修士論文

繰り返し学習を用いたドラム譜解析

藤谷　光伸

主指導教員　　東条　敏

北陸先端科学技術大学院大学
先端科学技術研究科
（情報科学）

令和２年３月

Abstract

With the advance of information science and the use of music data, research related to music, such as automatic generation of music, composition support, and analysis of music, has been performed using information science technology. Furthermore, it is pointed out that music takes a form very similar to a language, such as recognition using ears and being described as a symbol on a musical score, and music and language evolved from the same origin. We also know that the brain functions used for music and language overlap. From these similarities, it is thought that music has a grammar like language. Therefore, analysis of music using a technique cultivated in a language has been attempted. However, unlike language, music has a vague meaning and is difficult to formalize, so it is difficult to implement by computer. In addition, when a statistical natural language processing method is used, a large amount of music data is required. Therefore, it is necessary to try to analyze music from the essence of human language processing. In this study, we analyze the music score using the method of evolutionary linguistics.

Evolutionary linguistics is a study aimed at elucidating human thoughts and properties by pursuing the origin and evolution of human language from two aspects, biological evolution and cultural evolution. Noam Chomsky defined language as a human-specific biological trait and defined the mechanism as a generative grammar. The purpose is to understand the human brain and thoughts from language analysis. In addition, Chomsky proposed that universal grammars exist, with human beings having a natural language acquisition device. Only a limited number of sentences can be learned during childhood to acquire language. Therefore, Simon Kirby asserted that even without assuming a special device for acquiring language, infants acquire generative grammar by performing generalized learning with little linguistic knowledge. And, computer simulations using an iterated learning model showed that a language without a structure has synthesizing and recursive processes, indicating the possibility of cultural evolution of the language. In this research, based on this Kirby's Iterated Learning Model, a model for analyzing music scores is created.

Kirby's Iterated Learning Model is a computer simulation of verbal communication composed of two types of agents, parent and child. The parent agent generates a character string from its own language knowledge and passes the character string and the meaning of the character string to the child agent. The child agent acquires a grammar from the stored linguistic knowledge by performing generalization learning. Then, the child agent that has acquired the grammar becomes a new parent agent and passes the character string to the new child agent. By repeatedly learning this process over several generations, linguistic knowledge with high expressiveness is acquired.

Previous research has attempted to discover the structure of music based on Kirby's Iterated Learning Model. A symbol sequence is created from a musical score, and learning is performed using an iterated learning model. As a result of an experiment of music score analysis using Burg Müller 25 Etüden, it was possible to observe a part of the cadence rule and the progression from dissonance to consonance, but there is still a problem in finding the grammatical rules of music.

The problems of previous studies are that they assume too much meaning similar to language for music and symbolize music scores. Previous study, piano scores are used for

musical scores. In the piano score, two staves, the right hand (main melody) and the left hand (accompaniment), are arranged at the top and bottom, and progress simultaneously. When symbolizing this score, the order of notes included in the range of one beat is not considered. And when the number of left-hand notes is smaller than the number of right-hand notes, the left-hand notes are equally allocated and symbolized. Therefore, in this study, symbolization is performed while maintaining the order of notes. In addition, the piano score is very complicated and unsuitable for the analysis of the score because the progress of the accompaniment and the main melody must be considered. Therefore, this time, the drum score was adopted as the music score. Drum score is simpler than piano score and has a musical rhythm structure such as 8 beats, so it can be said that it is suitable for discovering the structure of music. In this research, instead of labeled context-free grammar what Kirby's iterative learning model and the previously used, we constructed it to acquire rules of context-free grammar, created initial production rules from drum score, and generated symbols from production rules. We generated a sequence and tried to find drum grammar rules using iterated learning model. This model does not use musical knowledge and is a model for learning grammar from a common part of a symbol sequence.

As a result of the experiment, the grammar rules converged in earlier generations compared to previous studies. Then, the obtained grammar rules were converted to Chomsky normal form and converted to Greibach normal form, and the result was a regular grammar. It is suggested that the generated grammar obtained from the drum score may have only the same restrictions as regular grammar.

In this study, Iterated Learning Model learned drum score in units divided into bars. For this reason, rhythm instruments such as drums cannot be said to have regular grammar. In the future, it is necessary to devise a model for learning grammar without specifying the budget unit. Also, this time, we could only collect 9 drum scores, so it was an experiment with few sentences. Therefore, it is a future work to collect and analyze more score data.

概要

情報科学が進歩し，データで音楽を扱う様になったことから，情報科学の技術を用いて楽曲の自動生成や作曲支援，音楽の解析など，音楽に関連した研究が行われている．さらに，音楽は耳を使って認識することや，楽譜に記号として記述されることなど，言語とよく似た形態をとり，音楽と言語は同じ起源から進化したものであると指摘されている．また，音楽と言語に使用する脳の機能が重複していることが分かっている．これらの類似点から，音楽にも言語ように文法が存在すると考えられる．そのため，言語で培われてきた手法を用いた楽曲の解析などが試みられている．しかし，言語と違い音楽は意味が曖昧であり形式化を行うことが困難であるため，計算機による実装が難しい．また，統計的な自然言語処理の手法を用いる場合，大量の楽曲データを必要とする．そのため，音楽の解析に，より人間の言語処理の本質からのアプローチを試みる必要がある．そこで，本研究では進化言語学の手法を用いて楽曲の楽譜解析を行う．

進化言語学は，人間の言語の起源や進化を，生物的進化と文化的進化の 2 つの側面から追及することによって人間の思考や性質の解明を目的とする学問である．Chomsky は，言語を人間特有の生物学的形質と位置づけ，その仕組みを生成文法として定義した．この目的は，言語の分析から，人間の脳や思考を理解することにある．さらに，Chomsky は，人間は生まれながらに言語を獲得する装置を持っているとし，普遍文法を提唱した．これに対し，Kirby は，言語を獲得する特殊な装置を仮定しなくても，言語を獲得する幼少期に，限られた数の文しか学習することができないために，幼児の頭で汎化学習が行われることで生成文法を獲得するとし，繰り返し学習モデルを用いた計算機によるシミュレーションで，構造を持たない言語が合成性や再帰性を持つ過程を示し，言語の文化的進化の可能性を示した．本研究では，この Kirby の繰り返し学習モデルに倣って，曲の楽譜の解析を行うモデルを作成する．

Kirby の繰り返し学習モデルは，親と子の二種類のエージェントで構成される言語コミュニケーションの計算機によるシミュレーションである．親エージェントは自身の言語知識から文字列を生成し，子エージェントにその文字列と，その文字列の意味するところを受け渡す．子エージェントは蓄えた言語知識から，汎化学習を行うことで文法を獲得する．そして，文法を獲得した子エージェントは新たに親エージェントとなり，新しい子エージェントに文字列を受け渡す．この

プロセスを数世代に渡って繰り返し学習することで，表現度の高い言語知識を取得する．

先行研究では，この Kirby の繰り返し学習モデルを元に音楽の構造発見を試みている．楽譜からシンボル列を作成し，繰り返し学習モデルを用いて学習を行う．ブルグミュラー２５の練習曲を用いた楽譜解析の実験の結果，カデンツ規則の一部や，不協和音から協和音への進行を観測することができたが，音楽の文法規則の発見にはまだ課題を残す結果となった．

先行研究の問題点は，音楽に対して，言語と同じような意味を仮定しすぎている点と，楽譜のシンボル化にある．この研究では，楽譜にピアノ譜を用いている．ピアノ譜は，右手（主旋律）と左手（伴奏）の二つの五線譜が上段と下段に並び同時に進行する．この楽譜のシンボル化の際に，一拍の範囲に含まれている音符の順序を考慮しておらず，右手の音符に比べ左手の音符数が少ない時に，左手の音符を均等に割り振ってシンボル化している．そこで，本研究では，音符の順序を保ったままシンボル化を行う．また，ピアノ譜は，伴奏と主旋律の進行を考慮しなければならないため，非常に複雑であり楽譜の解析には不適であった．そこで，今回は楽譜としてドラム譜を採用した．ドラム譜はピアノ譜と比べ単純であり，8 ビートなどの音楽のリズム構造を持っているため，音楽の構造発見に適していると言える．

本研究では，Kirby の繰り返し学習モデルや以前までのラベル付きの文脈自由文法ではなく，文脈自由文法の規則を獲得するように構築し，ドラム譜から初期の生成規則を作成し，生成規則からシンボル列を生成し，繰り返し学習を用いてドラムの文法規則の発見を試みた．このモデルは音楽的な知識を用いず，シンボル列の共通部分から文法の学習を行うモデルである．実験の結果，先行研究と比べ，早い世代で文法規則が収束した．そして，取得した文法規則に対し，チョムスキー標準形への変換を行い，グライバッハ標準形への変換を行ったところ，正規文法となった．ドラム譜から獲得した生成文法は，正規文法程度の制限しか持っていない可能性を示した．

本研究では，ドラム譜の解析にあたり，1 小節ごとに区切った単位での学習を行っている．そのため，ドラムのようなリズム楽器は正規文法であるとまでは言えない．今後は区切る単位を指定せず文法の学習を行うモデルを考案する必要がある．また，　今回は 9 曲のドラム譜を集めることしかできなかったため，少ない文での実験となった．そのため，より多くの楽譜データを集めて解析を行うこ

とが今後の課題である.

目次

図目次

表目次

第1章 はじめに

1.1 研究背景

音楽は耳を使って認識することや，楽譜に記号として記述されることなど，言語とよく似た形態をとり，音楽と言語は同じ起源から進化したものであると指摘されている．さらに，音楽と言語に使用する脳の機能が重複していることが分かっている[8]．これらの類似点から，音楽にも言語ように文法が存在すると考えられる．

1.2 先行研究

先行研究[2]では，繰り返し学習モデルを元に音楽知識を用いずに楽譜から，音楽の規則などの構造発見を試みている．その結果，不協和音から協和音への進行や，カデンツと呼ばれる音楽の規則の一部が観察されたが，音楽の文法規則発見にはまだ課題を残す．この研究では，楽譜のシンボル化を行う際に，ベース音を主旋律に均等に割り振っていたことや，四分音符や八分音符などの音符の順序などの情報を考慮していなかった問題点が考えられる．

1.3 研究目的

音楽には音楽理論などのルールがあり，コード進行などの構造を持っているが，言語と比べ曖昧である．本研究では，進化言語学の知識や手法を用いて，音楽知識を用いずに楽譜の解析を行い，音楽の構造の発見と評価を行い，進化言語学の知識を用いた楽譜解析の有効性を示すことを目的とする．

1.4 本論文の構成

本論文の構成は，第 1 章で研究背景，研究目的について述べる．第 2 章では言語と音楽の関係について述べる．第 3 章では形式文法について述べる．第 4 章では Kirby の繰り返し学習モデルについて述べる．第 5 章では本研究システムで用いた手法である，繰り返し学習モデルを用いた楽譜解析について述べる．第 6 章では実験について述べる．第 7 章でおわりに本研究のまとめについて述べる．

第2章 言語と音楽

我々は小鳥のさえずりを一種の音楽のように捉えるが，小鳥にとってはこのさえずりは立派な言語でのコミュニケーションであり，そこには文法が存在する．我々がこのさえずりを音楽と認識するのは，そこにリズムや抑揚を感じるためである．人間の脳に関した研究では，一般に言語に関する機能は左脳にあり，音楽に関する機能は右脳にあると考えられているが，脳画像技術を用いた研究によって，音楽と言語は脳の共通の部分を使用していることが指摘されている[8]．音楽も言語も，もともと自然にあるものではなく，人間の文化によって生み出され進化してきた．言語には，強弱や高低といったアクセントや，モーラやシラブルと呼ばれる拍や音節の概念がある．これは，音楽でのメロディーやリズムによく似ている．このように，言語と音楽には共通点が多く，同じ起源から進化したものであるといわれている．この章では言語と音楽について述べる．

