una quinta ad una nota [28] e quindi non si poteva creare la situazione di quinte parallele.

$$r_{Q}(u\,v, x\,y) = \begin{cases} \textbf{\textit{false}} & \text{se } x = Q_{+}(u) \text{ AND } y = Q_{+}(v) \\ \textbf{\textit{true}} & \text{altrimenti} \end{cases}$$

da cui

$$r_{(XII)}(\underline{u}\,\underline{v}, x\,y) = \bigwedge_{i=1,\dots,m} r_{Q}(u_i\,v_i, x\,y)$$

Similmente, anche per le regole (XVI) e (XVII) introduciamo una regola che escluda il caso che due voci arrivino per moto retto su di un unisono e su di una consonanza perfetta rispettivamente. Vale la pena notare anche qui che tali regole non erano presenti nel caso delle due voci poiché le consonanze perfette producono un risultato armonico debole e quindi nel caso di sole due voci sono state trascurate (le quinte anche per il motivo espresso nel discutere la realizzazione della regola (XII) ; le ottave e l'unisono erano state escluse tranne che all'inizio del soggetto : non si pone pertanto il problema di come arrivarci, essendo ogni voce in pausa prima del suo ingresso). Le regole sono

$$r_{U}(u\,v, x\,y) = \begin{cases} \textbf{\textit{false}} & \text{se } y = v \text{ AND } \text{moto}(u, v) = \text{moto}(x, y) \\ \textbf{\textit{true}} & \text{altrimenti} \end{cases}$$

[28] Una quinta rivoltata dà una quarta che genera, se posta al basso, uno stato di secondo rivolto, illecito come stabilito dalla regola (II)

$$r_C(u\,v, x\,y) = \begin{cases} \textbf{\textit{false}} & \text{se intervallo}(v,y) \in \text{Consonanze\_Perfette} \\ & \quad \text{AND moto}(u, v) = \text{moto}(x, y) \\ \textbf{\textit{true}} & \text{altrimenti} \end{cases}$$

da cui

$$r_{(XVI)}(\underline{u}\,\underline{v}, x\,y) = \bigwedge_{i=1,\dots,m} r_U(u_i\,v_i, x\,y)$$

$$r_{(XVII)}(\underline{u}\,\underline{v}, x\,y) = \textit{if}\ (\text{voce\_più\_bassa} == \text{parte\_libera})\ \textit{then}\ r_C(x\,y, u_{max}\,v_{max})\ \textit{else if}\ (\text{voce\_più\_alta} == \text{parte\_libera})\ \textit{then}\ r_C(u_1\,v_1, x\,y)\ \textit{else}\ \textbf{\textit{true}}$$ [29]

Vale la pena notare che la formulazione di $r_{(XVII)}$ è stata data usando l'*if – then – else* che non appartiene ai costrutti fondamentali della logica; in realtà però esso può facilmente essere realizzato tramite l'implicazione [30] . Per una maggiore leggibilità abbiamo però preferito questa notazione.

Resta infine la definizione della regola (XVIII) : essa è l'unica che viene data direttamente per più voci (ciò è dovuto al fatto che è l'unica a parlare esplicitamente di accordi). Essa stabilisce che la terza di una triade non può essere presentata da più di una voce : questo è motivato dal fatto che essa è la nota più debole dell'accordo e quindi non può essere troppo enfatizzata. La

[29] Cioè se la parte libera non è né la più alta né la più bassa in quel momento, allora la regola non ha nessuna influenza e può quindi considerarsi verificata.

[30] L'espressione *if* C *then* A_1 *else* A_2 è equivalente a $(C \Rightarrow A_1) \wedge (\neg C \Rightarrow A_2)$ (dove A1 e A2 sono espressioni che restituiscono un valore booleano)

cosa più complessa in questa regola è la necessità di identificare la triade in questione : una volta fatto ciò basta escludere la terza se questa è già presente. Per fare ciò riportiamo le seguenti osservazioni; per una maggiore leggibilità, omettiamo anche in questo caso una precisa formulazione logica sapendo che esse sono formalizzabili tramite implicazioni.

Partiamo anzitutto dal caso più semplice, cioè quello in cui abbiamo tre note a disposizione (cioè quando creiamo l'ultima parte libera, avendo quindi le parti delle altre tre voci [31]) . Si possono presentare tre casi :

1) *le tre note sono uguali (a meno dell'ottava di appartenenza) :*

   Sia $N \in \{A,B,C,D,E,F,G\}$ la nota triplicata. Poiché si può triplicare solo la fondamentale dell'accordo, avremo un accordo di "N maggiore" o "N minore" a seconda del ruolo di N nella tonalità di impianto [32] .

2) *le tre note sono tutte diverse (a parte l'ottava di appartenenza) :*

   La triade allora è completa; per determinare la fondamentale basta identificare tra quali note c'è un intervallo di quinta e restituire la nota inferiore di tale intervallo [33] .

3) *due note sono uguali e una è diversa (a parte l'ottava di appartenenza) :*

   Se riusciamo ad identificare una quinta possiamo dichiarare risolto il problema (come al punto precedente). Altrimenti bisogna distinguere due sottocasi :

---

[31] Per meglio capire la situazione, si consideri nel prossimo capitolo la procedura `faiPartiLibere`

[32] In do maggiore C , F e G saranno accordi maggiori, mentre D , E e A saranno minori. (N.B.: B non può essere armonizzato allo stato fondamentale poiché non sono accettate triadi diminuite)

[33] Se abbiamo ad esempio C – E – A vediamo che l'intervallo A / E è una quinta e affermiamo (giustamente) che si tratta di un accordo di A (minore per la nota precedente)

(i) *la nota diversa è la terza delle due note uguali :* in tal caso la fondamentale è la nota uguale [34]

(ii) *le note uguali sono la terza della nota diversa :* in tal caso la fondamentale è la quinta sotto le note uguali [35]

Il caso in cui abbiamo solo due note date è più delicato poiché non tutte le situazioni possono essere risolte (al contrario del caso precedente). Si possono comunque fare alcune osservazioni :

a) *<u>se le due note sono uguali (a meno dell'ottava di appartenenza) :</u>*
Sicuramente la fondamentale dell'accordo non può essere la terza inferiore a tali note (altrimenti il terzo grado sarebbe già ribattuto) [36]

b) *<u>se le due note sono diverse (a parte l'ottava di appartenenza) :</u>*
Se formano una quinta allora l'accordo è identificato come nel punto (2) .
Se invece formano una terza l'unica cosa che si può dire è che, se la terza sta nella voce più bassa che in quel momento non è in pausa, allora l'accordo ha come fondamentale la nota che non è la terza [37] .

