A Formal Language Theory Approach To Music Generation

by

Walter Schulze

Thesis presented in partial fulfilment of the requirements for the degree of Master of Science in Computer Science at Stellenbosch University

Department of Mathematics, Applied Mathematics and Computer Science,
University of Stellenbosch,
Private Bag X1, Matieland 7602, South Africa.

Supervisor: Prof. AB van der Merwe

December 2009

Declaration

I, the undersigned, hereby declare that the work contained in this thesis is my own original work and that I have not previously in its entirety or in part submitted it at any university for a degree.

Signature: . . . . . .

Date: . . . . . .

Abstract

We investigate the suitability of applying some of the probabilistic and automata theoretic ideas, that have been extremely successful in the areas of speech and natural language processing, to the area of musical style imitation. By using music written in a certain style as training data, parameters are calculated for (visible and hidden) Markov models (of mixed, higher or first order), in order to capture the musical style of the training data in terms of mathematical models. These models are then used to imitate two instrument music in the trained style.

Uittreksel

Hierdie tesis ondersoek die toepasbaarheid van probabilitiese en outomaatteoretiese konsepte, wat uiters suksesvol toegepas word in die gebied van spraak en natuurlike taal-verwerking, op die gebied van musiekstyl nabootsing. Deur gebruik te maak van musiek wat geskryf is in 'n gegewe styl as aanleer data, word parameters vir (sigbare en onsigbare) Markov modelle (van gemengde, hoër- of eerste- orde) bereken, ten einde die musiekstyl van die data waarvan geleer is, in terme van wiskundige modelle te beskryf. Hierdie modelle word gebruik om musiek vir twee instrumente te genereer, wat die musiek waaruit geleer is, naboots.

Acknowledgements

Acknowledgements

Dedications

Dedicated to ...

Contents

Chapter 1

Introduction

Chapter 2

Music Theory

2.1 Introduction

This chapter discusses music theory starting with the notion of a musical note. A single musical note is represented by four properties (Ottman, 1983):

- Pitch, how high or low the sound is;
- Duration or note value, how long the sound is held;
- Intensity and loudness of the note;
- Timbre, or the instrument the note is being played on.

Notes are grouped to form chords, which are in turn placed in an ordered sequence to form a chord progression. The ordered sequence of durations of notes in a melody or chords in a chord progression, is the rhythm of the composition. This chapter includes the discussion of chord progression, rhythm, musical notation, intervals, scales, chords, chord inversions, cadences, note duration, time signatures and tempo.

2.2 Pitch

When a string vibrates 261.63 times per second, it has a frequency of 261.63 Hz (Hertz) and a pitch of middle C. This frequency naming convention was

only endorsed by the *International Organization for Standardization* in 1955 (Randel, 2003), and before 1955 there were several popular tuning standards. The standard piano has 88 keys and their frequencies are all related to middle C by the formula $\mathsf{freq(C)} \times 2^{\frac{s}{12}}$, where $\mathsf{freq(C)}$ is the frequency of middle C and s is an integer in the interval $[-39, 48]$. The absolute value of s can also be regarded as the number of semitones, number of half steps or interval size from the specific note to middle C. Each of these pitches has one of the following names: C, C♯, D, D♯, E, F, F♯, G, G♯, A, A♯, B. Every 12 semitones these pitch names repeat and the frequency doubles. This interval size of 12 is referred to as an octave. Octave registers are used to distinguish between pitch names at different frequencies. The octave registers for C are denoted by CC, C, c, c^1, c^2, c^3, c^4, c^5, where c^1 is used for middle C. The lowest note on the standard piano is AAA and has a frequency of 27.5 Hz. This is where humans start to find it hard to distinguish between different pitches (Levitin, 2006). Accidentals modify the pitch of a given note. Sharps (♯) are used to raise the pitch of a given note by one semitone or a half step, and a flat (♭), is used to lower the pitch by one semitone or a half step. This means that C♯ and D♭ are the same for all purposes, except in formal music theory.

2.3 Duration

Rhythm is the variation of duration of a sequence of notes or a series of note values. The duration of a note or note value indicates how long the note is sounded. This is indicated using fractions, for example: whole (𝅝), half (𝅗𝅥), quarter (♩). A dot after a note increases the duration of a note by 50 percent. Thus the note 𝅗𝅥. has a duration of three eighths, as shown in Figure 2.1. When no pitch is sounded for a duration of time, this is called a rest, and it is indicated by using a rest sign which corresponds to the duration, as shown in Figure 2.2.

A measure divides a musical composition into equal time units specified in terms of number of note values. The number of note values is specified by the time signature, which consists of an upper and a lower number. The number of beats per measure is represented by the upper number, while

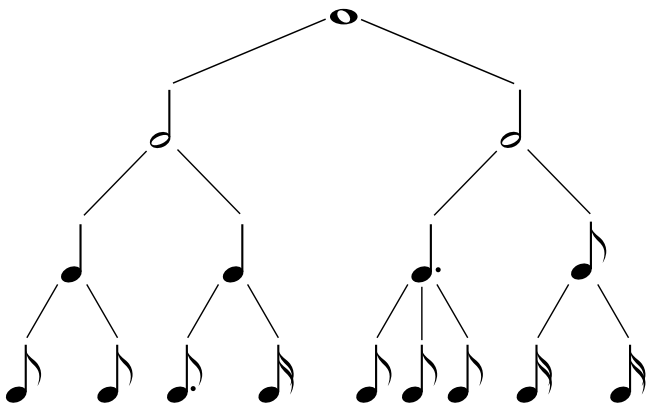


Figure 2.1: Note duration tree

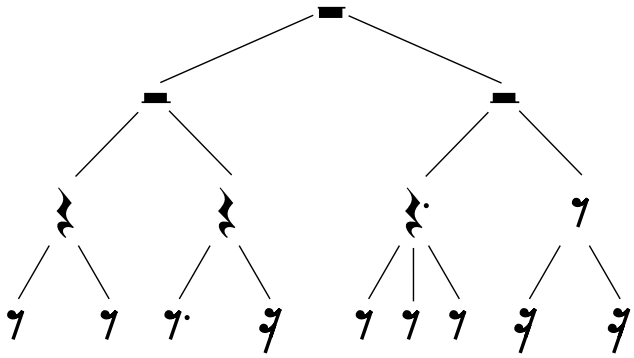


Figure 2.2: Rest duration tree

the duration of a beat is represented by the lower number. For example, if the upper number is two, there are two beats per measure, and if the lower number is 8, the duration of a beat is an eighth. This implies that there will be two eighths per measure. Popular time signatures include:

$\frac{4}{4}$ used in most forms of western classical and pop music;

$\frac{2}{2}$ used for marches and also in fast orchestral music;

$\frac{2}{4}$ used by polkas and sometimes marches;

$\frac{3}{4}$ used for waltzes, scherzi, minuets and some ballads.