2.1 言語と楽曲解析

言語と音楽は同じ起源だと考えられており，類似点が多いこともあり，音楽の楽曲解析に自然言語処理で培われた知見が用いられることがある．自然言語を用いた楽曲解析の手法は大きく分けて２つあり，１つは大量の楽曲データを集めて統計的手法で学習する方法で，もう１つは自然言語処理の知見を活かし音楽的知識に基づいた解析を行う方法である．統計的な手法は，学習するデータ量が十分であれば，良い結果を得ることができる．ルールベースの手法では，音楽を形式的に記述し，音楽的知識を網羅的に記述する必要があるが，良い結果を得ることができる．しかし，これらの手法は，膨大な計算資源と複雑なシステムの構築を必要とする問題がある．

我々にとって言語は，コミュニケーションの手段であり，複雑な意味を表現することができる必要がある．一方で，音楽は言語と違い複雑な意味を表現する必要はないため，言語と比べ単純な構造をしていると考えられる．音楽は言語と比べ曖昧であり，言語のように形式的に記述することが難しい．そこで，音楽の解析には，自然言語処理の手法ではなく，人間の言語処理の本質からのアプローチを試みる必要がある．

2.2 進化言語学

進化言語学は，人間の言語の起源や進化を，生物的進化と文化的進化の 2 つの側面から追及することによって人間の思考や性質の解明を目的とする学問である．チョムスキーは，言語を人間特有の生物学的形質と位置づけ，その仕組みを生成文法として定義した．この目的は，言語の分析から，人間の脳や思考を理解することにある．さらに，チョムスキーは，人間は生まれながらに言語を獲得する装置を持っているとし，普遍文法を提唱した．これに対し，カービィは，言語を獲得する特殊な装置を仮定しなくても，言語を獲得する幼少期に，限られた数の文しか学習することができないために，幼児の頭で汎化学習が行われることで生成文法を獲得するとし，繰り返し学習モデルを用いた計算機によるシミュレーションで，構造を持たない言語が合成性や再帰性を持つ過程を示した．このカービィの枠組みは，チョムスキーの主張に対して，言語の文化的進化の可能性を示した．

2.3 音楽の文法

ジュウシマツの歌には文法があり，それは正規文法であり，有限状態オートマトンで表現することができると言われている[7]．一方で，我々の言語はほぼ文脈自由文法となっており，生成文法によって定義することができる．音楽には音楽理論が存在する．音楽理論の 1 つにカデンツと呼ばれる規則がある．カデンツとは，我々に音楽の終了を感じさせる制約であり，この制約が満たされていないと曲が中途半端であると感じる．音楽の開始と終了は同じ調であるべきとされ，そこでの主和音はトニックと呼ばれる特徴を持つ．このトニックから見て，完全五度上の和音をドミナントと呼び，完全五度下の和音はサブドミナントと呼ばれる．ドミナントは緊張を持たせ，トニックに戻ることによって終始感を持たせる．トニックを T，ドミナントを D，サブドミナントを S とすると，カデンツの規則は以下の様になる．

1. T → D → T
2. T → S → D → T
3. T → S → T

これらのカデンツ規則の一部を有限状態オートマトンで示したものを図 2.1 に示す.

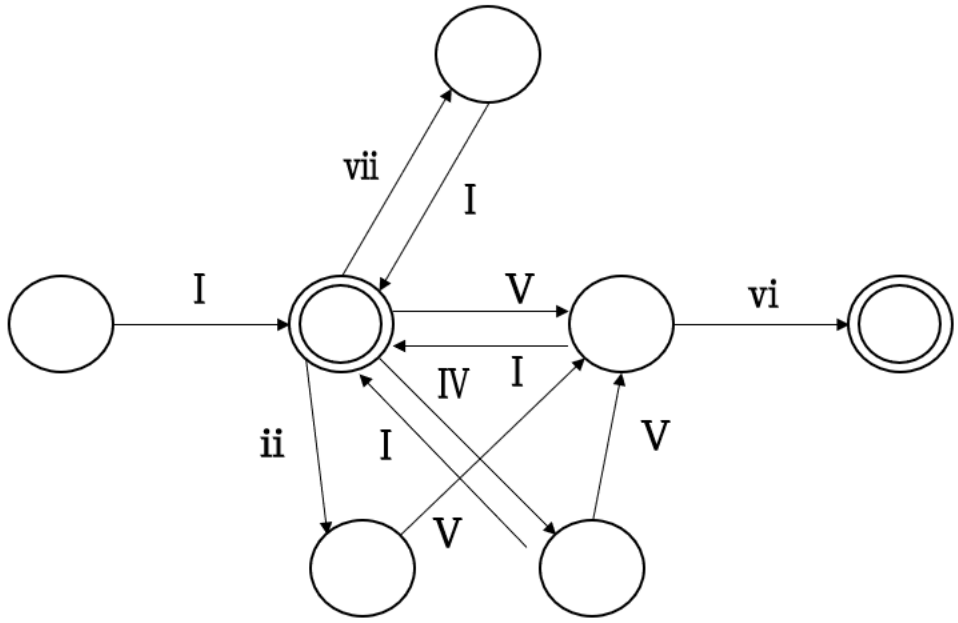

図 2.1 カデンツを表現した有限状態オートマトン

ここで, I, ii, iii... はそれぞれ, ド, レ, ミに対応した三和音のことである. Ⅰ, Ⅳ, Ⅴはそれぞれ, Ⅰ（ドミソ), Ⅳ（ファラド), Ⅴ（ソシレ）の様に大文字で長三和音を表す. ii, iii, vi, viiは短三和音を小文字で表している. 左端の丸から始まり, 二重丸が終点となる.

カデンツ規則を有限状態オートマトンで表現をすることができたが, 音楽の文法が正規文法であると言う訳ではない. しかし, このように音楽を生成文法によって表現する試みも行われている.

第3章 形式文法

3.1 言語と文法

人間の文化の中で自然発生的に発展してできた言語を自然言語といい，我々が普段使用している日本語や英語などの言語はこの自然言語である．一方，特定の目的のために人間が意図的に作った言語を形式言語という．チョムスキーはこの形式言語を置き換え規則によって定義した．形式文法は，生成規則と，非終端記号，終端記号とよって成り立ち，以下のように定義される．

$$G = (N, \Sigma, R, S)$$

N： 非終端記号の集合

Σ ： 終端記号の集合

R： 生成規則の集合

S： 開始記号

3.2 チョムスキー階層

形式言語の生成規則に制限をつけることで文法を 0 型，文脈依存文法(1 型)，文脈自由文法(2 型)，正規文法（3 型）の 4 つの組に分類した．これらの各文法には図 2.1 のような関係が成り立っている．

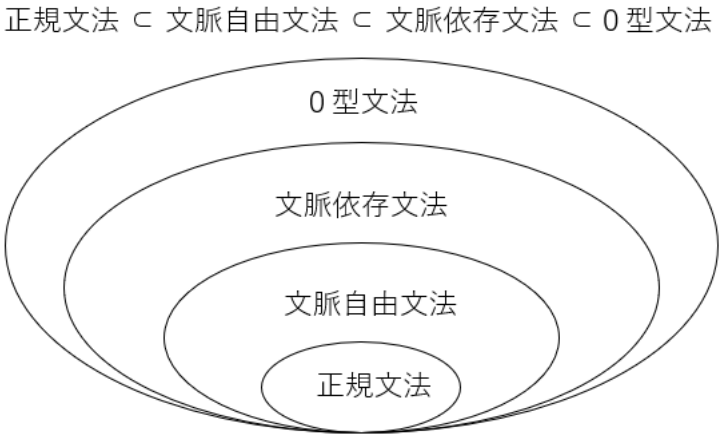

図 3.1 チョムスキー階層

3.2.1 0 型文法

0 型文法は制限のない文法で全ての形式文法を含んでいる． 0 型文法の生成規則は次のような形となる．

$$\alpha \quad \rightarrow \quad \beta$$

ここで， α， β は終端記号と非終端記号からなる任意の文字列である．

3.2.2 文脈依存文法（1 型文法）

G=(N, Σ, R, S)の R の任意の規則が以下の形をしているとき，G は文脈依存文法である．

$$\alpha \mathrm{A} \beta \quad \rightarrow \quad \alpha \gamma \beta$$

ここで，A は非終端記号， α， β， γ は終端記号と非終端記号からなる任意の文字列である．ここで，左辺に現れる α と β によって終端記号 A の書き換えを定義している．つまり，書き換え規則の適用が前後の文脈に依存している．

3.2.3 文脈自由文法（2 型文法）

G=(N, Σ, R, S)の R の任意の規則が以下の形をしているとき，G は文脈自由文法である．

$$\mathrm{A} \quad \rightarrow \quad \alpha$$

ここで，A は終端記号， α は終端記号と非終端記号からなる任意の文字列である．文脈依存文法よりも生成規則の制限が強い文法で，左辺が単一の終端記号のみからなる文法である．

3.2.4 正規文法（3 型文法）

G=(N, Σ, R, S)の R の任意の規則が以下の形をしているとき，G は正規文法である．

$$A \rightarrow a$$
$$A \rightarrow aB$$

ここで A,B は非終端記号で a は終端記号である．チョムスキー階層の中で最も制限が強い文法で，非終端記号を終端記号に置き換えるか，非終端記号を終端記号の後ろに終端記号をつけた形に置き換える規則のみを生成規則に持つ．

3.3 チョムスキー標準形

G=(N，Σ，R，S)を文脈自由文法とした時，R の任意の規則が以下の形をしているとき，G をチョムスキー標準形という．

$$A \rightarrow BC$$
$$A \rightarrow a$$

ここで，A は非終端記号，B,C は開始記号でない非終端記号，a は終端記号である．任意の文脈自由文法は，チョムスキー標準形に変換することができる．このチョムスキー標準形は，CYK 法などのアルゴリズムの基礎となっている．

3.4 グライバッハ標準形

G=(N，Σ，R，S)を文脈自由文法とした時，R の任意の規則が以下の形をしているとき，G をグライバッハ標準形という．

$$A \rightarrow \alpha X$$

ここで，A は非終端記号で，α は終端記号，X は開始記号以外の非終端記号からなる文字列をあらわす．すべての文脈自由文法はグライバッハ標準形に変換することができ，グライバッハ標準形は変換先（右辺）に変換元（左辺）の規則が出現しない特徴がある．

3.5 文脈自由文法の変形

この節では文脈自由文法の変換について述べる．文脈自由文法を変形するこ

とで扱いやすい形に変形することができる．文脈自由文法をチョムスキー標準形に変形し，グライバッハ標準形へと変形する手順を示す．

3.5.1 チョムスキー標準形への変形

文脈自由文法を，チョムスキー標準形に変形する手順を示す．

1. 既にチョムスキー標準形になっている生成規則はそのまま．
2. チョムスキー標準形になっていない生成規則を，新しく非終端記号を導入することにとってチョムスキー標準形に変形する．

チョムスキー標準形になっていない規則の変形について述べる．チョムスキー標準形は，左辺に一つの非終端記号，右辺に 1 つの終端記号，または 2 つの非終端記号の列からなる規則の集合である．生成規則の中から上述した規則になっていないものに対して新たに非終端記号を導入することでチョムスキー標準形の形に直す．以下の規則があるとする．

$$A \rightarrow aB$$
$$B \rightarrow ABC$$

まず，A の規則について，右辺が終端記号と非終端記号の形になっている．この場合，終端記号 a を非終端記号<a>に置き換え，<a> → a．という規則を新たに追加する．B の規則について，右辺の長さが 3 であり，チョムスキー標準形の形になっていない．この場合も先ほどと同様に，BC をそれに対応した非終端記号<BC>に置き換え，規則に<BC> → BC を追加する．先ほどの規則は以下のように書き換えられる．

$$A \rightarrow \langle a\rangle B$$
$$B \rightarrow A\langle BC\rangle$$
$$\langle a\rangle \rightarrow a$$
$$\langle BC\rangle \rightarrow BC$$

この手順を行うことで，チョムスキー標準形の形に変形することができる．

3.5.2 グライバッハ標準形への変形

チョムスキー標準形からグライバッハ標準形に変形する手順を示す．グライバッハ標準形への変形は，左再帰性の除去と生成規則の置換を行うことで変形させる．

- 生成規則の置換

以下の生成規則があるとする．

$$A \rightarrow B\alpha$$
$$B \rightarrow \beta$$

ここで，A,B は非終端記号，α，β は任意の記号列である．グライバッハ標準形になっていない規則に対して生成規則の置換を行う．上記の式を以下の形に置き換える．

$$A \rightarrow \beta\alpha$$
$$B \rightarrow \beta$$

A→Bγ の B の部分をβ に置き換えを行っている．

- 左再帰性の除去

以下の生成規則があるとする．

$$A \rightarrow A\gamma$$

A は非終端記号，γ は任意の記号列である．上記の生成規則は，右辺の先頭に左辺の終端記号が出現している．このような規則を，左再帰性を持つという．この左再帰性のある規則の除去を行う．

これら 2 つの処理を使い，チョムスキー標準形からグライバッハ標準形に変形する．処理の手順は以下の通りである．

1. 非終端記号に順序付けを行う．
2. 順序付けされた非終端記号を数字が低い順に規則を確認し，右辺の先頭の

方が番号の小さいものを探す

3. ２で発見した生成規則に生成規則の置換を行う.
4. 3 の置換でグライバッハ標準形になったらそのままにし，左再帰性が現れたら左再帰性の除去を行う.
5. 数字の大きい順に生成規則の置換を行いグライバッハ標準形の形に変形する.
6. 左再帰性の除去で導入した非終端記号を左辺にもち，グライバッハ標準刑になっていないものに対して，生成規則の置換でグライバッハ標準形に変形する.