Nel primo caso l'accordo è identificato e quindi si può facilmente vedere se la terza è presente; in caso affermativo si impedisce alla parte libera che si sta costruendo di raddoppiarla, altrimenti non si impone alcun vincolo. Nel secondo caso, se si riesce ad identificare l'accordo si procede in maniera

[34] Se abbiamo ad esempio C – E – C siamo in C (maggiore). Non possiamo essere in A (minore) altrimenti la terza (C) sarebbe raddoppiata.

[35] Se abbiamo ad esempio C – E – E allora siamo in A (minore). Se infatti fossimo in C (maggiore) avremmo la terza (E) raddoppiata, cosa non concessa.

[36] Se abbiamo C – C non possiamo essere in A (essendo C la terza della triade di A)

[37] Se abbiamo E – C e il E è alla voce più bassa allora non possiamo essere in A (altrimenti avremmo una armonizzazione di quarta e sesta) e quindi possiamo essere solo in C .

identica, altrimenti non si impone alcun vincolo, rinviando eventualmente ad un momento successivo (cioè all'aggiunta dell'ultima voce) il compito di riconoscere l'accordo e decidere di conseguenza.

In conclusione vale la pena osservare che la regola (XVII) è stata esposta nella sua formulazione "debole" : infatti in un contesto rigoroso come è quello della fuga le consonanze perfette non sono raggiungibili per moto retto tra qualunque coppia di voci (non solo le estreme). Ciò è dovuto al fatto che la formulazione applicata è comunque accettabile in prima istanza : usare la versione "forte" avrebbe troppo complicato il meccanismo di generazione delle parti libere che andremo tra breve ad esporre [38] .

[38] Il problema sta nel fatto che con la versione "forte" è facile che si presenti la necessità di backtrack non solo all'interno della stessa voce ma anche tra voci diverse (modificare parti libere già create o addirittura il controsoggetto). In questa prima formulazione, le complicazioni implementative sarebbero state troppe; inoltre la regola può essere accettata anche nella formulazione debole, e quindi non si pone una grande restrizione al sistema.

Capitolo 8 : Un sistema implementato per la creazione dell'esposizione

In conclusione riportiamo tutti i passi necessari per risolvere il problema posto inizialmente : dato un soggetto omoritmico (in semibrevi) creare con esso l'esposizione di una fuga reale con contrappunto di 1^ specie. Il tutto è stato implementato in C++ dando vita ad un sistema chiamato *SEBASTIAN* (in onore del sommo musicista barocco Johann Sebastian Bach). Riportiamo in questo capitolo alcune scelte implementative fatte. Alcuni algoritmi sono già stati presentati nel paragrafo 3.6 (erano quelli tipici di ogni implementazione di una SG) ; riportiamo ora gli aspetti più specificamente musicali.

Strutture dati

Vediamo anzitutto come sono state codificate le note. Abbiamo deciso di leggerle dall'utente con la notazione alfabetica perché così ogni nota aveva come nome un singolo carattere[39], seguito da un carattere indicante un eventuale alterazione cromatica ed infine un numero indicante l'ottava di appartenenza. Si ha quindi

[39] Un problema con la denominazione italiana era la presenza del sol che con i suoi tre caratteri si differenzia da tutte le altre note, il cui nome è di due caratteri : questo avrebbe creato alcuni problemi (semplicemente risolvibili) di lettura. Inoltre la gestione di un singolo carattere è molto più leggera della gestione una stringa di caratteri.

```
class  Nota  {
     char nome ;
     char alterazione ;
     int ottava ;
```

con vari metodi per la gestione di tali parametri e per la conversione di formati. Infatti questa codifica delle note è usata solo per comunicare con l'utente : il sistema lavora sempre con una codifica numerica delle note. Si realizza pertanto la seguente traduzione

$$
\begin{aligned}
C_1 &\leftrightarrow 0 \\
C\#_1 &\leftrightarrow 1 \\
&\dots \\
B_1 &\leftrightarrow 11 \\
C_2 &\leftrightarrow 12 \\
&\dots
\end{aligned}
$$

e si lavora quindi solo con numeri per riconvertire (solo alla fine) i risultati ottenuti.

Si è poi realizzata una classe per la gestione di melodie (cioè sequenze di note) : con essa si memorizzeranno soggetto, controsoggetto e le linee di ognuna delle quattro voci. La realizzazione di tale struttura dati è quella di un semplice array di interi (le note della melodia), con in aggiunta un intero che ne contiene la lunghezza

```
class  NoteArray  {
     int * vettore ;
     int lung ;
```

Sempre istanziando questa classe si ottengono gli oggetti che contengono le possibili note da accoppiare al soggetto (al momento della creazione del controsoggetto), alla coppia soggetto/controsoggetto (al momento della creazione della prima parte libera) ed alla tripla soggetto/controsoggetto/prima_parte_libera (al momento della creazione della seconda parte libera). La classe risultante è la seguente

```
class  Possibilita  {
        NoteArray * vettore ;
        int lung ;
```

in cui ad ogni nota (o coppia di note o terna di note rispettivamente) da sincronizzare corrisponde un insieme di possibili note sincronizzabili contenute nel `NoteArray` associato a quella posizione. Le procedure che realizzano le regole contrappuntistiche prevederanno poi ad eliminare tra queste, tutte quelle che creano situazioni sgradevoli se eseguite dopo la nota scelta per la posizione precedente.

C'è infine la classe principale `Sebastian` che realizza tutta l'esposizione; riportiamo di seguito i suoi campi e la procedura principale. In essa

- `sogg` e `csogg` sono lunghi n (dove n denota il numero di note nel soggetto)
- `sopr` , `contr` , `ten` e `bass` sono lunghi 4·n poiché il soggetto viene esposto quattro volte (una per ogni voce)

- `modelloEntrate` è una variabile intera che conterrà il tipo di esposizione
- `entrate` è un array in cui la locazione i conterrà un puntatore alla struttura che descrive la voce entrata per i–esima nell'esposizione

```
class  Sebastian  {

     NoteArray sogg , csogg ,
               sopr , contr , ten , bass ;
     int modelloEntrate;
     NoteArray * entrate[4];

     public :
          sebastian() {
               leggiSogg();
               codificaSogg();
               faiControSogg();
               faiEsposizione();
               faiPartiLibere();
               decodifica();
          }
          void main() {
               new sebastian();
          }
}
```

Procedure

Riferendoci alla classe principale appena presentata, le procedure `leggiSogg()` , `codificaSogg()` e `decodifica()` sono state velocemente descritte in precedenza e servono solo per far comunicare l'utente e la macchina nei formati ad ognuno più congeniali.