The division provided by measures can be bypassed, by letting a note sound or ring from one measure to another.

Tempo of music is described in beats per minute (bpm) and affects the number of seconds a note is played. Suppose that the tempo is 120 bpm and the time signature is $\frac{2}{4}$. Then a beat is a quarter, there are 120 quarters played each minute, and a quarter is played for half a second. This also means that since there are two beats per measure, that the duration of a measure is equal to a second. Furthermore, if the tempo is changed to 60 bpm, then an eighth would be played for half a second. Musicians might play gradually slower towards the end of a composition. This is called a ritardando, and can be used to indicate that the composition is ending.

2.4 Timbre

The timbre of a note is composed of three properties:

- overtone profile;
- attack;
- flux.

When the pitch c^1 is played on a standard piano, one of the strings inside the piano vibrates at several frequencies. The smallest frequency is the defining or fundamental frequency of c^1, namely 261.63 Hz. The other frequencies, known as overtones, are unique for each instrument. Often the frequencies are multiples of the fundamental frequency; for example 523.55 Hz, 784.89 Hz, 1046.52 Hz, etc. Each overtone in the series has its own loudness value relative to the loudness of the other frequencies. These frequency relations are "programmed" in our brains so that *restoration of the missing fundamental* happens when we hear for example a frequency series 220 Hz, 330 Hz, 440 Hz, 550 Hz, which misses the fundamental frequency 110 Hz (Levitin, 2006). Thus although not present, our brain will add the missing frequency.

The attack is the initial frequencies when a note is played. On some music instruments these frequencies have more complex relations than the overtone series. Flux is the way the sound changes once a note starts playing.

Figure 2.3: Clef Signs

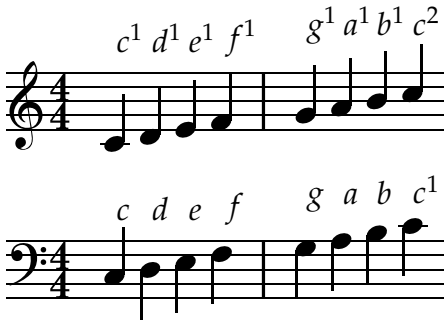


Figure 2.4: Staffs

2.5 Music Notation

Next we discuss music notation. Each pitch is represented on the music staff, which consists of five parallel horizontal lines. Each line represents a specific pitch as shown in Figure 2.4. Clef signs assign a specific pitch to a given line. Figure 2.3 shows the most popular clef signs, the G and F clef. The G clef assigns the G above middle C to the line encircled by its curl, while the F clef assigns the F below middle C to the line between its two dots. The time signature shown right of the clef in Figure 2.4 indicates that there are four quarters in a measure.

2.6 Melody and Harmony

When listening to music, one hears a multitude of sounds at the same time, and also one after the other. Melody can be described as notes heard in succession while harmony as notes heard at the same time. We can also describe melody and harmony as respectively the horizontal and vertical movement of the music.

Scales are collections or subsets of pitches used to compose music. In other

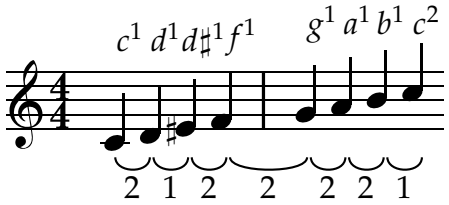


Figure 2.5: C melodic minor (ascending) scale with indicated semitone intervals

words, for a composition, a scale is chosen and then mostly those pitches are used for the composition. In a more complex composition the scale can for example be changed midway through a composition. Even with these exceptions, scales are still important when composing.

Example 2.6.1 (Basic scale examples) **-** The notes in the scales C major, D major, C natural minor, C melodic minor and C pentatonic.

- C major : {C,D,E,F,G,A,B}
- D major : {D,E,F♯,G,A,B,C♯}
- C natural minor : {C,D,D♯,F,G,G♯,A♯}
- C melodic minor (ascending) : {C,D,D♯,F,G,A,B}
- C melodic minor (descending) : {C,D,D♯,F,G,G♯,A♯}
- C pentatonic : {C,D,F,G,A}

In Example 2.6.1 the notes in sample scales are given. Scales are defined by their starting pitch, called the root, and type. C major has a root key of C and is of type major. The type of a scale defines the sequence of intervals. A whole step is equal to two half steps or two semitones and has an interval size of two. The major scale has the following sequence of steps: {whole, whole, half, whole, whole, whole, half}. The melodic minor (ascending) scale has the following sequence of steps: {whole, half, whole, whole, whole, whole, half} and is shown in Figure 2.5. Note that the melodic minor has two forms (ascending and descending) depending on whether the melody is approaching the root note from below or above respectively. There are many other types of scales, for example the harmonic minor and whole tone scales (Randel, 2003).

A melody is a sequence of notes played one after the other. It is not the ab-

solute pitches that identify a melody, but the intervals between them. The melodic contour or pitch profile takes into account only the positive or negative movement of the melody at every interval, in other words whether the next pitch is higher of lower than the previous one. The melodic contour is encoded by Parsons code as follows:

- "u" = up;
- "d" = down;
- "r" = repeat, when the next pitch is equal to the previous one;
- "*" = first pitch.

For example, Twinkle Twinkle Little Star represented by parsons code is: *rururddrdrdrd (Parsons, 1975). In McNab *et al.* (2000) the melodic contour is used to search a database of compositions. They found that using interval sizes, and not just the contour, provided better results for the purpose of music identification, since fewer intervals are required to identify a composition. The interval distance between C and F♯, called a tritone, or in the middle ages the devil's interval, was banned by the Roman Catholic church (Levitin, 2006). When listening to a melody the volume does not impair your ability to recognise the composition. Recognising the melody will also not be affected by changing the root key.

A chord is a group of notes played at the same time. The simplest chord is called a triad, which consists of three notes (see Figure 2.6 [1]). Whereas melody is defined by horizontal intervals, the chord type is defined by vertical intervals. The first (or lowest) note is called the root, followed by the third and fifth interval. The third and fifth intervals are respectively the third and fifth note in the major or minor scale relative to the first (root) note. The fifth interval is the same for the major and minor triad and is called a perfect fifth interval. The third interval for the major and minor triad is called a major third and minor third interval respectively. The minor third interval is three semitones, one less than the major third's four semitones. Another chord, the diminished triad is defined by a minor third and a diminished fifth interval. The diminished fifth interval is six semitones, one

[1] Note that we have dropped the case sensitivity of the minor roman numerals in the rest of the thesis for ease of explanation.