上記の処理を行うことで，チョムスキー標準形からグライバッハ標準形に変形することができる.

第4章 繰り返し学習モデル

Kirby は複数世代にわたる親と子の会話を, 発話と学習を行う二種類のエージェント間のコミュニケーションとしてモデル化し，言語知識を獲得する過程を示した. この繰り返し学習モデルは, 幼児の言語獲得などの研究に用いられている．[4]本章では，3.1 で Kirby の繰り返し学習モデルの概要を述べ，3.2 で学習アルゴリズムについて述べる.

4.1 Kirby の繰り返し学習モデル

Kirby の繰り返し学習モデルは, 複数世代にわたる親と子の二種類のエージェントによる文の受け渡しと学習により，無規則であった言語が構造を持った言語へと進化する過程をシミュレートすることができる（図 3.1）. この時，親と子エージェントは 1 世代に 1 体ずつ存在し，文の受け渡しは，親エージェントから子エージェントへ一方的に送られ，子エージェントは親エージェントから受け取った言語知識を蓄え，蓄えた知識で学習を行い，言語知識を獲得する. 子エージェントは, 初めは何の知識も持たないが, 大人エージェントから受け取った文から言語知識を獲得した後に, 大人エージェントに成長し, 次の世代の子エージェントに自身の言語知識から生成した文の受け渡しを行う.

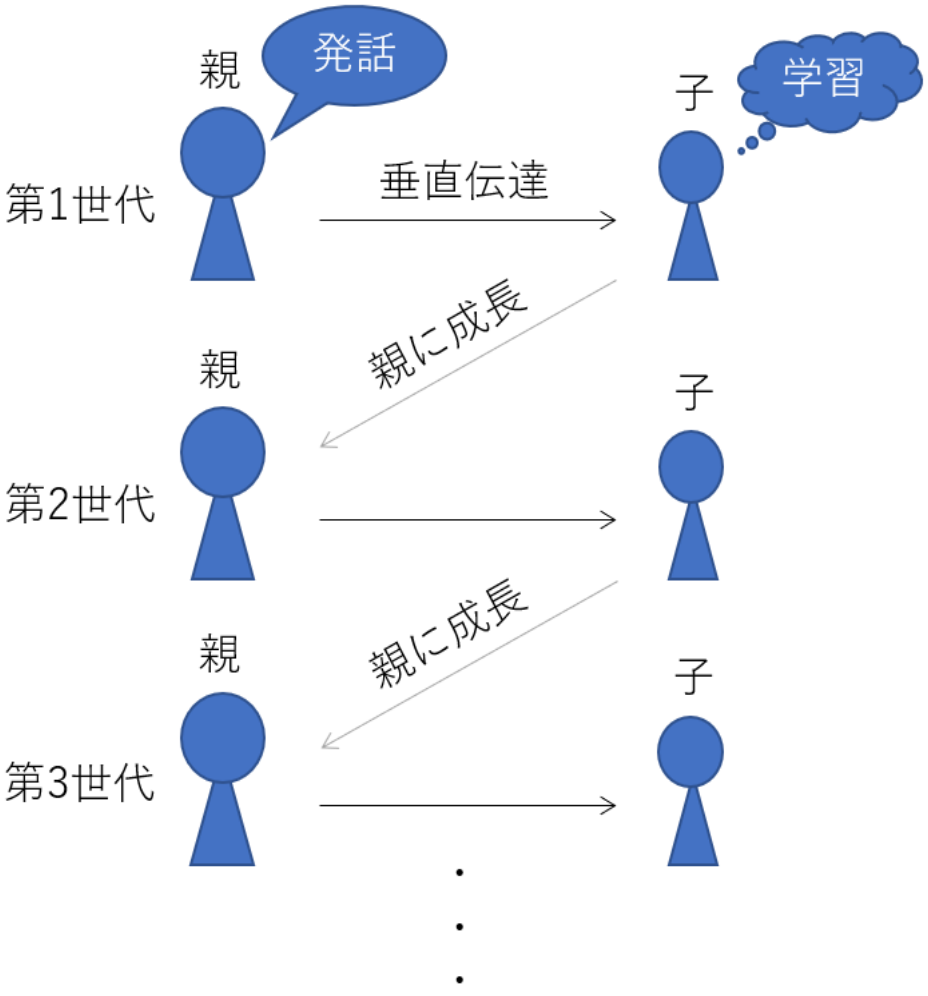

図 4.1 繰り返し学習モデルの概要

4.2 繰り返し学習モデルにおける言語知識

繰り返し学習モデルで用いられる言語知識は，文脈自由文法の終端記号と非終端記号に文法条件を加えた生成規則の集合で構成されている．この規則では，左辺は条件付きの非終端記号となり，生成規則から導出される終端記号の文字列が発話となる．条件は二階述語項構造で表現された以下の式を意味として対応させている．

$$P_i(X_j, X_k) \qquad (j \neq k)$$

ここで，P_i，X_j，X_kはそれぞれ行動，動作主，被動作主にあたり，「ジョンはメアリーを愛している」という意味は live(john,mary)と記述する．この意味に形式を結び付けた以下の式は，「ジョンはメアリーを愛している」という意味を表

す発話が，“johnlovesmary”であることを表現している．

$$\text{love}(john, mary) \rightarrow johnlovesmary$$

親エージェントは，自身の言語知識から文字列を生成し，子エージェントに送る．親エージェントが下記の表 3.1 のような生成規則を持っている場合，生成可能な文字列は，“johnlovesmary”，“johnlikesmary”，“marylivesjohn”，“marylikesjohn”の 4 種類となる．

r_i	発話規則
r_1	$S/love(x_1, x_2) \rightarrow N_1/x_1\ loves\ N_1/x_2$
r_2	$S/like(x_1, x_2) \rightarrow N_1/x_1\ likes\ N_1/x_2$
r_3	$N_1/john \rightarrow john$
r_4	$N_1/mary \rightarrow mary$

表 4.1 親エージェントの言語知識の例

4.3 学習アルゴリズム

子エージェントは，受け取った文から知識を蓄え，汎化学習を行い言語知識の獲得を試みる．学習システムは，chunk,merge,replace の三種類があり，これらの学習を言語知識に変化がなくなるまで繰り返し行う．

4.3.1 chunk

chunk は，二つの規則における共通部分を変数化しカテゴリーにまとめる操作である．chunk により，ルールに新しい非終端記号が追加される．

$$\begin{aligned} \text{S/like(mary,john)} &\rightarrow \text{marylikesjohn} \\ \text{S/love(mary,john)} &\rightarrow \text{marylovesjohn} \\ &\downarrow \text{chunk} \\ \text{S/}X_1\text{(mary.john)} &\rightarrow \text{mary } N_0/X_1 \text{ esjohn} \end{aligned}$$

N_0/like → lik

N_0/lov →lov

“marylikesjohn”と“marylovesjohn”の二つの文字列を比較したとき，“lik”と“lov”の部分が異なる．この時，S/like(mary,john)と S/love(mary,john)の規則の chunk を行う．chunk を行った結果，S/X_1(mary.john)とN_0/like，N_0/lov が規則に追加され，S/like(mary,john)と S/love(mary,john)が削除される．

4.3.2 merge

merge は，生成規則の中から同じ規則を見つけ，一つにまとめる操作である．merge により，非終端記号が統合される．

S/hate(X_2, X_3) → N_1/X_2 hates N_2/X_3

N_1/gavin → gavin

N_2/pete → pete

N_3/gavin → gavin

↓ merge

S/hate(X_2, X_3) → N_1/X_2 hates N_2/X_3

N_1/gavin → gavin

N_2/pete → pete

規則の中で，N_1/gavin とN_3/gavin は同じ文字列を生成する．この場合，これら二つの規則が統合され，一つにまとめられる．また，merge によって統合された規則が他の規則で使用されていた場合，統合先の規則に書き換えられる．

4.3.3 replace

replace は，生成規則の中から，既にある単語の規則を文の規則に適用し，置き換える操作である．replace により，文章の規則を既存の非終端記号に置き換える．

S/admire(john,pete) → johnadmirespete

N_3/admire → admire

↓ replace

$$S/X_1(\text{john,pete}) \rightarrow \text{john } N_3/X_1 \text{ pete}$$
$$N_3/\text{admire} \rightarrow \text{admire}$$

S/admire(john,pete)で生成される文字列の中に，N_3/admire で生成される文字列を発見し，一致部分の変数化を行う．

4.4 表現度

Kirby の繰り返し学習モデルは, 世代交代を繰り返すことで不規則な言語が合成的な規則を含んだ言語に変化する様子を観測することができる．合成的な規則とは, 変数を含んだ意味要素からなる規則のことで, 合成度の高い規則だと, 少ない規則数で表現することのできる文の数が多くなる．

文法規則数が減少し，子の発話できる文が増加していることが世代を追って変化する様子を確認することができる（図 3.1）

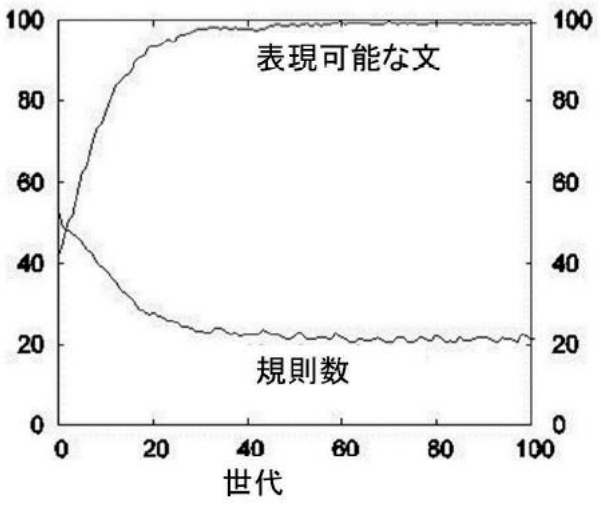

図 4.2 生成可能な表現数と構成規則の推移 [1]

4.5 繰り返し学習モデルと認知バイアス

子供が言語を獲得する速度は速く, 一日に平均 7~15 単語を獲得すると言われている．これは, 言語の学習に認知バイアスと呼ばれる制約が関わっているため

であると指摘されている. 認知バイアスとは, 統計学的な誤りや記憶の誤りなど, 人間が陥りやすい思考の誤りや，思い込みのことである．子供はある対称 A に対して，B というラベルの対応付けを行うと，ラベル B に対して，対象 A を対応付ける．つまり，A ならば B という推論が成立しているとき，同時に B ならば A と考えてしまう．子供に限らず，人間はこのような誤りをしてしまう傾向にあるといい，この認知バイアスを対称性バイアスという．的場らは，Kirby の繰り返し学習モデルをベースに，この対称性バイアスを組み込んだモデルの構築を行った[4]．このモデルの特徴は，文を受け取る子エージェントが，正確に文に対する意味を受け取ることができないことである．親から受け取った文の意味するところを理解していなくても，子エージェント自身がすでにもっている言語知識を用いて, 親から受け取った文の意味を補う様に学習する．この様な学習を

S/p(a,b)　→　abcbcd

という生成規則に対する対称性バイアスであると考え，以下の様に“abcbcd”という文からその意味を補完する学習プロセスを Kirby の繰り返しモデルに組み込む.