La procedura `faiControSogg()` realizza il funzionamento di una grammatica di sincronizzazione ed è già stata presentata nel paragrafo 3.6 . A ciò sono state aggiunte alcune euristiche di scelta nel caso in cui l'insieme delle possibilità per una certa nota contenga più elementi. In sintesi esse sono :

1) privilegia i moti obliquo e contrario rispetto a quello retto
2) prepara la situazione presente in modo da poter realizzare ciò anche al passo successivo (cioè se siamo al passo i , e l'intervallo compiuto dal soggetto dalla nota i alla i+1 è ascendente, allora il controsoggetto in posizione i dovrà essere il più lontano possibile dal soggetto cosicché scenderà al passo i+1 realizzando un moto contrario col soggetto e viceversa)
3) cerca di far compiere al controsoggetto gli intervalli melodici più piccoli

Inizialmente cercherà la nota che soddisfa tutte e tre queste condizioni; se non esiste (caso piuttosto frequente) segue il seguente ordine :

- sceglie la nota che soddisfa (1) e (3)
- sceglie la nota che soddisfa (2) e (3)
- sceglie la nota che soddisfa (3)
- sceglie una qualsiasi nota

Resta quindi solo da descrivere come organizzare il materiale tematico nell'esposizione e della conseguente generazione delle parti libere. Per tali algoritmi, la cui implementazione in C++ risulta molto laboriosa daremo solo una descrizione delle linee guida che hanno condotto la stesura del codice.

La procedura `faiEsposizione()` estrae a sorte un numero nell'insieme { 0 , … , 3 } per decidere quale tipo di esposizione realizzare tra i quattro possibili [40] ; tale intero è memorizzato in `modelloEntrate` . A questo punto si può inizializzare l'array `entrate` . Fatto ciò si stabilisce si posizionano opportunamente le entrate di ogni voce; in questo ambito vale la pena ricordare le estensioni delle voci e cioè

- il *soprano* può avere note comprese tra C_3 e A_4
- il *contralto* può avere note comprese tra G_2 e D_4
- il *tenore* può avere note comprese tra C_2 e A_3
- il *basso* può avere note comprese tra F_1 e E_3

Per trovare il miglior punto di partenza per la prima voce, si calcolerà l'estensione del soggetto e del controsoggetto concatenati e si cercherà poi l'ottava migliore per eseguire la melodia così ottenuta, senza ovviamente che i limiti superiore ed inferiore di quest'ultima varchino i limiti appena presentati per la voce in questione. Quindi si inseriranno soggetto in tonica e controsoggetto in dominante nelle prime posizioni del `NoteArray` associato alla voce.

Fatto ciò, si calcolerà l'entrata della seconda voce a partire dal controsoggetto della prima : cioè gli posizionerà la risposta (cioè il soggetto in dominante) o sopra (nei primi due modelli di entrate) o sotto (negli ultimi due)

[40] Per lo schema delle entrate, si veda il paragrafo 5.2

alla distanza che posiziona la nuova entrata nella posizione migliore per la voce in questione.

In maniera del tutto analoga si procede per le altre voci (sempre alternando entrate in tonica e in dominante : la prima e la terza voce espongono il soggetto in tonica e il controsoggetto in dominante, la seconda espone soggetto in dominante e controsoggetto in tonica, mentre la quarta espone solo il soggetto in dominante poiché non c'è nessun'altra a voce a tenere il soggetto in quel momento e quindi il controsoggetto non compare).

La procedura `faiPartiLibere()` ha lo scopo di riempire i buchi lasciati dalla procedura precedente e di completare così l'esposizione. Per strutturare in maniera più intuitiva il processo di generazione delle parti libere si è scelto il seguente approccio : sia A – B – C – D l'ordine di ingresso delle voci nell'esposizione, stabilito da `entrate` (cioè A è la voce che per prima espone il soggetto, B è la voce che entra per seconda presentando la risposta mentre A tiene il controsoggetto, e così via). Per meglio seguire il ragionamento, visualizziamo la situazione nel caso del primo modello di entrate

A	SOGGETTO	CONTROSOGG	A_1	A_2
B		RISPOSTA	CONTROSOGG	B_1
C			SOGGETTO	CONTROSOGG
D				RISPOSTA

Si genera anzitutto la parte libera che B tiene dopo aver eseguito il controsoggetto (che quindi si eseguirà assieme all'entrata di D e del

controsoggetto a C) che chiameremo B_1 ; poi si genera A_1 (da eseguire assieme all'entrata di C ed al controsoggetto a B) ed infine A_2 (eseguito con l'entrata di D , il controsoggetto a C e B_1) . Ci sembra più intuitivo generare di seguito A_1 e A_2 poiché formano in realtà una stessa melodia; generare prima B_1 e poi $A_1 – A_2$ è solo dovuto ad una questione di ordine (facendo il contrario resterebbe un buco tra A_2 e il controsoggetto a C) .

Seguendo tale ordine, nel generare B_1 e A_1 si utilizzeranno regole del contrappunto a tre realizzate mediante una GSG con due stringhe date; per la generazione di A_2 , invece, sarà usato il contrappunto a quattro e una GSG con tre stringhe date. In realtà, nella prassi compositiva, A_2 e B_1 sarebbero decise simultaneamente; questo però prevederebbe un meccanismo generativo che, date due stringhe in input, produca due stringhe in output (con le relative complicazioni dovute alla gestione di una situazione di errore e quindi di backtrack, tanto per citare la situazione più intricata). Non a caso, infatti, le GSG prevedono di dare in output sempre una sola stringa : la gestione di due o più stringhe di output complicherebbe di moltissimo una trattazione già abbastanza delicata.