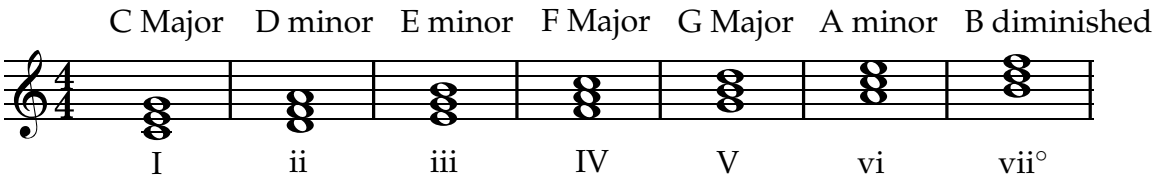


Figure 2.6: Some example triads

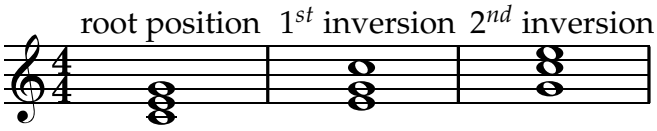


Figure 2.7: C major and its inversions

less than the perfect fifth's seven semitones. Lastly, the augmented triad is defined by the major third and augmented fifth (8 semitones) intervals. There are many other types of chords, for example the dominant seventh, major seventh, dominant eleventh, etc (Ottman, 1983). Contradictory to our previous statement, that the notes of chord are played at the same time, an arpeggio, also called a broken chord, is where the notes of chord are played consecutively. This simplifies the definition of a chord to just a group of notes.

In Figure 2.7, various versions of the C major triad is shown. These triads are referred to as inversions of the C major triad, since the notes representing the C major triad (C,E,G) can be played in any order and still form the C major triad. This implies that a chord type has several sequences of vertical intervals to choose from. A triad without its third is called a power chord. Playing two power chords directly one after the other, is called a parallel fifth, and does not conform to classical music theory rules (Randel, 2003). A parallel motion is when two notes move by the same vertical interval. The parallel fifth is when the vertical interval between the two notes is a fifth. Any combination of vertical intervals can form a chord, but whether a given chord fits in a composition is up to the style of music and the chords between which the given chord is played.

2.7 Chord Progressions and Cadence

A chord progression is an ordered sequence of chords. Chord progressions are usually repeated a few times in a composition. For example, a verse of music in the style of punk might have the chord progression: I $\rightarrow$ V $\rightarrow$ VI $\rightarrow$ IV. This will usually be repeated four times. This repetition is representative of a motif, a repeating and developing melodic or rhythmic idea, or the basic component of the composition. Roman numerals are used to indicate the chord number in the scale of the composition. In Figure 2.6 the triads of the C major scale with their respective numerals are shown.

A cadence or a falling, as it is called in western music, is a certain sequence of intervals or chords that end a phrase (verse, chorus, etc.). When a phrase ends with the chord progression V $\rightarrow$ I or VII $\rightarrow$ I, it is referred to as an authentic or perfect cadence. The chord progression VII $\rightarrow$ I is also representative of resolving dissonance, or harmonic tension. Other types of cadences include (Adams, 2000):

- half cadence: I $\rightarrow$ V
- plagal cadence (Amen cadence): IV $\rightarrow$ I
- deceptive cadence: V $\rightarrow$ VI

Cadences give a definite ending, indicating to the listener that the piece of music is concluding.

2.8 Conclusion

This chapter gave an introduction to music theory, covering the terminology required for this thesis. For our music generation system we consider compositions to be chords for the rhythm guitar or piano and a melody for the lead.

Chapter 3

Automata and Markov Models

3.1 Introduction

Probabilistic finite automata and some of its subclasses are discussed in this chapter. We assume a pre-existing knowledge of basic probability theory and in particular Bayes' theorem.

3.2 Markov Chains

A Markov chain models a sequence of events by using states and transition probabilities between states. This model adheres to the first order Markov assumption which states:

$$P(q_t|q_{t-1}, q_{t-2}, \ldots, q_1) = P(q_t|q_{t-1}),$$

where $q_1, \ldots, q_t$ is a set of states. Thus in a Markov chain the probability of being in state q_t, at time t, depends only on the previous state, at time t-1. A homogeneous Markov chain, which is defined in Definition 3.2.1, adheres to the stationarity assumption, which states that the transition probabilities of the Markov chain are time-independent. This implies that the probability of moving to a next state, at any time, only depends on the current state.

Definition 3.2.1 (Markov chains) **-** A Markov chain (MC) is a 3-tuple $\langle$ Q,a,π $\rangle$ where:

- Q is the state space,
- a : Q $\times$ Q $\rightarrow$ $[0,1]$ is a mapping defining the probability of each transition,
- and π : Q $\rightarrow$ $[0,1]$ is a mapping defining the initial probability of each state.

The following constraints must be satisfied:

- for $q_i \in Q$, $\sum_{q_j \in Q} a(q_i, q_j) = 1$,
- $\sum_{q \in Q} \pi(q) = 1$.

One approach to estimating the transition probabilities of Markov chains is by calculating the maximum likelihood estimate, using frequency or empirical counts as in the next example.

Example 3.2.1 (Calculating the parameters of a Markov chain) **-** Assume that the training sequences are two melodies (c, d, e, c, d, c, d, e, c, d) and (d, e, d, e, c, d, c, d, e, c). This implies that the state space Q is $\{c, d, e\}$. The transition probabilities are calculated as follows:

$$a(q_i, q_j) = \frac{\#(q_i \rightarrow q_j)}{\sum_{q_k \in Q} \#(q_i \rightarrow q_k)} \quad ,$$

where $\#(q_i \rightarrow q_j)$ is the number of times state q_i is followed by state q_j. Thus:

$$a = \begin{pmatrix} a(c,c) & a(c,d) & a(c,e) \\ a(d,c) & a(d,d) & a(d,e) \\ a(e,c) & a(e,d) & a(e,e) \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ \frac{2}{7} & 0 & \frac{5}{7} \\ \frac{4}{5} & \frac{1}{5} & 0 \end{pmatrix}$$

and

$$\pi = \begin{pmatrix} \pi(c), & \pi(d), & \pi(e) \end{pmatrix} = \begin{pmatrix} \frac{1}{2}, & \frac{1}{2}, & 0 \end{pmatrix}$$

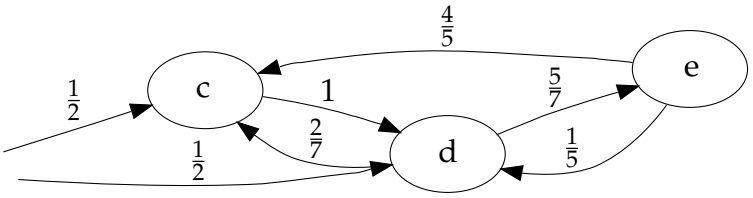


Figure 3.1: A first order Markov chain

We can visualise the Markov chain as a graph where vertices represent states and edges represent transitions as in Figure 3.1.