????????　←　abcbcd

Kirby の繰り返し学習モデルに，対称性バイアスを適用させた実験を行った．対称性バイアスを用いて学習を行うエージェントは，不明な文を無視するモデルや，ランダムに意味を推論するモデルに比べ, 早い世代で高い表現力を獲得し，少ない規則数の文法規則を構成した．この実験の結果から, 認知バイアスが言語獲得に有用であることを示した.

4.6 繰り返し学習モデルによる楽曲構造の発見

先行研究[2]ではこの Kirby の繰り返し学習をモデルに楽曲構造を発見するモデルの構築を試みている．Kirby の繰り返し学習モデルでは，構造の持つ意味に対応したシンボル列を抽出することで文法構造を獲得していたが，この研究ではシンボル列から意味構造を発見する学習を行っている．このモデルは，Kirby モデルに倣いラベル付きの文脈自由文法扱っており，意味を M，カテゴリーを

C,意味変数を x で表現する.

4.6.1 先行研究の chunk

2 つのルールのシンボル列を比較し，chunk が可能ならば，それを表現する意味構造と文法構造を構築

$$\mathrm{S}/M_1 \quad \rightarrow \quad \text{marylikesjohn}$$
$$\mathrm{S}/M_2 \quad \rightarrow \quad \text{marylovesjohn}$$

↓chunk

$$\mathrm{S}/M_3, x_1 \quad \rightarrow \quad \text{mary } C_1/x_1 \text{ esjohn}$$
$$C_1/M_4 \quad \rightarrow \quad \text{lik}$$
$$C_1/M_5 \quad \rightarrow \quad \text{lov}$$

4.6.2 先行研究の merge

2 つのルールで同じカテゴリー,シンボルである場合，意味とカテゴリーをまとめ，まとめた結果を生成規則全体に適用する.

$$\mathrm{S}/M_1, x_1, x_2 \quad \rightarrow \quad C_1/x_1 \text{ hates } C_2/x_2$$
$$C_1/M_2 \quad \rightarrow \quad \text{kate}$$
$$C_1/M_3 \quad \rightarrow \quad \text{gavin}$$
$$C_2/M_4 \quad \rightarrow \quad \text{kate}$$

↓merge

$$\mathrm{S}/M_1, x_1, x_2 \quad \rightarrow \quad C_3/x_1 \text{ hate } C_3/x_2$$
$$C_3/M_5 \quad \rightarrow \quad \text{kate}$$
$$C_3/M_3 \quad \rightarrow \quad \text{gavin}$$

4.6.3 先行研究の replace

2 つのルールのうち，片方のシンボル列が．他のカテゴリ・シンボルを含んでいれば長いシンボル列の一致部分を置き換える．

$$\mathrm{S}/M_1 \quad \rightarrow \quad \text{johnhatespete}$$
$$C_1/M_2 \quad \rightarrow \quad \text{hate}$$

$$\downarrow \text{replace}$$

$$\mathrm{S}/M_3, x_1 \quad \rightarrow \quad \text{john } C_2/x_1 \text{ spete}$$
$$C_2/M_2 \quad \rightarrow \quad \text{hate}$$

4.6.4 先行研究の楽譜のシンボル化

先行研究では，楽譜にピアノ譜を採用している．ピアノ譜は右手（主旋律）と左手（ベース音）の進行があり，上段と下段に分けられた五線譜が同時に進行する．これら 2 つの進行をシンボル列に変換するため，1 拍の区間に含まれる音を 1 つのシンボルとして表現しており，右手と左手の音符数が不均衡ならば，均等に振り分けてシンボル化している(図 4.3).

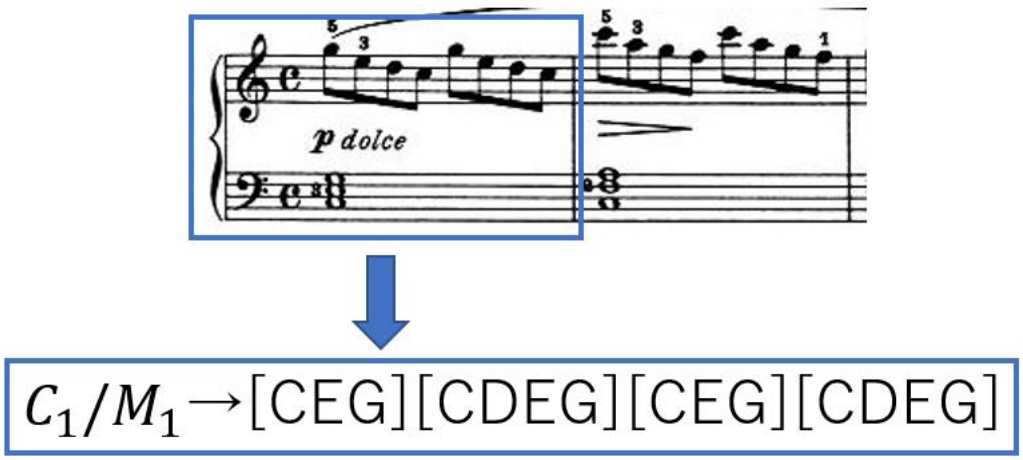

図 4.3 先行研究の楽譜のシンボル列変換

4.6.5 実験結果

このモデルを用いて，自然言語を用いた実験と音楽を用いた実験を行ってい

る．自然言語を用いた実験では，３～４単語の簡単な例文を用いて解析した結果，自然言語の知見に合う結果を得ている．音楽を用いた実験では，ブルグミュラー25 の練習曲を対象に楽曲の楽譜解析を行っている．その結果，カデンツ規則の一部や，協和音から協和音へ回帰を観察することができた．

第5章 繰り返し学習モデルを用いたドラム譜解析

この章では今回の実験に用いた手法について述べる. Kirby の繰り返し学習モデルでは, 拡張された文脈自由文法を言語知識として用いていたが, 今回の手法では文脈自由文法としてモデルを構成した. また, 今回は楽譜として, リズム楽器であるドラムの楽譜であるドラム譜を採用した.

5.1 先行研究の問題点

先行研究では, Kirby の繰り返し学習モデルに倣って, 生成規則に拡張された文脈自由文法を採用し, 意味の概念を扱っている. しかし, 音楽の意味は言語と違い非常に曖昧であり, 音楽の意味を仮定しすぎている. また, 先行研究ではピアノ譜を採用している. ピアノ譜は, 右手（主旋律）と左手（ベース音）がそれぞれ別の五線譜に表現され, 上段と下段に並べられる. 楽譜をシンボル列に変換する際に, 1 拍の区間に含まれている音符を 1 つにまとめてシンボル化を行っており, 1 拍に含まれる音符の順序と音符の高低を考慮していなかった. さらに, 左手の音符が右手の音符に比べて少ない時, 左手の音符を均等に割り振ってシンボル化している. ピアノ譜は, 主旋律や伴奏があり, これら 2 つをまとめて 1 つのシンボルにすることは困難であり, 学習に不適であった. そこで, 今回は楽譜の解析例として, ドラム譜を採用する. ドラム譜は, ピアノ譜よりも単純で, 8 ビートや 16 ビートなどのリズム構造を持っており, 音楽の文法規則の発見を試みるのに適している. 音楽の 3 要素は, メロディー, コード, リズムといわれており, 今回は音楽のリズムに注目して解析を行う. よって, 本研究では, 音楽の意味を取り扱わず, 楽譜としてドラム譜を採用する. 繰り返し学習モデルを構築し, シンボル列からの学習でドラム譜から文法規則の獲得を試みる.

5.2 楽譜の変換

本研究で用いる繰り返し学習モデルは, シンボル列を対象としている. そこで, 繰り返し学習モデルを使用するために楽譜をシンボル列へと変換する必要があ

る．音楽をシンボル列に変換する際に, 時間に関して考える必要があり. まず, 楽譜をどのように区切るか, どの範囲を 1 文字として扱うかである. 音楽は小節の単位で区切られており, 4 小節, 8 小節といった 4 の倍数の小節数で 1 つのフレーズとなることが多い. そのため, 本研究では 4 小節で 1 文として扱った. また, 音楽には拍という概念があり, 4 分の 4 拍子は 1 小節の中に 4 分音符が 4 つ含まれているといった意味である. そこで, この拍の概念を本研究でも取り入れ, 1 拍を 1 文字に変換した. 1 拍は, 4 分音符一つを基準としており, 8 分音符なら 2 つ, 4 分音符なら 4 つで 1 拍となる. 今回は, 音符の順序も考慮し, 1 文字に変換した. また, 本研究では楽譜としてドラム譜を使用した. ドラム譜は, ハイハットやバスドラム, スネアなどのドラムセットで用いられる楽器を五線譜に対応させて楽譜としている. ドラム譜の例を図 4.1 に示す. 1 文字に変換した例を図 4.2 に示す. 図 4.2 のシンボルは, 辞書型で表現され, key がシンボル, value が音符列である.