Stabilito quindi questo ordine, la generazione della parte libera rispecchia abbastanza fedelmente i passi fatti nella generazione del controsoggetto[41] , con la sola differenza che bisogna in questo caso considerare tutte le stringhe date. In sintesi i passi compiuti dall'algoritmo di sincronizzazione generalizzato sono :

[41] Infatti sia la procedura `faiControsoggetto` che la presente `faiPartiLibere` sono un adattamento della procedura `sincronizza` del paragrafo 3.6

Sia $\underline{a}^{(1)} \ldots \underline{a}^{(n)}$ la sequenza di coppie di note date

i = 0

while $i \leq n$

– crea l'insieme $S_R(\underline{a}^{(i)})$ *// vedi paragrafo 7.2*
– Applica le regole del contrappunto a tre considerando $\underline{a}^{(i)}$ come note date e $S_R(\underline{a}^{(i)})$ come insieme delle possibilità
– Se il risultato è l'insieme vuoto allora
  . sia j il primo punto di scelta con più possibilità precedente i
  . se non esiste j allora ABORT *//non riesce a creare la parte libera*
  . fa una scelta diversa per la locazione j della parte libera
  . i = j+1 *// cioè fa backtrack alla locazione j*

  altrimenti
  . mette in posizione i della parte libera una nota tra quelle rimaste
  . i ++

Le euristiche per la scelta sono, oltre a quelle già citate per il controsoggetto, anche le seguenti :

– cerca di rendere gli accordi il più completi possibile (riproporre una stessa nota il meno possibile)
– cerca di riproporre la tonica

Il funzionamento del sistema è così completamente descritto [42] .

[42] Chiaramente il contrappunto a quattro (che genera una parte libera date tre parti) funziona analogamente solo che le $\underline{a}^{(i)}$ sono terne di note e le regole sono del contrappunto a quattro

CONCLUSIONI E

SVILUPPI FUTURI

Il presente lavoro si è svolto su due piani : anzitutto ha presentato e caratterizzato un particolare tipo di grammatiche (le OLIN) mostrando che il potere generativo è pari a quello delle grammatiche regolari; ha poi presentato un meccanismo per sincronizzare opportunamente una stringa data in base ad un certo insieme di regole ed ha poi esteso questo meccanismo a più stringhe date. Essendo le grammatiche ottenute di tipo OLIN, si è mostrato che il linguaggio generato da tale procedimento era regolare. Infine ha applicato tutta la teoria presentata ad un esempio musicale : sincronizzare il soggetto col controsoggetto e poi le parti libere con questi ultimi all'interno dell'esposizione di una fuga reale. Infine si è presentato il sistema implementato nella sua totalità.

A seconda del piano considerato si possono fare quindi varie considerazioni :

1) *dal punto di vista dell'informatica teorica* si può dire che è possibile dare una ulteriore caratterizzazione dei linguaggi regolari (quelli cioè generati da una grammatica OLIN) ; quindi la linearità destra equivale alla regolarità non solo se occorre nelle parti destre delle produzioni della grammatica ma anche nelle parti sinistre (originando così una grammatica contestuale).

   Inoltre è stato dato un meccanismo per generare stringhe sincronizzabili con una data parola (o più di una) secondo opportune regole specificando solo queste ultime e la stringa (o le stringhe) interessata. Il processo è generativo e può essere implementato in maniera efficiente. L'applicabilità del metodo può essere estesa a tutti quei fenomeni di sincronizzazione in cui le regole riguardino porzioni costanti delle parole coinvolte.

I risultati proposti possono essere ulteriormente generalizzati andando a definire una grammatica di sincronizzazione per espressioni regolari : si può pensare, cioè, di passare al sistema non una sola stringa, ma più stringhe (anche infinite di numero) codificate tramite una espressione regolare. Si può anche dedicare attenzione alla possibilità di simulare alcuni tipi di sticker system con un numero finito di grammatiche (come già discusso nel paragrafo 3.7). Inoltre sarebbe interessante cercare di identificare, dato un insieme di regole R , l'insieme di tutte le stringhe sincronizzabili secondo esso. Chiaramente tutti questi lavori esulano dall'applicazione musicale e quindi sono stati rinviati ad un lavoro futuro.

2) *dal punto di vista dell'informatica musicale* si può aggiungere il presente lavoro ai vari fatti per adattare i processi generativi e formali tipici dell'informatica teorica ad un ambiente da essa lontano come è la musica. La relativa semplicità con cui è stato affrontato e risolto il problema fa ben sperare per lavori futuri volti ad eliminare i vincoli posti in questa prima fase di sviluppo del sistema. In particolare :

   – *la tonalità :* può facilmente essere rimosso il vincolo di pensare tutto in do maggiore. E' stata fatta questa scelta per semplificare le notazioni e l'implementazione, ma può essere generalizzata accettando tutte le tonalità. Infatti le regole non dipendono da ciò ed il sistema può semplicemente trasportarsi tutto in do maggiore, in questa tonalità lavorare e riportare poi tutti i risultati nella tonalità originale.

   – *il ritmo :* come già detto, questo è stato forse il più grosso limite del nostro sistema nel senso che le esposizioni prodotte sono molto lontane

da esposizioni reali poiché in queste ultime molto si gioca sul piano ritmico.

Comunque il lavoro fatto è utile anche volendo togliere questo vincolo : si può infatti, dato il soggetto, toglierne il ritmo scoprendone quindi le note costituenti fondamentali e su queste realizzare l'esposizione applicando il lavoro presente; in seguito si rimette il ritmo nel soggetto ed obbedendo a regole di varietà si decidono opportuni ritmi per controsoggetto e parti libere.

Il primo compito (l'eliminazione del ritmo dal soggetto) è più complesso di quanto possa sembrare : comunque il problema può essere risolto applicando le *time span reductions* presentate e discusse in dettaglio in [L,J83] . Lo svantaggio di queste regole è il loro numero elevato e la grande predisposizione a conflitti di parsing; con opportuni accorgimenti, però, entrambi questi problemi possono essere ridotti.

Il secondo compito (identificazione del ritmo nelle parti create dal sistema) è invece più semplice di quanto possa sembrare : si può semplicemente dare al sistema una base di conoscenza (creata a mano o tramite procedimenti di apprendimento automatico) da cui attingere informazioni nelle varie situazioni. Il risultato sarà chiaramente poco vario ed originale ma sicuramente veloce e ben funzionante. Procedimenti più raffinati sono poco pensabili perché richiedono comunque un forte intervento personale guidato dal buon gusto e dalla sensibilità, cose che un elaboratore per il momento gestisce male.