3.3 Higher Order Markov Chains

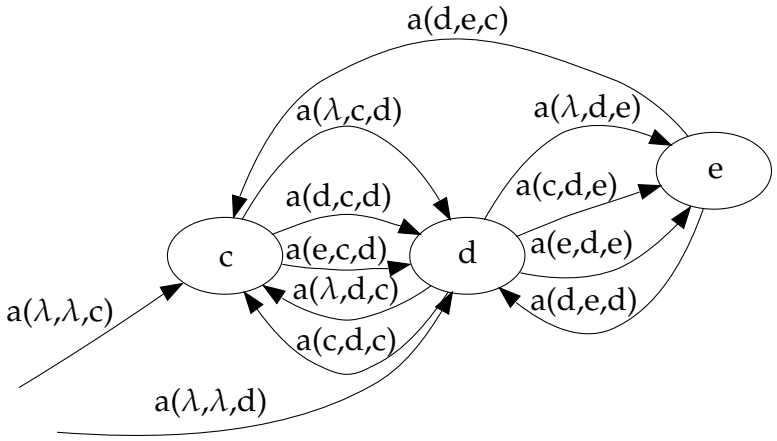
In contrast to Markov chains as defined in Definition 3.2.1, higher order Markov chains have a memory length larger than one. More precisely, an L^{th} order Markov chain operates under the assumption:

$$P(q_t|q_{t-1}, q_{t-2}, \ldots, q_1) = P(q_t|q_{t-1}, \ldots, q_{t-min(t-1,L)}).$$

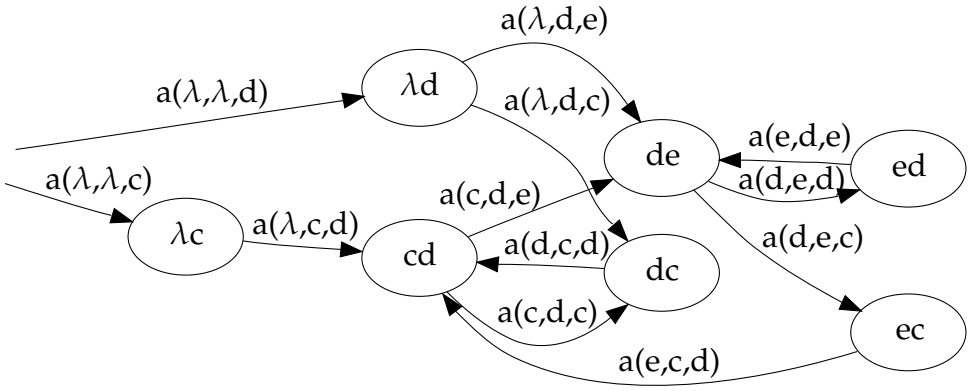
When $t \leq L$, q_t is a startup state, and the excessive states, q_t with (t < 0), are represented by the empty string λ. In the case of L^{th} order Markov chains we use $a(q_{t-L}, \ldots, q_{t-1}, q_t)$, instead of $a(q_{t-1}, q_t)$, to indicate transition probabilities. This results in multiple transition probabilities between states q_{t-1} and q_t, as in Figure 3.2.

The complication introduced by higher order Markov chains can be removed by increasing the number of states and adding a history to the states, as in Figure 3.2. In order to transform an L^{th} order Markov chain to a first order Markov chain, we replace the state space Q by $\cup_{i=1}^{L} Q^i$, where Q^i is all state sequences of length i. More precisely, if q'_t and q'_{t-1} are states in a first order Markov chain, corresponding to a given higher order Markov chain, at times t and t-1 respectively, with $q'_t = (q_{t-L+1}, \ldots, q_t)$ and $q'_{t-1} = (q_{t-L}, \ldots, q_{t-1})$, then we have the following:

$$\begin{aligned} a(q'_{t-1}, q'_t) &:= P(q'_t|q'_{t-1}) \\ &= P(q_{t-L+1}, \ldots, q_t|q_{t-L}, \ldots, q_{t-1}) \\ &= P(q_t|q_{t-L}, \ldots, q_{t-1}) \\ &= a(q_{t-L}, \ldots, q_{t-1}, q_t). \end{aligned}$$



A second order Markov chain



A first order Markov chain

Figure 3.2: A second order Markov chain and its equivalent first order Markov chain

We are thus able to convert a higher order Markov chain to an equivalent first order Markov chain. Next we discuss prediction suffix automata which are equivalent to mixed (variable) order Markov chains.

3.4 Prediction Suffix Automata

Conversion from higher to first order Markov chains, as discussed in the previous section, involves moving the memory from the transitions to the states. Mixed order Markov chains allow transitions of various memory lengths, whereas all transitions of higher order Markov chains have the

same memory length. Prediction Suffix Automata (PSA) represent memory using state labels in a similar fashion to first order Markov chains obtained from translating higher order Markov chains to their equivalent first order Markov chains. Both mixed and higher order Markov chains represent their memory using transitions, whereas prediction suffix automata (PSA) uses state labels. This flexibility of memory lengths in a PSA allows us to avoid the exponential growth associated with higher order Markov chains.

Next we define prediction suffix automata, which are equivalent to mixed order Markov chains.

Definition 3.4.1 (Prediction suffix automata (Ron *et al.*, 1996; Schwardt, 2007)) **-** A prediction suffix automata (PSA) is a 4-tuple $\langle\ \Sigma, \mathrm{Q}, \tau, \mathrm{p}\ \rangle$ where:

- Σ is a finite input alphabet,
- $\mathrm{Q} \subset \Sigma^*$ is a finite set of finite-length strings (so that $\lambda \in \mathrm{Q}$) which is the state space,
- $\tau : \mathrm{Q} \times \Sigma \rightarrow \mathrm{Q}$, is the state transition function,
- $\mathrm{p} : \mathrm{Q} \times \Sigma \rightarrow [0, 1]$, is the next symbol probability distribution.

The following constraints must be satisfied:

- $\sum_{\sigma \in \Sigma} p(q, \sigma) = 1$ for all $q \in Q$;
- the start state q_0 is the empty string λ;
- for all $q \in Q$ and $\sigma \in \Sigma$, $\tau(q, \sigma)$ is equal to the longest suffix of $q\sigma$ that is in Q.