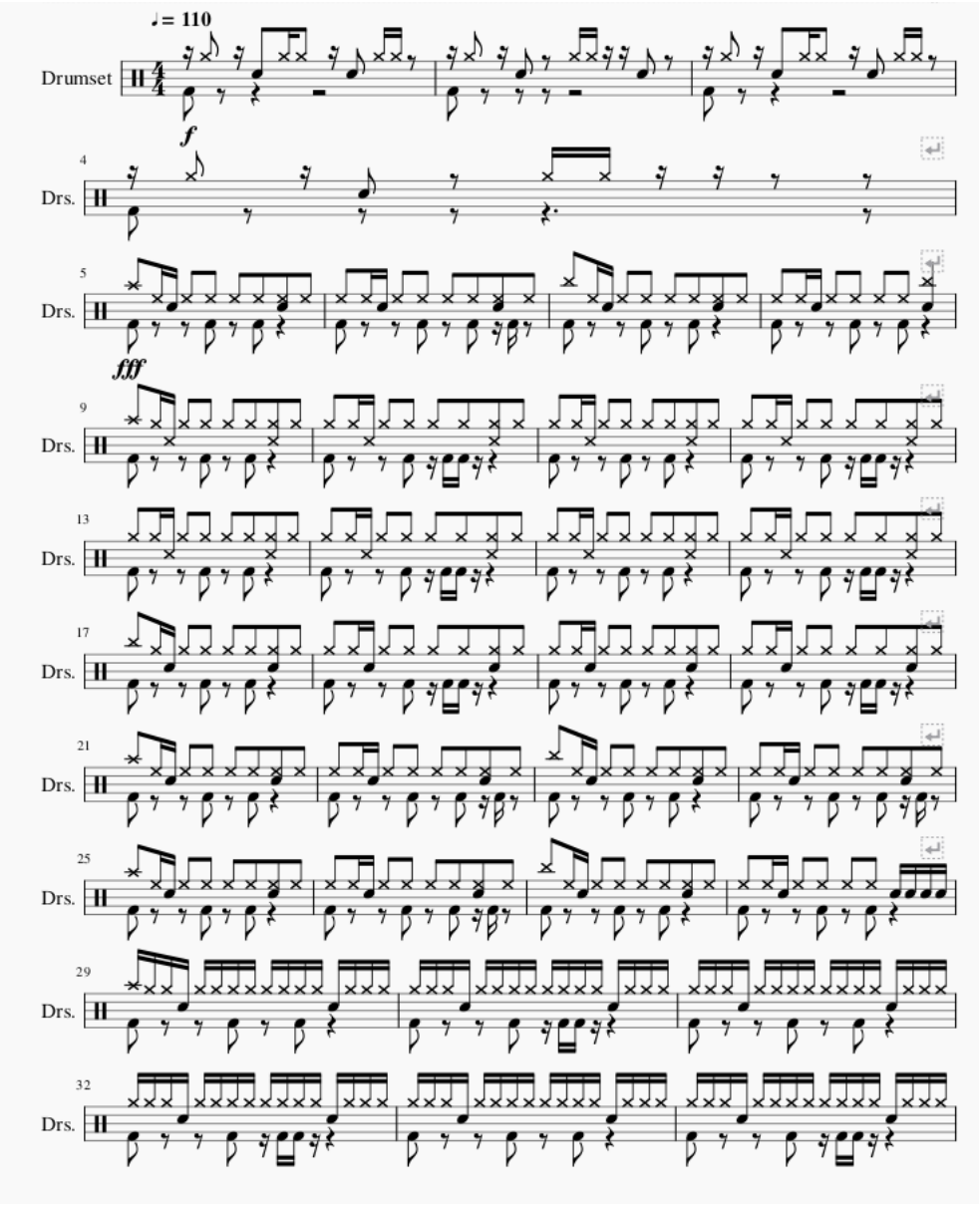

図 5.1 ドラム譜の例

```
{'a': [['BD'], ['HH'], [], []],
 'b': [['S'], [], ['HH'], ['HH']],
 'c': [[], [], ['S'], []],
 'd': [['HH'], ['HH'], [], []],
 'e': [['S'], [], [], []],
 'f': [[], [], [], []],
 'g': [['CC', 'BD'], [], ['HH'], ['S']],
 'h': [['HH'], [], ['HH', 'BD'], []],
 'i': [['S', 'HH'], [], ['HH'], []],
 'j': [['HH', 'BD'], [], ['HH'], ['S']],
 'k': [['S', 'HH'], ['BD'], ['HH'], []],
 'l': [['S', 'CC'], [], [], []],
 'm': [['HH'], ['BD'], ['HH', 'BD'], []],
 'n': [['S'], ['S'], ['S'], ['S']],
 'o': [['CC', 'BD'], ['HH'], ['HH'], ['S']],
 'p': [['HH'], ['HH'], ['HH', 'BD'], ['HH']],
 'q': [['S'], ['HH'], ['HH'], ['HH']],
 'r': [['HH', 'BD'], ['HH'], ['HH'], ['S']],
 's': [['HH'], ['HH', 'BD'], ['HH', 'BD'],
['HH']],
 't': [['S'], ['HH'], ['S'], ['S']],
 'u': [['HH'], [], ['T'], []],
 'v': [['S'], ['S'], ['T'], []],
 'w': [['S'], [], ['HH', 'BD'], []],
 'x': [['S'], [], ['HH'], []],
 'y': [['S'], ['S'], ['HH'], []],
 'z': [['S'], ['T'], ['T'], ['T']],
 '{': [['S', 'HH'], [], ['S', 'HH'], []],
 '|': [['S'], ['HH', 'BD'], ['HH'], ['HH']],
 '}': [['HH', 'BD'], [], ['HH'], []],
 '~': [['HH'], [], [], []],
 '\x7f': [['T'], ['T'], ['T'], ['T']],
 '\x80': [['CC', 'BD'], [], ['HH'], []],
 '\x81': [['HH', 'BD'], [], ['T'], []],
 '\x82': [['S', 'BD'], [], [], []],
 '\x83': [['S', 'BD'], [], ['S'], []],
 '\x84': [['S', 'BD'], ['S'], ['S'], ['S']],
 '\x85': [['T', 'BD'], ['T'], ['T'], ['T']],
 '\x86': [['T', 'BD'], [], ['T'], []]}
```

図 5.2 楽譜のシンボル化

5.3 初期のルール

楽譜から初期の親エージェントに与えるルールを生成する．小節で分割を行い，　1 小節を 1 つの単語として扱う．4 分の 4 拍子を例とすると，1 小節に 4 拍あるので，4 文字で 1 小節となり，小節ごとに初期のルールに追加する．そして，4 小節のまとまりを 1 つの文として文ルールの追加を行う．ルール生成の例

を以下に示す．（図 4.3）

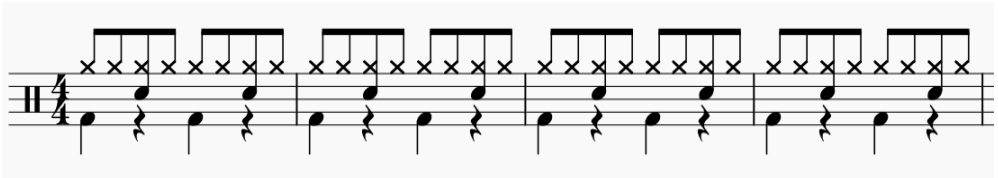

図 5.3 4 小節のドラム譜

↓

$S_0 \rightarrow M_0\ M_1\ M_2\ M_3$

$M_0 \rightarrow$ abab

$M_1 \rightarrow$ abab

$M_2 \rightarrow$ abab

$M_3 \rightarrow$ abab

a : ['HH', 'BD'], [], ['HH'], []

b : ['S', 'HH'], [], ['HH'], []

5.4 繰り返し学習

繰り返し学習モデルを楽譜に適用する．今回の研究で用いた手法では，1 小節 4 文字の 4 小節分を文として扱っており，16 文字の文字列を文として扱っている．そのため，chunk 等の学習を行う際に 1 小節，つまり 4 文字毎に分割を行い，同順序の小節同士での学習を行う．繰り返し学習モデルと同様に chunk, merge，replace の 3 種類の学習アルゴリズムを用いて文から文法を獲得する．

5.4.1 chunk

2 つの文を比較し，一致部分を新しい単語規則として生成規則に追加し，元の文を非終端記号に置き換える．

$S_0 \rightarrow$ abababababababab

$S_1 \rightarrow$ abababababababac

↓ chunk

$S_0 \rightarrow M_0\ M_0\ M_0\ M_1$b

$S_1 \rightarrow M_0\ M_0\ M_0\ M_1$c

$M_0 \rightarrow$ abab

$$M_1 \quad \rightarrow \quad \text{aba}$$

5.4.2 merge

非終端記号の中から，同じ文字列を生成する規則を統合し，統合した結果を生成規則全体に適用する

$$\begin{aligned} S_0 \quad &\rightarrow \quad M_0\ M_2\ M_0\ \text{cdef} \\ M_0 \quad &\rightarrow \quad \text{abab} \\ M_2 \quad &\rightarrow \quad \text{abab} \\ &\downarrow \quad \text{merge} \\ S_0 \quad &\rightarrow \quad M_0\ M_0\ M_0\ \text{cdef} \\ M_0 \quad &\rightarrow \quad \text{abab} \end{aligned}$$

5.4.3 replace

単語規則で生成される文字列が，他の規則で生成される文字列に含まれている場合，一致した部分を非終端記号に置換する．

$$\begin{aligned} S_0 \quad &\rightarrow \quad \text{abababababababac} \\ M_0 \quad &\rightarrow \quad \text{aba} \\ &\downarrow \quad \text{Replace} \\ S_0 \quad &\rightarrow \quad M_0\text{b}\ M_0\text{b}\ M_0\text{b}\ M_0\text{c} \\ M_0 \quad &\rightarrow \quad \text{aba} \end{aligned}$$

楽譜からシンボル列に変換し，初期のルールを作成し，繰り返し学習を適用し複数世代学習を行うことで文法規則の獲得を試みる．実験の流れを図 5.4 に示す．

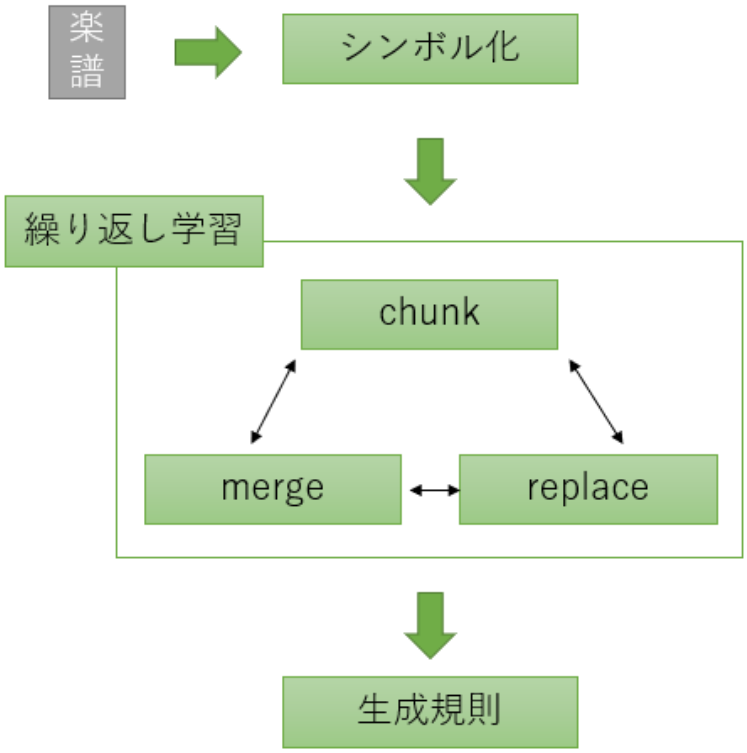

図 5.4 本手法の概略図

第6章 実験

この章では実験の結果と評価について述べる．4 章で述べた，繰り返し学習モデルでの楽譜の解析の結果を，2 章で述べた形式文法の手法を用いて分析行う．楽譜をシンボル列としてとらえ，繰り返し学習モデルを用いて生成規則を取得し，文脈自由文法の変換を行った．

6.1 実験準備

本研究では，繰り返し学習モデルを元にプログラムを作成した．プログラムの作成には Python（3.6）を用いた．楽譜の解析にはドラム譜を用いた．ドラム譜は xml 形式のファイルとなっており，xml ファイルのスクレイピングを行い，シンボル列への変換を行った．楽譜からシンボル列に変換するプログラムの作成も行った．楽譜はウェブサイト https://musescore.com/[9]からダウンロードを行い，9 楽曲を用意した．この楽譜を 4 小節ごとに区切り，初期のルールには 213 文が存在しルールの総数は 1065 個である．

6.2 実験

作成した繰り返し学習モデルで，楽譜の解析を行った． まず，楽譜からシンボルと初期のルールを作成する．次に，初期のルールを親エージェントに与え，文字列を生成し，子エージェントに受け渡す．子エージェントは受け取った文字列を蓄え，蓄えた文字列から chunk,merge,replace の操作を行い学習する．この学習を繰り返し行う．

6.3 実験結果

楽譜から初期のルールを作成し，そのルールを用いて繰り返し学習モデルで学習を行った．学習の結果，始め 1065 個あったルールから，200 個のルールに減少した．10 世代学習をさせたときのルール数の変化を表 5.1 に示す．

世代数	規則数
0	1065
1	201
2	210
3	197
4	203
5	209
6	209
7	201
8	205
9	203
10	200

表 6.1 世代ごとの規則数の変化

次に学習で取得した生成規則の一部を以下に示す.

M87 → ['h', 'M25']
M88 → ['M17', 's']
M89 → ['¡g']
M90 → ['M14', '{~']
M91 → ['gÏ']
M92 → ['M12', 'e']
M93 → ['a ¶ ']
S9173 → [['M33'], ['M42'], ['M42'], ['M12', '±']]
S9525 → [['M42'], ['M42'], ['M42'], ['M14', 'ÁÂ']]
S9702 → [['M46'], ['M47'], ['M46'], ['M47']]
S9751 → [['M33'], ['M42'], ['M27'], ['M14', '¥x9b ´ ']]
S9784 → [['M53'], ['M53'], ['M53'], ['M53']]
S9799 → [['M44'], ['M43'], ['M44'], ['M28', 'Ê']]

6.4 生成規則の変形

実験で取得した生成規則に対して, 変形を行った. 繰り返し学習モデルで取得する規則は文脈自由文法となっており, この文脈自由文法に対して, グライバッハ標準形への変形を行う. 変換を行った結果の一部を以下に示す.