– *fuga tonale :* la scelta di limitarsi solo a soggetti che generano una fuga tonale è stata semplicemente di comodo : non si voleva sovraccaricare questo primo lavoro con un altro argomento molto delicato della composizione. Infatti le regole per la realizzazione della risposta in una

fuga tonale sono tantissime e poco deterministiche (in alcune situazioni sono ben applicabili, in altre meno; per uno stesso soggetto più risposte sono possibili e spesso sono tutte lecite e utili per un qualche motivo; etc.) .

Il lavoro fatto comunque può essere esteso per la gestione di questo caso : basta infatti sostituire la attuale funzione che crea la risposta (una semplice trasposizione in tonalità di dominante) con una molto più complessa che codifichi tutti i vari casi contenuti nella teoria della fuga.

E' anche interessante fare alcuni rapidi confronti con i lavori presentati nell'esposizione come illustri predecessori della presente tesi. Rispetto al testo [L,J83] vale la pena notare alcune differenze importanti : anzitutto l'approccio al problema è diverso, essendo il loro un approccio analitico mentre il nostro generativo. Inoltre il lavoro qui svolto affronta una problematica molto particolare mentre l'altro testo vuole affrontare assieme tutte le possibili composizioni e pratiche musicali : è' ciò che ci ha reso possibile formulare la nostra trattazione in maniera chiara e facilmente leggibile, senza accontentarci di presentare a parole i principi guida ma dandone una precisa formalizzazione. I risultati ottenuti sono inoltre stati implementati in maniera efficiente dando vita ad un programma che gira in tempi brevissimi. Su questi punti chiaramente il lavoro presentato risulta competitivo e più soddisfacente : il prezzo pagato è quello di una notevole semplificazione dei fenomeni affrontati. Un approccio più realistico che vada a considerare la problematica ritmica e la fuga tonale dovrà per forza perdere un po' in termini di eleganza ed efficienza per adattarsi alla situazione più complessa che si affronterà.

Rispetto a [B,D,J99] , i paragoni risultano più interessanti poiché entrambi i lavori hanno una finalità generativa e sono stati implementati. In entrambi la problematica delle varie tonalità è stata accantonata poiché poco interessante e quindi inutilmente pesante; inoltre in entrambe è accettato il ricorso al backtrack. E' interessante notare come una forma più rigida (come è l'esposizione di una fuga) crei meno problemi in questo senso : il ricorso da noi fatto al backtrack è risultato piuttosto basso. Ciò è comunque abbastanza logico poiché nel loro esperimento, avendo margini di libertà molto maggiori, era abbastanza facile prendere una strada che non soddisfacesse tutti i parametri; nel nostro caso, essendo semplificato il problema dall'assenza del ritmo e del testo, i vincoli da rispettare erano decisamente meno. Vale però la pena notare che la nostra gestione delle quattro voci trova nell'altro lavoro solo due voci : il canto ed il basso. E' questa una notevole semplificazione (sia da un punto di vista teorico che di efficienza di implementazione) : il nostro programma ricorre a backtrack quasi solo nella generazione di parti libere, quando cioè deve gestire tre o quattro voci. Inoltre, come già osservato nell'introduzione, anche in questo caso le regole sono dette solo a parole, nonostante le critiche mosse dagli autori contro questa prassi : il nostro lavoro è in questo contesto molto più dettagliato poiché non solo le esprime in forma molto chiara (le definizioni date nei capitoli precedenti non erano a parole ma risultano a nostro avviso ugualmente chiare e leggibili) ma esprime anche un efficiente meccanismo formale (le grammatiche di sincronizzazione) per creare con esse un sistema generativo.

Vale infine la pena di commentare il diverso approccio alla generazione: noi usiamo una generazione on–line (nel senso che creiamo la musica da sinistra a destra, dalla prima nota all'ultima), più tipico delle derivazioni in una grammatica regolare, mentre loro applicano un meccanismo generativo almeno non contestuale (in realtà siamo abbastanza convinti che un

ricorso seppur minimo alla contestualità risulta necessario). Ciò è dovuto al fatto che il problema da noi affrontato è definito piuttosto rigidamente (le regole per la creazione della fuga impongono schemi esatti, nell'esposizione non fanno ricorso a modulazioni, non devono seguire nessun testo poetico, …); il loro problema è invece molto meno vincolato e quindi lascia una libertà di creazione che male si realizzerebbe con un procedimento on–line. Per gli stessi motivi anche i lavori precedenti al presente (ed in particolare [Rom97]) hanno fatto ricorso a questo approccio : non dovendo rispettare una forma definita a priori era necessario creare quella che in [B,D,J99] è chiamata *macroforma* potendo agire su tutto il brano simultaneamente (un meccanismo chiaramente off–line).

Concludendo, quindi, le premesse per buoni sviluppi futuri ci sono tutte: l'idea di creare esposizioni di fughe in maniera del tutto automatica si può considerare realizzabile in tempi anche abbastanza brevi. La fuga però non è formata solo dall'esposizione : ci sono anche i divertimenti e gli stretti. La problematica degli stretti non è lontana da quella dell'esposizione : l'unica differenza è che le entrate di soggetto e risposta avvengono in maniera molto più ravvicinata (prima che la voce precedente abbia terminato di esporre il suo soggetto o risposta). Il meccanismo risolutivo può quindi essere simile : basta aggiungere una funzione (neanche troppo complessa) che determini in quali posizioni all'interno del soggetto la voce successiva possa immettere in stretto la sua entrata.

La problematica dei divertimenti è invece molto più difficilmente risolvibile : in questo punto della fuga è infatti fondamentale l'intervento della fantasia del compositore nell'elaborare il materiale tematico a propria disposizione al fine di sviluppare al massimo gli elementi presentati

nell'esposizione, creando contemporaneamente il raccordo con gli stretti (in cui tutto il materiale esposto viene raggruppato in forma di una grandioso riassunto finale). Come già più volte detto, anche in questo caso l'uomo risulta (per fortuna!!) insostituibile dalla macchina.