If in a PSA $\langle\ \Sigma, \mathrm{Q}, \tau, \mathrm{p}\ \rangle$ the state space of the PSA contains all states of length L, i.e. if $\Sigma^L \subseteq Q$, we refer to the PSA as an L-PSA. Note that an L-PSA is equivalent to an L^{th} order Markov chain. Both models limit the maximum memory length to L.

Training an L-PSA from training sets is achieved by first training an L-PST (see next section for the definition of a PST) and then converting the L-PST to an equivalent L-PSA. A PSA's transition function specifies the next state, given the previous state and alphabet symbol, whereas a PST needs to calculate the next state. This is done by appending the alphabet symbol to the

right of the current state string and then searching for the longest suffix that is also a state in the PST. This calculation in a PST can take L times longer than when a PSA's transition function is used. The PSA also has extra added prefix states, which connects previously unreachable states to the structure.

In the next section we discuss PSTs, including training and converting a PST to a PSA.

3.5 Prediction Suffix Trees

Definition 3.5.1 (prediction suffix trees (Ron *et al.*, 1996; Schwardt, 2007)) **-** A prediction suffix tree (PST) is a 3-tuple $\langle\ \Sigma, \mathrm{Q}, \mathrm{p}\ \rangle$ where:

- Σ is a finite alphabet;
- Q is the finite state space and $Q \subset \Sigma^*$ is a finite set of finite-length strings with $\lambda \in Q$;
- $\mathrm{p} : \mathrm{Q} \times \Sigma \rightarrow [0, 1]$, is the next symbol probability distribution.

The following constraints must be satisfied:

- $\sum_{\sigma \in \Sigma} p(q, \sigma) = 1$ for all $q \in Q$;
- for all $q \in (Q - \lambda)$ there exists s $\in \Sigma$ and $q' \in Q$ such that q = sq', and q' is the parent of q;
- the root of the tree is labelled by λ.

Ziv and Lempel (1978) developed a variable order algorithm for lossless data compression. LearnPSA is an equivalent lossy compression algorithm. The LearnPSA algorithm (Ron *et al.*, 1996) is used to calculate the state space and probability distributions of a PST. This algorithm finds all the strings with a statistical significance, given certain input parameters for the training sequences. The algorithm's design was motivated by the Probably Approximately Correct (PAC) learning model (Valiant, 1984). Starting at the root node, which is represented by the empty string, the algorithm follows a top-down approach to build a tree to which nodes are added, which appear a significant number of times in the training data and which have a

unique next symbol distribution when compared to shorter suffixes of the same node. The parameters for LearnPSA are listed below:

- Σ is the input alphabet;
- L is the maximum string length allowed to label a state;
- n is the maximum number of states allowed;
- $\delta \in (0, 1)$ is the approximation parameter;
- and $\omega_1, \ldots, \omega_T$ are the training sequences.

The threshold values are calculated as follows:

- $\gamma = \frac{\delta}{48L|\Sigma|}$, is the smoothing factor;
- $P_{min} = \frac{\delta}{2nLlog(1/\gamma)} - \frac{|\Sigma|}{8n}$, is the minimum empirical string probability;
- $\alpha = (1 + \gamma|\Sigma|) \times \gamma$, is the minimum empirical next symbol probability;
- $\beta = 1 + 3\gamma|\Sigma|$, is the minimum empirical next symbol probability ratio.

Next we define the following functions:

- $\eta_i(q, s)$ = the number of times the string $q.s$ appears in the training sequence, ω_i ;
- $\eta_i(q, *) = \sum_{s \in \Sigma} \eta_i(q, s)$;
- $p(q) = \frac{\sum_{i=1}^{T} \eta_i(q,*)}{(\sum_{i=1}^{T} |\omega_i|) - T}$;
- $p(q, s) = \frac{\sum_{i=1}^{T} \eta_i(q,s)}{\sum_{i=1}^{T} \eta_i(q,*)}$,

where $p(q, s)$ represents the frequency probabilities. These functions are calculated for every $q \in \Sigma^{\leq L} = \cup_{i=0}^{L} \Sigma^i$ and $s \in \Sigma$. We need to subtract T from the sum of the length of all training sequences in the denominator of $\frac{\sum_{i=1}^{T} \eta_i(q,*)}{(\sum_{i=1}^{T} |\omega_i|) - T}$, since the last symbol in each training sequence does not have a next symbol. Also, we define parent(q) to be the longest proper suffix of q.

The pseudocode for LearnPSA is given on the next page:

LearnPSA(Σ,L,n,δ,ω)

---

01 $Q = \{\lambda\}$

Initialise the state space to include the root state λ.

02 $F = \{s \mid s \in \Sigma \; and \; P(s) \geq P_{min}\}$

Initialise the frontier set of states to be considered to include every alphabet symbol with a high enough occurrence rate in the training sequence.

03 *while* $F \neq \varnothing$
04 $\quad q \; = \; F.pop()$

While there are states to be considered, remove the currently considered state from the frontier.

05 $\quad for \; all \; s \in \Sigma$
06 $\quad\quad if \; p(q,s) \geq \alpha$ and $\frac{p(q,s)}{p(parent(q),s)} \geq \beta$
07 $\quad\quad\quad Q = Q \cup q$

If the empirical next symbol probability is significant enough, according to α, and if the state q provides significantly more statistical information about the next symbol than its parent does, q is added to the state space.

08 $\quad\quad\quad F = F \cup (\text{Suffixes}(q) - Q)$
09 $\quad\quad end \; if$

All suffixes of q which are not in the state space or the frontier are now also considered as potential states.

10 $\quad end \; for$
11 $\quad if \; |q| < L \; : \; F = F \cup \{s \cdot q \mid s \in \Sigma \; and \; P(s \cdot q) \geq P_{min}\}$

Consider all children of q with a high enough frequency count.

12 *end while*
13 $p(q,s) = p(q,s) \times (1 - |\Sigma| \times \gamma) + \gamma$

Set the next symbol probability for every state and symbol respectively, while taking a smoothing factor into account.