A0 → ['a', 'A201']
A1 → ['a', 'A319']
A2 → ['a', 'A202']
A3 → ['q', 'A320']
A4 → ['q', 'A321']
A5 → ['o', 'A322']
A6 → ['g', 'A323']
A7 → ['¥x84', 'A324']
A8 → ['d', 'A325']
A9 → ['a', 'A326']
A10 → ['d', 'A327']
A11 → ['a', 'A328']
A12 → ['¥x84', 'A205']
A13 → ['¥x84', 'A201']
A14 → ['¥x84', 'A329']
A15 → ['q', 'A330']
A16 → ['d', 'A331']
A17 → ['u', 'A332']
A18 → ['x', 'A215']
A19 → ['¥x95', 'A333']
A20 → ['c', 'A334']

次にグライバッハ標準形になった規則を変形することで次のような規則を得ることができた.

A80 → ['§', '¤', '¥', '¦', 'A379']
A81 → ['a', '¥x9f', '¥x84', 'b', 'A380']
A82 → ['d', '¥x9f', '¥x84', 'b', 'A381']
A83 → ['d', '¥x84', 'b', 'a', 'A382']
A84 → ['x', 'v', 'A383']
A85 → ['c', 'b', '¥x84', 'b', 'A384']
A86 → ['c', 'A385']
A87 → ['o', '}', 'p', 'b', 'A386']
A88 → ['¥x92', 'b', 'c', 'b', 'A387']

A89 → ['c', 'b', '¥x84', 'b', 'A388']
A90 → ['a', 'b', 'c', 'b', 'A389']
A91 → ['¥x82', 'g', 'g', 'g', 'A390']
A92 → ['¥x95', '¥x94', '¥x95', '¥x94', 'A391']
A93 → ['g', 'g', 'g', 'g', 'A392']
A94 → ['¥x83', '¥x84', 'b', '¥x83', 'A393']
A95 → ['a', '¥x9f', 'A394']
A96 → ['¥x83', '¥x84', 'b', '¥x83', 'A395']
A97 → ['d', '¥x84', 'b', 'a', 'A396']
A98 → ['¥x82', 'g', 'g', 'g', 'A397']
A99 → ['a', 'b', 'a', 'b', 'A398']
A100 → ['¥x82', 'g', 'g', 'g', 'A399']

ドラム譜を繰り返し学習モデルで学習させ，取得した規則を変形させた結果，正規文法の形になった．

第7章 おわりに

本研究では，繰り返し学習モデルを基に楽譜を解析するモデルを作成し，ドラム譜を用いて楽譜の解析を行った．このモデルは音楽的な知識を用いず，シンボル列の共通部分から文法の学習を行うモデルである．実験の結果，2 世代目には既に文法規則が収束している．そして，取得した文法規則に対して文脈自由文法の変換を行った．チョムスキー標準形に変形し，グライバッハ標準形への変形を行った．生成規則を変形した結果，ドラム譜から獲得した生成文法は，正規文法程度の制限しか持っていない可能性を示した．

本研究では，ドラム譜の解析にあたり，1 小節ごとに区切った単位での学習を行っている．そのため，ドラムのようなリズム楽器は正規文法であるとまでは言えない．今後は区切る単位を指定せず文法の学習を行うモデルを考案する必要がある．また，　今回は 9 曲のドラム譜を集めることしかできなかったため，少ない文での実験となった．そのため，より多くの楽譜データを集めて解析を行う必要がある．

参考文献

[1] 東条敏, “進化言語学における認知バイアスの有効性,” *人工知能学会全国大会論文集　27: 2D4-0S-03a-4,* 2013.

[2] 須藤洸基, 意味の曖昧さを考慮した音楽・言語構造の繰り返し学習, 北陸先端科学技術大学院大学博士論文, 2019.

[3] 須藤洸基, 東条敏, 進化言語学に基づいた楽譜解析手法の提案, 情報処理学会研究報告, 2018-SLP-120, 2018.

[4] 的場隆一, 中村誠, 東条敏, “構文獲得における対称性バイアスの有効性,” 認知科学 vol.15, no.3(2008).

[5] 東条敏, 平田圭二, 音楽・数学・言語　情報科学が拓く音楽の地平, 近代科学社, 2017.

[6] 富田悦次 , 横森貴, オートマトン・言語理論, 森北出版株式会社, 2015.

[7] 藤田耕司, 岡ノ谷一夫, 進化言語学の構築, 株式会社ひつじ書房, 2012.

[8] ニルス・L・ウォーリン, ビョルン・マーカー, スティーブン・ブラウン著, 山本聡訳, 音楽の起源(上), 株式会社シナノ, 2013.

[9] Musescore, https://musescore.com/.

謝辞

本研究を進めるにあたり，未熟な私に対し，熱心にご指導・ご助言をいただきました東条敏教授に深く感謝いたします.

また，審査を引き受けてくださいました，白井清昭准教授，Nguyen,Minh Le 准教授，岡田将吾准教授に深く感謝いたします.

最後になりましたが，助言や協力をいただきました東条研究室の皆様に厚く御礼申し上げます.

付録A.　プログラムのソースコード

本研究で作成したプログラムのソースコードを以下に示す.