BIBLIOGRAFIA

[Ach98] M.Achilli *"Pattern con operatori per il controllo di riscritture libere da contesto"* Università di Roma "La Sapienza" – dipartimento di scienze dell'informazione (1998)

[Bar81] M.Baroni *"Sulla nozione di grammatica musicale"* "Rivista italiana di musicologia" (XVI) (1981)

[B,D,J99] M.Baroni , R.Dalmonte , C.Jacoboni *"Le regole della musica"* EDT (1999)

[Bri71] W.Bright *"Language and music: areas for cooperation"* "Musique en jeu" (V) (1971)

[Cer96] M.Cervellin *"Teorie algebriche, linguaggi e pattern"* Università di Roma "La Sapienza" – dipartimento di scienze dell'informazione (1996)

[Coh62] J.E.Cohen *"Information theory and music "* "Behavioral science" (VII) (1962)

[Dub] T.Dubois *"Trattato di contrappunto e fuga"* Ricordi – Milano (s.d.)

[Ged] A.Gedalge *" Trattato della fuga "* Ed.Curci – Milano (1995)

[Her76] M.Herndon *"The trasformational model of linguistics : its implications for music"* "Semiotica" (XV) (1976)

[L,J83] F.Lerdahl , R.Jackendoff *"A generative theory of tonal music"* The MIT Press (1983)

[L,S73] B.Lindblom , J.Sundberg *"Towards a generative theory of melody"* Atti del I congresso internazionale di semiotica musicale (Beograd) (1973)

[Moo77] J.A.Moorer *"On the transcription of music sound by computer"* "Computer music journal" (1977)

[Mey57] L.B.Meyer *"Maening in music and information theory"* "Estetica e teoria dell'informazione" (Bompiani) (1972)

[Nar77] E.Narmour *"Beyond Schenkerism"* University of Chicago Press (1977)

[Net58] B.Nettl *"Linguistic approaches to musical analysys"* "Musique en jeu" (V) (1971)

[P,R,S98] G.Paun , G.Rozenberg , A.Salomaa *" DNA Computing "* Springer (1998)

[Pin56] R.C.Pinkerton *"Information theory and melody"* "Scientific American" (CXCIV) (1956)

[Rom97] F.Romitti *"Un sistema per la generazione di brani musicali basato sulla riscrittura vincolata di stringhe"*

Università di Roma "La Sapienza" – dipartimento di scienze dell'informazione (1997)

[Wid94] G.Widmer *"The synergy of music theory and AI : learning multi–level expressive interpretation"*
Austrian Research Institute for AI (1994)

[Wid99] G.Widmer *"Large–scale induction of expressive performance rules : first quantitative risults"*
Austrian Research Institute for AI (1999)

[Xen71] I.Xenakis *"Formalized music"*
Indiana University Press (1971)

APPENDICE

Appendice I : Tavole dei più importanti soggetti e controsoggetti di Bach

I.1 Le fughe per organo

Numero di catalogo	Estensione del soggetto	Distanza del controsoggetto dal soggetto
BWV 131/a	9^ minore	6^ minore
BWV 531	10^ maggiore	13^ maggiore
BWV 532	9^ maggiore	6^ maggiore
BWV 533	5^	10^ maggiore
BWV 534	9^ minore	17^ maggiore
BWV 535	6^ minore	11^
BWV 536	6^ maggiore	7^ minore
BWV 537	7^ diminuita	12^ diminuita
BWV 538	8^	10^ maggiore
BWV 539	4^	8^
BWV 540	6^ minore	7^ maggiore
BWV 541	6^ maggiore	10^ minore
BWV 542	11^	10^ maggiore
BWV 543	11^	12^
BWV 544	7^ diminuita	12^ diminuita
BWV 545	8^	11^
BWV 546	9^ minore	10^ minore

BWV 547	6^ maggiore	10^ minore
BWV 548	8^	15^
BWV 549	9^ minore	13^ minore
BWV 550	9^ maggiore	13^ minore
BWV 551	5^	10^ minore
BWV 552	6^ minore	13^ minore
BWV 553	4^	6^ maggiore
BWV 554	8^	8^
BWV 555	5^	8^
BWV 556	5^	7^ minore
BWV 557	7^ minore	8^
BWV 558	6^ minore	8^
BWV 559	6^ minore	8^
BWV 560	8^	14^ minore
BWV 561	9^ minore	13^ minore
BWV 564	9^ maggiore	15^
BWV 565	8^	13^ minore
BWV 566	9^ maggiore	8^
BWV 574	8^	11^
BWV 575	8^	10^ maggiore
BWV 576	8^	10^ maggiore
BWV 577	9^ maggiore	15^
BWV 578	8^	15^
BWV 579	8^	8^
BWV 946	6^ maggiore	10^ maggiore

I.2 : Le fughe dal "Clavicembalo ben temperato"

Numero di catalogo	Estensione del soggetto	Distanza del controsoggetto dal soggetto
BWV 846	6^ maggiore	10^ minore
BWV 847	7^ maggiore	10^ minore
BWV 848	10^ maggiore	11^
BWV 849	4^ diminuita	8^
BWV 850	6^ maggiore	10^ maggiore
BWV 851	7^ diminuita	11^ diminuita
BWV 852	a) 6^ maggiore b) 8^	a) 17^ maggiore b) 13^ maggiore
BWV 853	6^ minore	8^
BWV 854	8^	7^ minore
BWV 855	8^	8^
BWV 856	6^ minore	6^ maggiore
BWV 857	8^	13^ diminuita
BWV 858	7^ minore	8^
BWV 859	5^	8^
BWV 860	9^ maggiore	11^
BWV 861	7^ diminuita	10^ minore
BWV 862	6^ maggiore	10^ maggiore
BWV 863	8^	7^ maggiore
BWV 864	5^ diminuita	13^ minore
BWV 865	9^ minore	10^ maggiore

BWV 866	10^ minore	13^ minore
BWV 867	9^ minore	10^ minore
BWV 868	7^ minore	9^ maggiore
BWV 869	10^ minore	13^ minore
BWV 870	6^ maggiore	11^
BWV 871	5^	10^ minore
BWV 872	5^	6^ maggiore
BWV 873	8^	10^ minore
BWV 874	7^ minore	6^ maggiore
BWV 875	8^	13^ maggiore
BWV 876	6^ maggiore	10^ maggiore
BWV 877	6^ minore	6^ minore
BWV 878	4^	8^
BWV 879	9^ minore	15^
BWV 880	9^ maggiore	8^
BWV 881	7^ diminuita	12^
BWV 882	8^	10^ minore
BWV 883	8^	10^ minore
BWV 884	11^	12^
BWV 885	6^ minore	12^ diminuita
BWV 886	9^ maggiore	8^
BWV 887	9^ minore	6^ maggiore
BWV 888	6^ maggiore	7^ minore
BWV 889	7^ diminuita	14^ minore