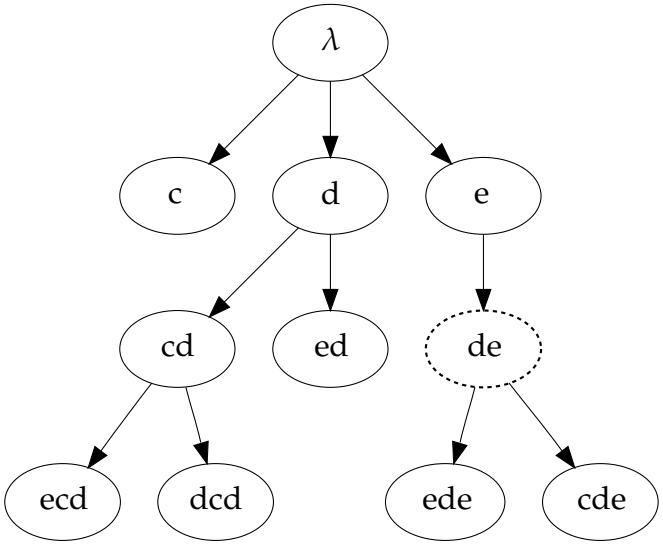


Figure 3.3: A prediction suffix tree

Completing the suffix tree is done by adding all missing parents of the state space. These parents inherit their next symbol probabilities from their respective parents.

Example 3.5.1 - In Figure 3.3 the resulting PST is shown when the input parameters are:

- $\Sigma = \{c, d, e\}$
- $L = 3$
- $n = 10$
- $\delta = 0.1$
- ω_1 = (c, d, e, c, d, c, d, e, c, d)
- ω_2 = (d, e, d, e, c, d, c, d, e, c)

The suffix ec was not added to the tree, since it has the same probability distribution as its parent c. This is the case since c is always followed by d independent of whether its predecessor is e or d. The node de is only added after the completion of the LearnPSA algorithm, in order to complete the constructed tree. Note that internal nodes could have the same next symbol distribution as their parent, in the case where they are used to complete the tree, but this is not true for leaf nodes.

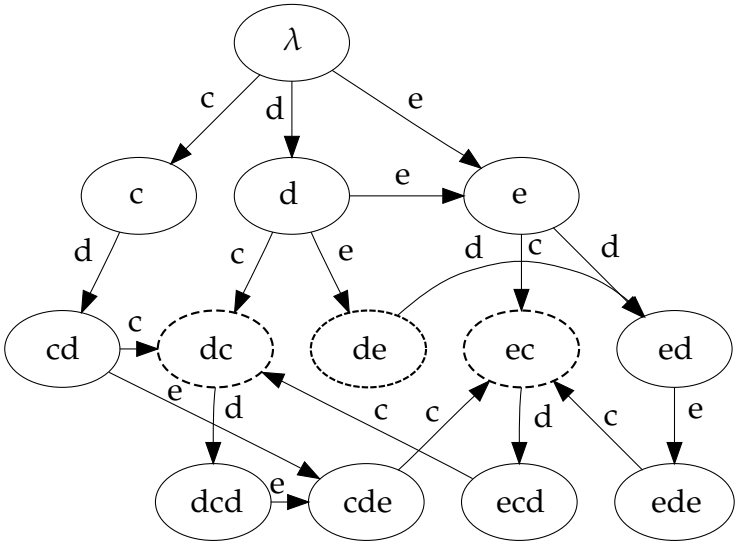


Figure 3.4: A prediction suffix automata corresponding to the PST in Figure 3.3

Converting a PST to a PSA is achieved by adding all missing prefixes to the state space and by constructing the transition function τ. Each state needs all its prefixes to allow the state to be reachable in the PSA. These prefixes inherit their transition probabilities from their longest proper suffix in the state space. Constructing τ involves finding the destination state for each source state and transition symbol. The destination state of a transition is the longest suffix of the string obtained by concatenating the source state and the transition symbol, which is also in the state space. The resulting PSA converted from the PST in Figure 3.3 can be seen in Figure 3.4. Nodes dc and ec are the prefixes added to complete the automaton.

PSAs are in general more compact than higher order Markov chains, since higher order Markov chains require all states of maximum length, where in contrast PSAs only require statistically significant states.

3.6 Hidden Markov Models

Hidden Markov models (HMMs) (Rabiner, 1990) are used to model the relationship between a hidden and an observed sequence. A discrete hidden Markov model is a Markov chain with a discrete probability distribution at each state. These discrete probability distributions define probabilities of

emitting a specific alphabet symbol in a given hidden state. Thus the states of the Markov chain are the hidden states.

Definition 3.6.1 (Discrete hidden Markov models (Dupont *et al.*, 2005)) **-** A discrete HMM is a 5-tuple $\langle \Sigma, Q, a, b, \pi \rangle$ where:

- Σ is a finite alphabet of visible symbols;
- Q is a finite set of hidden states;
- $a : Q \times Q \rightarrow [0, 1]$ is a mapping defining the probability of transitions between hidden states;
- $b : Q \times \Sigma \rightarrow [0, 1]$ is a mapping defining the emission probability of each visible symbol at a given hidden state, also called a confusion matrix;
- and $\pi : Q \rightarrow [0, 1]$ is a mapping that defines the initial probability of the hidden states.

The following constraints must be satisfied:

- for all $q_i \in Q$, $\sum_{q_j \in Q} a(q_i, q_j) = 1$;
- for all $q \in Q$, $\sum_{\varsigma \in \Sigma} b(q, \varsigma) = 1$;
- $\sum_{q \in Q} \pi(q) = 1$.

We denote the observation and hidden state sequence by $\chi = \chi_1, \ldots, \chi_n$ and $s = s_1, \ldots, s_n$, respectively, where $\chi_i \in \Sigma$ and $s_i \in Q$. We use the following notation:

$$\begin{aligned} \pi(q_i) &= P(s_1 = q_i); \\ a(q_i, q_j) &= P(s_t = q_j | s_{t-1} = q_i); \\ b(q_j, \chi_t) &= P(\chi_t | s_t = q_j). \end{aligned}$$

In the notation above, $P(x)$ is the probability of an event, while $P(x \mid y)$ is the probability of an event x given that event y has occurred.

In Example 3.6.1 the observation sequence represents the chord progression and the hidden sequence represents the melody of a given composition. This models the relation between two instruments, one playing chords and

the other a chordless melody. In the case where the hidden state sequence and observation sequence are available as training data, empirical counts can be used to calculate the parameters of the model, as for example when using music. When this is not the case, the Baum-Welch forward-backward algorithm is used.