・MusicILM.py

```
# -*- coding: utf-8 -*-
"""
Created on Tue Oct  8 18:33:27 2019

@author: mittyonb
"""

import os
import copy
import random
from split_score import get_score
"""
chunk
    1: abcdefg
    2: abcedfg
    -> XdeY
        XedY
"""
#class
# 親エージェント
class agentA():
    def __init__(self,r):
        self.rule = r

    def generate_symbol(self):
        symbol = []
        key = [k for k in list(self.rule.keys()) if k[0] == "S"]
        if key != []:
            sk = random.choices(key)[0]
            symbol = copy.deepcopy(self.rule[sk])
            for i in range(len(symbol)):
                if type(symbol[i]) != type([]):
                    if symbol[i][0] == "M":
                        if type(self.rule[symbol[i]]) == type([]) :
                            symbol[i]                                                    =
copy.deepcopy(random.choice(self.rule[symbol[i]]))
                        else:
```

```
                                    symbol[i]                                   =
copy.deepcopy(self.rule[symbol[i]])
                    elif type(symbol[i]) == type([]):
                        c = ""
                        for s in symbol[i]:
                            c += self.check_rule(s)
                        symbol[i] = c
        return "".join(symbol)

    def check_rule(self,symbol):
        if symbol[0] == "M":
            x = self.rule[symbol]
            if type(x) == type(""):
                return x
            elif type(x) == type([]):
                c = []
                for s in x:
                    if s[0] == "M":
                        c.append(self.rule[s][0])
                    else:
                        c.append(s)
                return "".join(c)
        else:
            return symbol

# 子エージェント
class agentB():
    def __init__(self):
        self.memory = []
        self.rule = {}
        self.si = 0
        self.mi = 0
    #Learn algorythm
    def learn(self):
        if self.si == 0:
            for s in self.memory:
                if s not in list(self.rule.values()):
                    self.rule["S" + str(self.si)] = s
                    self.si += 1
        if len(self.rule) > 1:
            self.count = 0
            self.chunk()
            self.merge()
            if self.mi > 0:
                self.replace()
```

```
    def chunk(self):
        s_keys = [k for k in list(self.rule.keys()) if k[0] == "S"]
        if len(s_keys) > 1:
            key = random.sample(s_keys,2)
        else:
            return
        k1 = key[0]
        k2 = key[1]
        result1 = []
        result2 = []
        same_list = []
        str1 = self.rule[k1]
        str2 = self.rule[k2]
        if not (type(str1) == type([]) and type(str2) == type([])):
            if type(str1) == type(""):
                str1 = list(split_list(str1,4))
            if type(str2) == type(""):
                str2 = list(split_list(str2,4))
            for i in range(len(str1)):
                if type(str1[i]) == type("") and type(str2[i]) == type(""):
                    same_list.append(self.same_str(str1[i],str2[i]))
                else:
                    same_list.append([])
            result1,result2 = self.chunk_str(str1,str2,same_list)
            z = 0
            for i in range(len(same_list)):
                if same_list[i] == []:
                    z += 1
            if z < 8:
                self.rule.pop(k1)
                self.rule.pop(k2)
                self.rule["S"+str(self.si)] = result1
                self.si += 1
                self.rule["S"+str(self.si)] = result2
                self.si += 1
        elif type(str1) == type([]) and type(str2) == type([]):
            for i in range(len(str1)):
                if type(str1[i]) == type("") and type(str2[i]) == type(""):
                    same_list.append(self.same_str(str1[i],str2[i]))
                elif (len(str1[i]) == 1 and len(str2[i]) == 1) and
(str1[i][0][0] != "M" and str2[i][0][0] != "M"):
                    same_list.append(self.same_str(str1[i][0],str2[i][0]))
                else:
                    same_list.append([])
```

```
                result1,result2 = self.chunk_str(str1,str2,same_list)
                z = 0
                for i in range(len(same_list)):
                    if same_list[i] == []:
                        z += 1
                if z < 8:
                    self.rule.pop(k1)
                    self.rule.pop(k2)
                    self.rule["S"+str(self.si)] = result1
                    self.si += 1
                    self.rule["S"+str(self.si)] = result2
                    self.si += 1

    def chunk_str(self,str1,str2,same_list):
        result1 = []
        result2 = []
        for i in range(len(same_list)):
            if type(str1[i]) == type("") and type(str2[i]) == type(""):
                r1 = list(copy.deepcopy(str1[i]))
                r2 = list(copy.deepcopy(str2[i]))
            else:
                r1 = copy.deepcopy(str1[i])
                r2 = copy.deepcopy(str2[i])
            if same_list[i] == [0,1]:
                self.rule["M" + str(self.mi)] =
list(set([str1[i][0]+str1[i][1],str2[i][0]+str2[i][1]]))
                r1[0] = "-M" + str(self.mi) + "-"
                del r1[1]
                r2[0] = "-M" + str(self.mi) + "-"
                del r2[1]
                self.mi += 1
                r1 = [x for x in ("".join(r1)).split("-") if x != ""]
                r2 = [x for x in ("".join(r2)).split("-") if x != ""]
            elif same_list[i] == [1,2]:
                self.rule["M" + str(self.mi)] =
list(set([str1[i][1]+str1[i][2],str2[i][1]+str2[i][2]]))
                r1[1] = "-M" + str(self.mi) + "-"
                del r1[2]
                r2[1] = "-M" + str(self.mi) + "-"
                del r2[2]
                self.mi += 1
                r1 = [x for x in ("".join(r1)).split("-") if x != ""]
                r2 = [x for x in ("".join(r2)).split("-") if x != ""]
            elif same_list[i] == [2,3]:
                self.rule["M" + str(self.mi)] =
```

```
list(set([str1[i][2]+str1[i][3],str2[i][2]+str2[i][3]]))
                    r1[2] = "-M" + str(self.mi) + "-"
                    del r1[3]
                    r2[2] = "-M" + str(self.mi) + "-"
                    del r2[3]
                    self.mi += 1
                    r1 = [x for x in ("".join(r1)).split("-") if x != ""]
                    r2 = [x for x in ("".join(r2)).split("-") if x != ""]
                elif same_list[i] == [0,1,2]:
                    self.rule["M"              +              str(self.mi)]              =
list(set([str1[i][0]+str1[i][1]+str1[i][2],str2[i][0]+str2[i][1]+str2[i][2]]))
                    r1[0] = "-M" + str(self.mi) + "-"
                    del r1[1]
                    del r1[1]
                    r2[0] = "-M" + str(self.mi) + "-"
                    del r2[1]
                    del r2[1]
                    self.mi += 1
                    r1 = [x for x in ("".join(r1)).split("-") if x != ""]
                    r2 = [x for x in ("".join(r2)).split("-") if x != ""]
                elif same_list[i] == [1,2,3]:
                    self.rule["M"              +              str(self.mi)]              =
list(set([str1[i][1]+str1[i][2]+str1[i][3],str2[i][1]+str2[i][2]+str2[i][3]]))
                    r1[1] = "-M" + str(self.mi) + "-"
                    del r1[2]
                    del r1[2]
                    r2[1] = "-M" + str(self.mi) + "-"
                    del r2[2]
                    del r2[2]
                    self.mi += 1
                    r1 = [x for x in ("".join(r1)).split("-") if x != ""]
                    r2 = [x for x in ("".join(r2)).split("-") if x != ""]
                elif same_list[i] == [0,2,3]:
                    self.rule["M"              +              str(self.mi)]              =
list(set([str1[i][2]+str1[i][3],str2[i][2]+str2[i][3]]))
                    r1[2] = "-M" + str(self.mi) + "-"
                    del r1[3]
                    r2[2] = "-M" + str(self.mi) + "-"
                    del r2[3]
                    self.mi += 1
                    r1 = [x for x in ("".join(r1)).split("-") if x != ""]
                    r2 = [x for x in ("".join(r2)).split("-") if x != ""]
                elif same_list[i] == [0,1,3]:
                    self.rule["M"              +              str(self.mi)]              =
list(set([str1[i][0]+str1[i][1],str2[i][0]+str2[i][1]]))
```

```
                    r1[0] = "-M" + str(self.mi) + "-"
                    del r1[1]
                    r2[0] = "-M" + str(self.mi) + "-"
                    del r2[1]
                    self.mi += 1
                    r1 = [x for x in ("".join(r1)).split("-") if x != ""]
                    r2 = [x for x in ("".join(r2)).split("-") if x != ""]
                elif same_list[i] == [0,1,2,3]:
                    self.rule["M" + str(self.mi)] = [str1[i][0]+str1[i][1]+str1[i][2]+str1[i][3]]
                    r1[0] = "-M" + str(self.mi) + "-"
                    del r1[1]
                    del r1[1]
                    del r1[1]
                    r2[0] = "-M" + str(self.mi) + "-"
                    del r2[1]
                    del r2[1]
                    del r2[1]
                    self.mi += 1
                    r1 = [x for x in ("".join(r1)).split("-") if x != ""]
                    r2 = [x for x in ("".join(r2)).split("-") if x != ""]
                elif same_list[i] != [] and len(same_list[i]) < 4:
                    r1 = "".join(r1)
                    r2 = "".join(r2)
                elif type(str1[i]) == type("") and type(str2[i]) == type(""):
                    r1 = "".join(r1)
                    r2 = "".join(r2)
                elif type(str1[i]) == type([]) and type(str2[i]) == type([]):
                    if str1[i] == str2[i] and len(str1[i]) > 1:
                        self.rule["M" + str(self.mi)] = str1[i]
                        r1 = ["M" + str(self.mi)]
                        r2 = ["M" + str(self.mi)]
                        self.mi += 1
                result1.append(r1)
                result2.append(r2)
        return result1 , result2

    def chunk_list(self,l1,l2):
        same = []
        if len(l1) == len(l2):
            for i in range(len(l1)):
                if l1[i] == l2[i]:
                    same.append(i)
        if len(same) > 1:
            if same == [0,1]:
```

```
                print(l1)
                print(l2)
                print("########")
                self.rule["M" + str(self.mi)] = [l1[0],l1[1]]
                l1[0] = "M" + str(self.mi)
                l2[0] = "M" + str(self.mi)
                del l1[1]
                del l2[1]
                self.mi += 1
                print(l1)
                print(l2)
            elif same == [1,2]:
                self.rule["M" + str(self.mi)] = [l1[1],l1[2]]
                l1[1] = "M" + str(self.mi)
                l2[1] = "M" + str(self.mi)
                del l1[2]
                del l2[2]
                self.mi += 1
            elif same == [2,3]:
                self.rule["M" + str(self.mi)] = [l1[2],l1[3]]
                l1[2] = "M" + str(self.mi)
                l2[2] = "M" + str(self.mi)
                del l1[3]
                del l2[3]
                self.mi += 1
        return l1,l2

    def merge(self):
        same_rule = []
        key = [k for k in self.rule.keys() if k[0] == "S"]
        rules = [d for d in self.rule.items()]
        for r in rules:
            x = []
            for i in range(len(rules)):
                if r[1] == rules[i][1]:
                    x.append(rules[i])
            if len(x) > 1 and x not in same_rule:
                same_rule.append(x)
        for sr in same_rule:
            if sr[0][0][0] == "M":
                for r in sr:
                    va = r[1]
                    for k in key:
                        for i in range(len(self.rule[k])):
                            if type(self.rule[k][i]) == type("") and
```

```
self.rule[k][i] == r[0]:
                                        self.rule[k][i] = "M"+str(self.mi)
                                    elif type(self.rule[k][i]) == type([]):
                                        for j in range(len(self.rule[k][i])):
                                            if self.rule[k][i][j] == r[0]:
                                                self.rule[k][i][j]                    =
"M"+str(self.mi)
                            mk = [k for k in self.rule.keys() if k[0] == "M"]
                            for k in mk:
                                if type(self.rule[k]) == type([]):
                                    for i in range(len(self.rule[k])):
                                        if r[0] == self.rule[k][i]:
                                            self.rule[k][i] = "M"+str(self.mi)
                                self.rule.pop(r[0])
                        self.rule["M"+str(self.mi)] = va
                        self.mi += 1
            rules = [d for d in self.rule.items()]
            same_rule = []
            for r in rules:
                x = []
                for i in range(len(rules)):
                    if r[1] == rules[i][1]:
                        x.append(rules[i])
                if len(x) > 1 and x not in same_rule:
                    same_rule.append(x)
            for sr in same_rule:
                if sr[0][0][0] == "S":
                    for r in sr:
                        va = r[1]
                        self.rule.pop(r[0])
                    self.rule["S"+str(self.si)] = va
                    self.si += 1

        def replace(self):
            mk = random.choice([k for k in list(self.rule.keys()) if k[0] == "M"])
            m = copy.deepcopy(self.rule[mk])
            if len(m) == 1:
                m = m[0]
            s_key = [k for k in list(self.rule.keys()) if k[0] == "S"]
            for i in range(len(s_key)):
                if type(self.rule[s_key[i]]) != type(""):
                    if type(m) == type(""):
                        for j in range(len(self.rule[s_key[i]])):
                            if type(self.rule[s_key[i]][j]) == type([]):
                                for n in range(len(self.rule[s_key[i]][j])):
```

```
                                    if m in self.rule[s_key[i]][j][n] and type(self.rule[s_key[i]][j][n]) != type([]):
                                        self.rule[s_key[i]][j][n] = self.rule[s_key[i]][j][n].replace(m,"-" + mk + "-")
                                        self.rule[s_key[i]][j] = [s for s in "-".join(self.rule[s_key[i]][j]).split("-") if s]
                                elif type(self.rule[s_key[i]][j]) == type(""):
                                    if m in self.rule[s_key[i]][j]:
                                        self.rule[s_key[i]][j] = self.rule[s_key[i]][j].replace(m,"-" + mk + "-")
                                        self.rule[s_key[i]][j] = self.rule[s_key[i]][j].split("-")
                                        self.rule[s_key[i]][j] = [s for s in self.rule[s_key[i]][j] if s]
                        elif type(m) == type([]):
                            for j in range(len(self.rule[s_key[i]])):
                                if type(self.rule[s_key[i]][j]) == type([]):
                                    if m == self.rule[s_key[i]][j]:
                                        self.rule[s_key[i]][j] = [mk]
                                elif type(self.rule[s_key[i]][j]) == type(""):
                                    pass


    def same_str(self,A,B):
        same = []
        for i in range(len(A)):
            if A[i] == B[i]:
                same.append(i)
        return same

    def same_list(self,A,B):
        same = []
        for i in range(len(A)):
            if A[i] == B[i]:
                same.append(i)
        return same

# 初期のルール
def init_rule(score_notes,symbol):
    i = 0
    x = 0
    y = 0
    ini_rule = {}
    measure = []
    for s in score_notes:
```

```
            for m in s:
                c = ""
                note = list(split_list(m,int(len(m)/4)))
                for n in note:
                    value = symbol.values()
                    if n not in value :
                        symbol[chr(ord("a") + y)] = n
                        y += 1
                key = symbol.keys()
                for n in note:
                    for k in key:
                        if n == symbol[k]:
                            c += k
                if c != "":
                    ini_rule["M"+str(i)] = c
                    measure.append("M" + str(i))
                    i+=1
                if len(measure) == 4:#n 小節で分割
                    ini_rule["S"+str(x)] = measure
                    x += 1
                    measure = []
        return ini_rule

def split_list(l, n):
        for idx in range(0, len(l), n):
            yield l[idx:idx + n]

#iterated learning model
def gene(pre_agent_rule):
        agent1 = agentA(pre_agent_rule)
        agent2 = agentB()
        n = 0
        n_pre_rule = {}# 前の学習結果を保持
        symbol = ""
        for _ in range(10000):
            symbol = agent1.generate_symbol()
            agent2.memory.append(symbol)
        while(n < 5000):# ルールに変化がなくなるまで学習
            n_pre_rule = agent2.rule
            agent2.learn()
            if n_pre_rule == agent2.rule : # ルールに変化がないか確認
                n += 1
            else:
```

```
                n = 0
    return agent2.rule

if __name__ == "__main__":
    g = 10          # 学習する世代数
    score_list = os.listdir("score/")
    symbol_dict = {}
    notes = []
    for i in range(len(score_list)):
        notes.append(get_score(score_list[i],i+1))
    rule = init_rule(notes,symbol_dict)
    first_rule = rule
    n_rule = [len(rule)]# ルール数を保存
    print("start ILM")
    for i in range(g):# n 世代学習
        print("------------------ gene " + str(i+1) + "/" + str(g) + " ------------------
-------")
        rule = gene(rule)   #学習
        n_rule.append(len(rule))
```

・split_score.py

```
# -*- coding: utf-8 -*-
"""
Created on Thu Dec 12 01:37:51 2019

@author: mittyonb
"""
from bs4 import BeautifulSoup

def inst_check(soup,symbol): #used instrument
    return {m["id"] : symbol[m.find("instrument-name").text] for m in
soup.find_all("score-instrument")}

def extract_music(soup,inst_list,score):
    m_list = []
    p_list = {p["id"] : p.find("part-name").string for p in
soup.find_all("score-part")}
    for p in soup.find_all("part"):
        duration = int(soup.find("attributes").divisions.string) * 4
        if p_list[p["id"]] == "Drumset" or p_list[p["id"]] == "Bateria" or
p_list[p["id"]] == "Batteria" or p_list[p["id"]] == "Schlagzeug":
            for m in p.find_all("measure"):
                n_list = [[] for _ in range(duration)]
```

```
                    cur_time = 0 # Current time
                    tmp_duration = 0
                    for nb in m.find_all({"note", "backup"}):
                        if nb.name == "backup": # 巻き戻し
                            cur_time -= int(nb.duration.string)
                        if nb.name == "note":
                            if not nb.chord: # 和音でなければ
                                cur_time += tmp_duration
                            if nb.unpitched:#音符を追加
                                if n_list[cur_time] == ["rest"]:
                                    n_list[cur_time][0] =
inst_list[nb.find("instrument")["id"]]
                                else:

n_list[cur_time].append(inst_list[nb.find("instrument")["id"]])
                            if nb.rest:
                                if nb.type:
                                    pass
                                else:
                                    if n_list[cur_time] == []:
                                        n_list[cur_time].append("rest")
                            if nb.duration: # 装飾音は duration ないので飛ば
す
                                tmp_duration = int(nb.duration.string)
                    m_list.append(n_list)
    return m_list

def get_score(filename,n):
    xml_name = ".¥score¥¥" + filename
    print(xml_name)
    soup = BeautifulSoup(open(xml_name,'r').read(), "lxml")
    symbol = {'Acoustic Bass Drum':"BD",'Bass Drum 1':"BD",'Side
Stick':"S",'Acoustic Snare':"S",'Electric Snare':"S",
        'Low Floor Tom':"T",'Closed Hi-Hat':"HH",'High Floor
Tom':"T",'Pedal Hi-Hat':"HH",'Low Tom':"T",
        'Open Hi-Hat':"HH",'Low-Mid Tom':"T",'Hi-Mid Tom':"T",'Crash
Cymbal 1':"CC",'High Tom':"T",
        'Ride Cymbal 1':"RC",'Chinese Cymbal':"CC",'Ride
Bell':"Bell",'Tambourine':"HH",'Splash Cymbal':"CC",
        'Cowbell':"Cowbell",'Crash Cymbal 2':"CC",'Ride Cymbal
2':"RC","Open Hi Conga":"OHC","Low Conga":"LC","Piano":"P",
        'Bumbo 1':"Bumbo","Bass Drum":"BD","Snare
(Acoustic)":"S","Snare (Electric)":"S","Tom 5":"T","Hi-Hat Closed":"HH",
```

```
            "Tom 4":"T","Hi-Hat Pedal":"HH","Tom 3":"T","Hi-Hat Open":"HH","Tom 2":"T","Tom 1":"T","Crash 1":"CC","Tom":"T",
            "Ride":"RC","China":"CC","Ride (Bell)":"Bell","open high conga":"OHC","low conga":"LC"}
    inst_list = inst_check(soup,symbol)
    return extract_music(soup,inst_list,n)
```