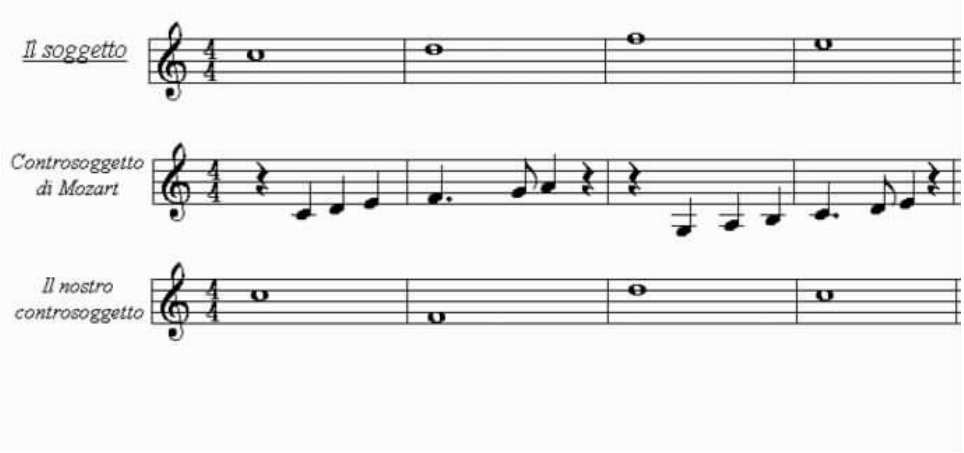
BWV 890	9^ maggiore	12^
BWV 891	7^ diminuita	7^ minore
BWV 892	8^	10^ maggiore
BWV 893	9^ minore	13^ minore

Appendice II : Applicazione del sistema ad un soggetto di Mozart

L'applicazione del sistema che ora presenteremo realizza una esposizione di una fuga a quattro voci usando un celebre tema di Wolfgang Amadeus Mozart : è il tema del quarto movimento della sinfonia K551 detta "Jupiter". Lo stesso autore vi costruisce sopra un fugato (cioè una fuga più libera e meno complessa strutturalmente). Ci sono vari motivi per cui abbiamo scelto tale tema :

- è un tema celeberrimo
- si adegua perfettamente ai vincoli che ci siamo posti (è in do maggiore, procede per semibrevi, è realizzabile in fuga reale)
- il sistema produce un risultato interessante, nonostante sia ancora ad una fase primordiale (non considera il ritmo)

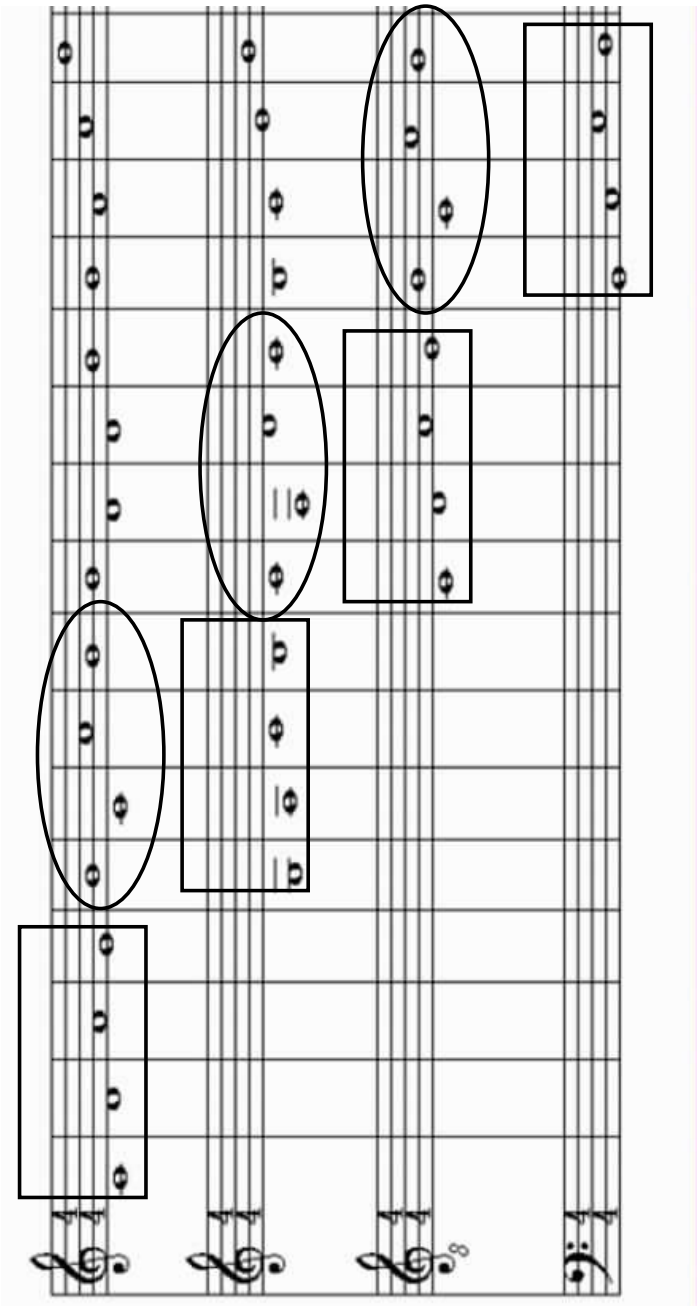
Riportiamo ora il tema, il controsoggetto proposto da Mozart e quello creato dal nostro sistema.