Example 3.6.1 (Training an HMM using empirical counts) **-** The training input is as follows:

$$\begin{array}{c|cccccccccc} \chi_1 & \text{I} & \text{I} & \text{I} & \text{II} & \text{II} & \text{II} & \text{I} & \text{I} & \text{II} & \text{II} \\ s_1 & \text{c} & \text{d} & \text{e} & \text{c} & \text{d} & \text{c} & \text{d} & \text{e} & \text{c} & \text{d} \\ \chi_2 & \text{I} & \text{I} & \text{I} & \text{I} & \text{II} & \text{II} & \text{II} & \text{II} & \text{I} & \text{I} \\ s_2 & \text{d} & \text{e} & \text{d} & \text{e} & \text{c} & \text{d} & \text{c} & \text{d} & \text{e} & \text{c} \end{array},$$

where χ is the observation sequence and s is the hidden state sequence. This implies that $\Sigma = \{\text{I, II}\}$ and $\text{Q} = \{\text{c, d, e}\}$. The hidden state sequence can be used as in Example 3.2.1 to determine the transition probabilities of the underlying Markov chain,

$$a = \begin{pmatrix} 0 & 1 & 0 \\ \frac{2}{7} & 0 & \frac{5}{7} \\ \frac{4}{5} & \frac{1}{5} & 0 \end{pmatrix} \text{ and } \pi = \begin{pmatrix} \frac{1}{2}, & \frac{1}{2}, & 0 \end{pmatrix}.$$

The confusion matrix, b, is also trained using counts as follows:

$$\text{for all } q \in Q \text{ and } \sigma \in \Sigma, \text{ we have that } b(q, \sigma) = \frac{\#(s_t = q \; and \; \chi_t = \sigma)}{\#(s_t = q)}.$$

Thus:

$$b = \begin{pmatrix} b(c, I) & b(c, II) \\ b(d, I) & b(d, II) \\ b(e, I) & b(e, II) \end{pmatrix} = \begin{pmatrix} \frac{2}{7} & \frac{5}{7} \\ \frac{1}{2} & \frac{1}{2} \\ 1 & 0 \end{pmatrix}.$$

A graphical representation of the HMM is shown in Figure 3.5, with the discrete probabilistic distribution at each state displayed as a histogram.

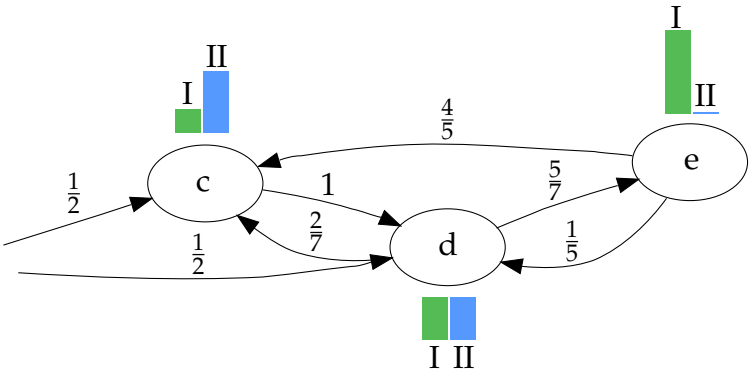


Figure 3.5: A discrete hidden Markov model with histograms defining the emission probabilities

3.7 Hidden Markov Model Algorithms

Next we discuss algorithms applicable to hidden Markov models. The forward algorithm is used to calculate the probability of a given observation sequence. This algorithm is used when multiple models are available, and the model which matches the observation sequence the best needs to be selected. The solution is a dynamic programming algorithm and involves filling in the forward matrix, $\alpha(t, q_i) = P(s_t = q_i, \chi_1^t)$, where χ_1^t represents the observation sequence $\chi_1, \ldots, \chi_t$. Each cell of the forward matrix is equal to the sum of the probabilities of all paths which lead to state q_i and which emits the specified observation sequence from time 1 to time t (see Figure 3.6). The forward matrix is calculated as follows:

$$\alpha(t, q_i) = \begin{cases} \pi(q_i) \times b(q_i, \chi_t) & \text{if } t = 1; \\ \left(\sum_{q_j \in Q} \alpha(t-1, q_j) \times a(q_j, q_i) \right) \times b(q_i, \chi_t) & \text{otherwise.} \end{cases}$$

Thus for a given model, the probability of a specific observation sequence χ_1^T is equal to $\sum_{q \in Q} \alpha(T, q)$. The asymptotic running time of the forward algorithm is $O(|Q|^2 T)$.

Viterbi decoding is used to find the state sequence with the maximum likelihood, given an observation sequence. The dynamic programming solution used for Viterbi decoding is similar to the forward algorithm, but instead of calculating the sum of all paths leading to a hidden state, the most probable path probability leading to a hidden state is determined. The probability

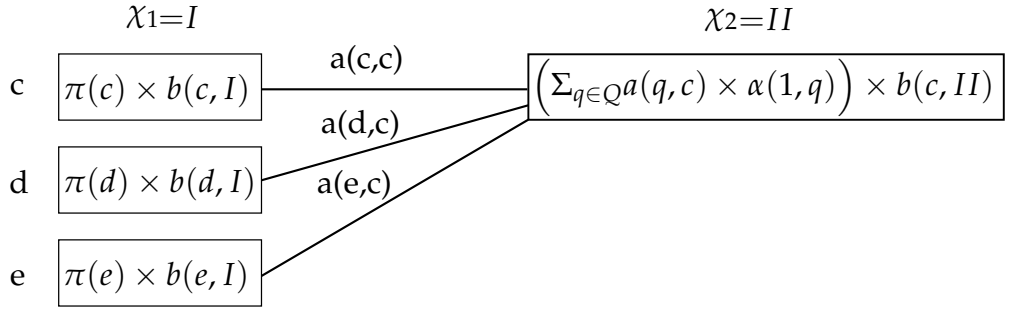


Figure 3.6: The forward algorithm

An application of the forward algorithm with observation sequence χ = (I,II) and state space Q = {a,b,c}.

represented by each cell of the matrix δ is defined as follows:

$$\delta(t, q_i) = P(s_t = q_i, s_{t-1} = s^*_{t-1}, \ldots, s_1 = s^*_1, \chi^t_1),$$

where s^* is the state sequence with the maximum probability, given the observation sequence. These probabilities are calculated recursively as follows:

$$\delta(t, q_i) = \begin{cases} \pi(q_i) \times b(q_i, \chi_t) & \text{if } t = 1; \\ \max_{q_j \in Q}(\delta(t-1, q_j) \times a(q_j, q_i)) \times b(q_i, \chi_t) & \text{otherwise.} \end{cases}$$

While calculating δ, we construct ϕ, which is used to store the state at time (t - 1) in the most probable path to state q_i at time t. We calculate ϕ, for t > 1, as follows:

$$\phi(t) = \text{argmax}_{q_j \in Q}(\delta(t-1, q_j) \times a(q_j, q_i)).$$

After calculating δ and ϕ we use backtracking to find the most probable path ι for a given observation sequence, as shown below:

$$\iota(t) = \begin{cases} \text{argmax}_{q_j \in Q} \delta(t, q_j) & \text{if } t = T; \\ \phi(t+1, \iota(t+1)) & \text{otherwise.} \end{cases}$$

The asymptotic running time of Viterbi decoding is O($|Q|^2$T), which is the same as the running time for the forward algorithm.