・cfg2gbn.py

```
# -*- coding: utf-8 -*-
"""
Created on Thu Jan  2 12:41:29 2020

@author: mittyonb
"""

#CFG -> CNF -> GNF
import copy
import collections

def flatten_view(nested_list):
    for element in nested_list:
        if isinstance(element, collections.Iterable) and not isinstance(element, str):
            yield from flatten_view(element)
        else:
            yield element

def split_list(l, n):
    for idx in range(0, len(l), n):
        yield l[idx:idx + n]


def del_invalid_sign(rule):
    symbol_list = {}
    keys = list(rule.keys())
    for k in keys:
        if len(rule[k]) == 1 :
            symbol_list[k] = rule[k][0]
            del rule[k]
        elif type(rule[k]) == type(""):
            symbol_list[k] = rule[k]
            del rule[k]
    symbol_k = list(symbol_list.keys()) #--
    keys = list(rule.keys())
    for k in [k for k in keys if k[0]=="M"]:
```

```
            flag = 1
            for r in rule[k]:
                if r[0] == "M" and r not in symbol_k:
                    flag = 0
            if flag:
                symbol_list[k] = rule[k]
                del rule[k]
    symbol_k = list(symbol_list.keys()) #--
    keys = list(rule.keys())
    for k in keys:
        flag = 1
        for r in rule[k]:
            for s in r:
                if s[0] == "M" and s not in symbol_k:
                    flag = 0
        if flag:
            symbol_list[k] = rule[k]
    s_key = [k for k in symbol_list.keys() if k[0] == "S"]
    check_list = []
    for k in s_key:
        for r in symbol_list[k]:
            for s in r:
                if s[0] == "M":
                    check_list.append(s)
    check_list = list(set(check_list))
    check_key = [k for k in symbol_list.keys() if k[0] == "M"]
    for c in check_list:
        if c not in check_key:
            del symbol_list[c]
    return symbol_list

def remove_Epsilon(rule):
    keys = [k for k in rule.keys()]
    m_key = [k for k in rule.keys() if k[0] == "M"]
    del_symbol = []
    for k in keys:
        if rule[k] == []:
            del_symbol.append(k)
    del_symbol = list(set(del_symbol))
    for _ in range(20):
        for k in m_key:
            if rule[k] in del_symbol:
                del_symbol.append(k)
    for d in del_symbol:
        del rule[d]
```

```
    return rule

def cfg_to_cnf(rule):
    n = 0
    symbol = []
    keys = [k for k in rule.keys()]
    m_key = [k for k in rule.keys() if k[0] == "M"]
    for k in keys:
        if type(rule[k]) == type(""):
            if len(rule[k]) > 1:
                s = list(rule[k])
                for i in range(len(s)):
                    if s[i] not in symbol:
                        symbol.append(s[i])
                        rule["N" + str(n)] = s[i]
                        s[i] = "N" + str(n)
                        n+=1
        elif type(rule[k]) == type([]):
            for i in range(len(rule[k])):
                if type(rule[k][i]) == type([]):
                    for j in range(len(rule[k][i])):
                        if ord(rule[k][i][j][0]) > 96:
                            s = list(rule[k][i][j])
                            for l in range(len(s)):
                                if s[l] not in symbol:
                                    symbol.append(s[l])
                                    rule["N" + str(n)] = s[l]
                                    s[l] = "N" + str(n)
                                    n+=1
                elif type(rule[k][i]) == type(""):
                    if ord(rule[k][i][0]) > 96:
                        s = list(rule[k][i])
                        for j in range(len(s)):
                            if s[j] not in symbol:
                                symbol.append(s[j])
                                rule["N" + str(n)] = s[j]
                                s[j] = "N" + str(n)
                                n+=1
    n_key = [k for k in rule.keys() if k[0] == "N"]
    for nk in n_key:
        for k in keys:
            if type(rule[k]) == type(""):
                s = list(rule[k])
                for i in range(len(s)):
                    if s[i] == rule[nk]:
```

```
                                s[i] = nk
                        rule[k] = s
                elif type(rule[k]) == type([]):
                        for i in range(len(rule[k])):
                                if type(rule[k][i]) == type("") and ord(rule[k][i][0]) >
96:
                                        s = list(rule[k][i])
                                        for j in range(len(s)):
                                                if s[j] == rule[nk]:
                                                        s[j] = nk
                                        if len(s) == 1:
                                                rule[k][i] = s[0]
                                        else:
                                                s = list(flatten_view(s))
                                                rule[k][i] = s
                                                rule[k] = list(flatten_view(rule[k]))
                                elif type(rule[k][i]) == type([]):
                                        r = copy.deepcopy(rule[k][i])
                                        for j in range(len(r)):
                                                flag = 0
                                                if ord(r[j][0]) > 96:
                                                        s = list(r[j])
                                                        if len(s) == 1:
                                                                if s[0] == rule[nk]:
                                                                        s = nk
                                                        else:
                                                                for l in range(len(s)):
                                                                        if s[l] == rule[nk]:
                                                                                s[l] = nk
                                                                        flag = 1
                                                        if flag:
                                                                r[j] = s
                                                                r = list(flatten_view(r))
                                                        else:
                                                                r[j] = s
                                                                r = list(flatten_view(r))
                                        rule[k][i] = r
        for k in rule.keys():
                if type(rule[k]) != type(""):
                        rule[k] = list(flatten_view(rule[k]))
        variable = []
        append = []
        m = 0
        for _ in range(1000):
                keys = [k for k in rule.keys()]
```

```
            for k in keys:
                if len(rule[k]) > 2:
                    while("M"+str(m) in m_key):
                        m += 1
                    if rule[k][1:] not in variable:
                        rule["M"+str(m)] = rule[k][1:]
                        rule[k] = [rule[k][0],"M"+str(m)]
                        variable.append(rule[k][1:])
                        append.append("M"+str(m))
                        m += 1
            for a in append:
                for k in m_key:
                    if len(rule[k]) > 2:
                        if rule[k][1:] == rule[a]:
                            rule[k] = [rule[k][0],a]
        return rule

def cnf_to_gnf(rule):
        keys = [k for k in rule.keys()]
        new_rule = {}
        rule_taple = []
        a = 0
        keys = [k for k in keys]
        for k in keys:
            new_rule["A" + str(a)] = rule[k]
            rule_taple.append((k,"A"+str(a)))
            a += 1

        for k in [k for k in new_rule.keys()]:
            if type(new_rule[k]) == type([]):
                r = copy.deepcopy(new_rule[k])
                for i in range(len(r)):
                    for t in rule_taple:
                        if r[i] == t[0]:
                            r[i] = t[1]
                new_rule[k] = r
        keys = [k for k in new_rule.keys()]
        l = len(keys)
        for _ in range(100):
            for i in range(l):
                if len(new_rule["A"+str(i)]) > 1:
                    if new_rule["A"+str(i)][0][0] != "A":
                        pass
                    elif                int(("A"+str(i)).split("A")[1])                >
int(new_rule["A"+str(i)][0].split("A")[1]):
```

```
                        s = copy.deepcopy(new_rule["A"+str(i)])
                        s[0] =
copy.deepcopy(new_rule[new_rule["A"+str(i)][0]])
                        z = []
                        for x in s:
                            if type(x) == type([]):
                                for y in x:
                                    z.append(y)
                            elif type(x) == type(""):
                                z.append(x)
                        new_rule["A"+str(i)] = z
    keys = [k for k in new_rule.keys()]
    l = len(keys)
    value = new_rule.values()
    for _ in range(100):
        for i in range(l-1,-1,-1):
            if len(new_rule["A"+str(i)]) > 1:
                if new_rule["A"+str(i)][0] not in value:
                    s = copy.deepcopy(new_rule["A"+str(i)])
                    s[0] =
copy.deepcopy(new_rule[new_rule["A"+str(i)][0]])
                    z = []
                    for x in s:
                        if type(x) == type([]):
                            for y in x:
                                z.append(y)
                        elif type(x) == type(""):
                            z.append(x)
                    new_rule["A"+str(i)] = z
    return new_rule

def organizing_gnf(rule):#グライバッハ標準形の展開
    keys = [k for k in rule.keys()]
    for k in keys:
        flag = 1
        if len(rule[k]) == 3:
            i = 1
            for _ in range(1000):
                flag2 = 0
                if len(rule[k][i]) == 1:
                    i+=1
                else:
                    s = copy.deepcopy(rule[k])
                    s[i] = copy.deepcopy(rule[rule[k][i]])
                    if len(rule[rule[k][i]]) == 1:
```

```
                    flag2 = 1
                z = []
                for x in s:
                    if type(x) == type([]):
                        for y in x:
                            z.append(y)
                    elif type(x) == type(""):
                        z.append(x)
                rule[k] = z
                if flag2:
                    break
                else:
                    i+= 1
        if len(rule[k][-2]) == 1:
            flag = 0
        while(flag):
            rule[k] = z
            s = copy.deepcopy(rule[k])
            s[-2] = copy.deepcopy(rule[rule[k][-2]])
            z = []
            for x in s:
                if type(x) == type([]):
                    for y in x:
                        z.append(y)
                elif type(x) == type(""):
                    z.append(x)
            rule[k] = z
            if rule[k][-2][0] != "A":
                flag = 0
    if len(rule[k]) == 4 and rule[k][-2][0] == "A":
        i = 1
        for _ in range(1000):
            flag2 = 0
            s = copy.deepcopy(rule[k])
            s[i] = copy.deepcopy(rule[rule[k][i]])
            if len(rule[rule[k][i]]) == 1:
                flag2 = 1
            z = []
            for x in s:
                if type(x) == type([]):
                    for y in x:
                        z.append(y)
                elif type(x) == type(""):
                    z.append(x)
            rule[k] = z
```

```
                    if flag2:
                        break
                    else:
                        i+= 1
                while(flag):
                    rule[k] = z
                    s = copy.deepcopy(rule[k])
                    s[-2] = copy.deepcopy(rule[rule[k][-2]])
                    z = []
                    for x in s:
                        if type(x) == type([]):
                            for y in x:
                                z.append(y)
                        elif type(x) == type(""):
                            z.append(x)
                    rule[k] = z
                    if rule[k][-2][0] != "A":
                        flag = 0
    return rule

def start(rule):
    A = del_invalid_sign(rule)
    A = remove_Epsilon(A)
    A = cfg_to_cnf(A)
    B = cnf_to_gnf(copy.deepcopy(A))
    C = organizing_gnf(copy.deepcopy(B))
    return A,B,C
```