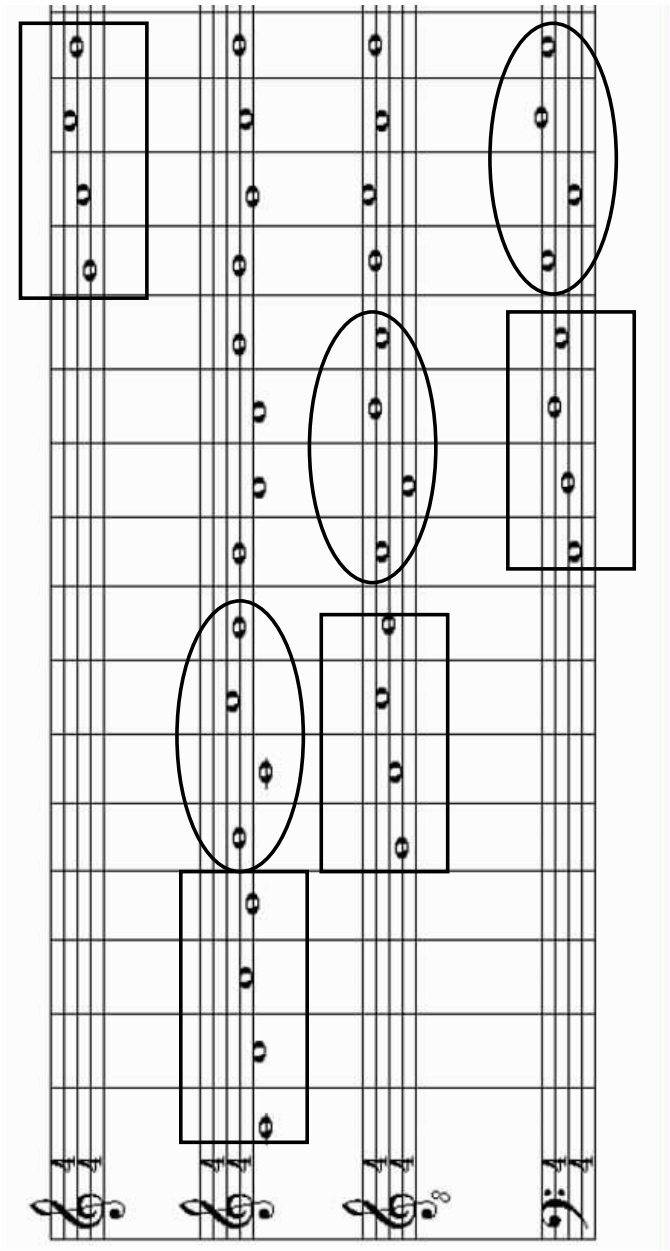
Chiaramente il controsoggetto di Mozart usa fortemente il contrasto ritmico col soggetto; non potendo per il momento fare altrettanto, il nostro sistema produce comunque un risultato buono poiché riesce, seppur con i minimi mezzi di cui dispone, a ben differenziare le due entità : il soggetto, infatti, procede per "piccoli passi" (una seconda ed una terza minore ascendenti ed una seconda discendente) mentre il controsoggetto compie salti più ampi (quinta discendente, sesta maggiore ascendente e poi una seconda discendente). Chiaramente il risultato non è paragonabile, ma è già un buon inizio.

Di seguito riportiamo le quattro possibili entrate; per meglio leggerle abbiamo evidenziato con dei rettangoli le entrate dei soggetti e con degli ovali quelle dei controsoggetti. E' interessante notare le piccole differenze tra le parti libere di ogni modello di esposizione : mentre soggetto e controsoggetto cambiano solo la voce di appartenenza (e quindi l'ottava di esecuzione), per le parti libere ciò non è sempre vero. Ciò è dovuto al fatto che per esse non valgono le regole di rivoltabilità e quindi alcune situazioni concesse in voci interne non sono lecite se realizzate dal basso (a causa del più volte citato problema del secondo rivolto, o armonizzazione di quarta e sesta di un accordo).

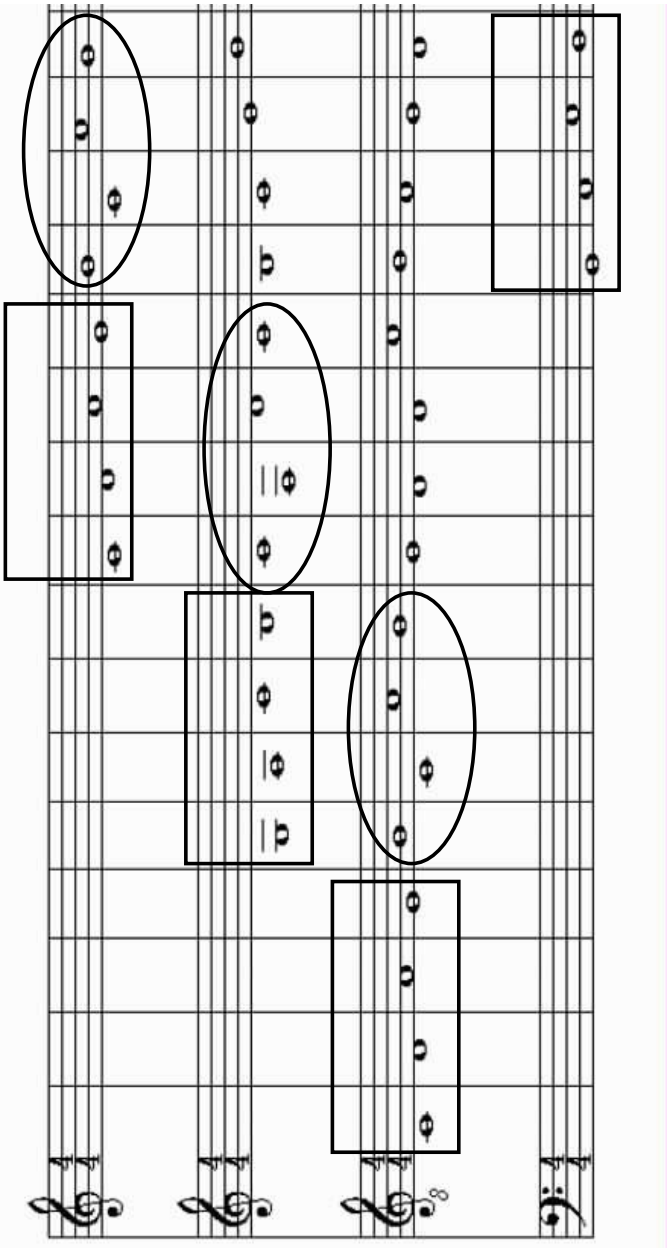
1° Modello : soprano – contralto – tenore – basso



2° Modello : contralto – tenore – basso – soprano



3° Modello : tenore – contralto – soprano – basso



4° Modello : basso – tenore – contralto – soprano