Note that Viterbi decoding only finds the most probable hidden state sequence and not a probability distribution over possible hidden state sequences, from which possible sequences of hidden states can be selected by taking the probability distribution into account. The A* search algorithm

(Hart *et al.*, 1968) is an alternative dynamic algorithm which finds the k most probable sequences.

The backward algorithm calculates the probability of emitting a partial observation sequence $\chi_{t+1}^T = \chi_{t+1}, \ldots, \chi_T$, given that the HMM is in the hidden state q_i at time t, and defines β as follows:

$$\beta(t, q_i) = P(\chi_{t+1}^T | s_t = q_i).$$

The backward matrix can be calculated by using a dynamic programming algorithm as shown below:

$$\beta(t, q_i) = \begin{cases} 1 & \text{if } t = T; \\ \sum\limits_{q_j \in Q} (a(q_i, q_j) \times b(q_j, \chi_{t+1}) \times \beta(t+1, q_j)) & \text{otherwise.} \end{cases}$$

Using the transition, confusion, forward and backward matrices we can calculate γ, which is the probability of transitioning from state q_i to q_j at time t, given the full observations sequence $\chi_1^T = \chi_1, \ldots, \chi_T$ (Huang *et al.*, 2001), as shown below for (t > 1):

$$\begin{aligned} \gamma(t, q_i, q_j) &= P(s_{t-1} = q_i, s_t = q_j | \chi_1^T) \\ &= \frac{\alpha(t-1, q_i) \times a(q_i, q_j) \times b(q_j, \chi_t) \times \beta(t, q_j)}{\sum\limits_{q_k \in Q} \alpha(T, q_k)}. \end{aligned}$$

The initial probabilities, at time $t = 1$, is given by:

$$\gamma(t, q_j) = \frac{\pi(q_j) \times b(q_j, \chi_1) \times \beta(t, q_j)}{\sum\limits_{q_k \in Q} \alpha(T, q_k)}.$$

By using γ, we can calculate the probability ω of transitioning to the next hidden state q_j at time t, given the current hidden state q_i at time (t - 1) and the full observation sequence $\chi_1^T = \chi_1, \ldots, \chi_T$, as follows:

$$\begin{aligned} \omega(t, q_i, q_j) &= P(s_t = q_j | s_{t-1} = q_i, \chi_1^T) \\ &= \frac{\gamma(t, q_i, q_j)}{\sum\limits_{q_j \in Q} \gamma(t, q_i, q_j)}. \end{aligned}$$

The Markov chain determined by $\omega(t, *, *)$ can be used to generate multiple possible hidden state sequences, given the observation sequence and the model. This is used in our music generation system to compose a melody given the chords.

3.8 Mixed Order Hidden Markov Models

First order HMMs consist of a first order Markov chain with a probability distribution associated with each state of the Markov chain. More generally, a mixed order hidden Markov model is a mixed order Markov chain with a probability distribution associated with each state of the Markov chain. Instead of using a mixed order Markov chain, as the base of the mixed order hidden Markov model, we use a Prediction Suffix Automata (PSA), since PSAs are equivalent to mixed order Markov chains. Each state of the PSA used in a given mixed order hidden Markov model, is associated with a probability distribution defining its emission probabilities. This probability distribution is the same for all states in the PSA which have the same last symbol in their respective state labels. Note that the mixed order nature of a PSA is kept in the state labels, instead of the transitions as is the case for mixed order Markov chains. This property makes the PSA's transitions first order and allows the algorithms discussed in Section 3.7, to be applied to a our mixed order hidden Markov model. Figure 3.7 shows the PSA in Figure 3.4 with associated probability distributions. The distributions f_c, f_d and f_e are represented by the rows of the confusion matrix.

3.9 Probabilistic Finite Automata

Finally, we discuss Probabilistic Finite Automata (PFA), since all other automata discussed in the previous part of this chapter are PFAs. The class of deterministic probabilistic finite automata (DPFA) is a subclass of the class of PFA, with the property that the transition from a state, given an input symbol, is unique (see Definition 3.9.1).

Note that all prediction suffix automata (PSA) are deterministic. The set of

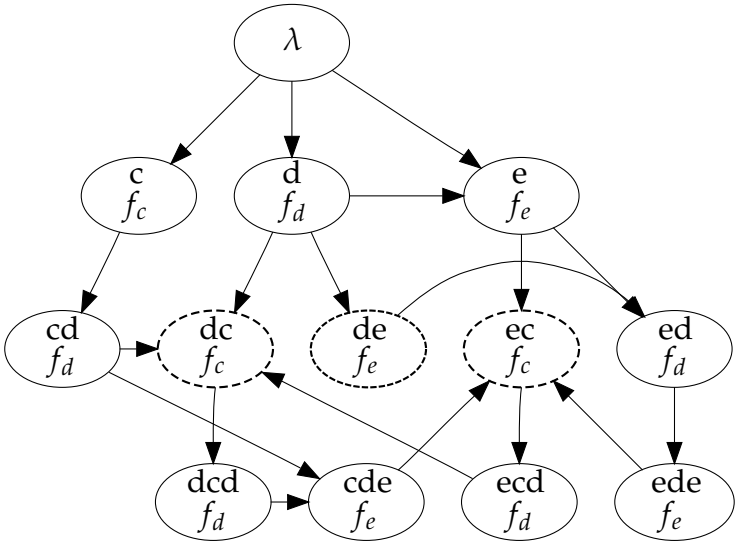


Figure 3.7: A mixed order Markov model

states of a PSA is contained in Σ^*, which is not necessarily the case for (deterministic or non-deterministic) PFAs. This implies that the probability distributions determined by PSAs, prediction suffix trees (PSTs) and Markov chains, are a subset of the probability distributions determined by DPFAs, which is in turn is a subset of the probability distributions determined by PFAs and hidden Markov models (HMMs) (Schwardt, 2007).

Next we give the formal definition of a PFA.

Definition 3.9.1 (Probabilistic finite automata (Thollard *et al.*, 2005)) **-** A probabilistic finite automaton (PFA) is a 5-tuple $\langle$ Q,Σ,τ,π,p $\rangle$ where:

- Σ is a finite input alphabet;
- Q is the state space;
- $\tau \subseteq Q \times \Sigma \times Q$ is a set of transitions;
- π : Q $\rightarrow [0,1]$ defines initial-state probabilities;
- p : $\tau \rightarrow [0,1]$ defines transition probabilities.

The following constraints must be satisfied:

- $\sum_{q \in Q} \pi(q) = 1$;
- for $q_i \in Q$, $\sum_{s \in \Sigma, q_j \in Q} p(q_i, s, q_j) = 1$.

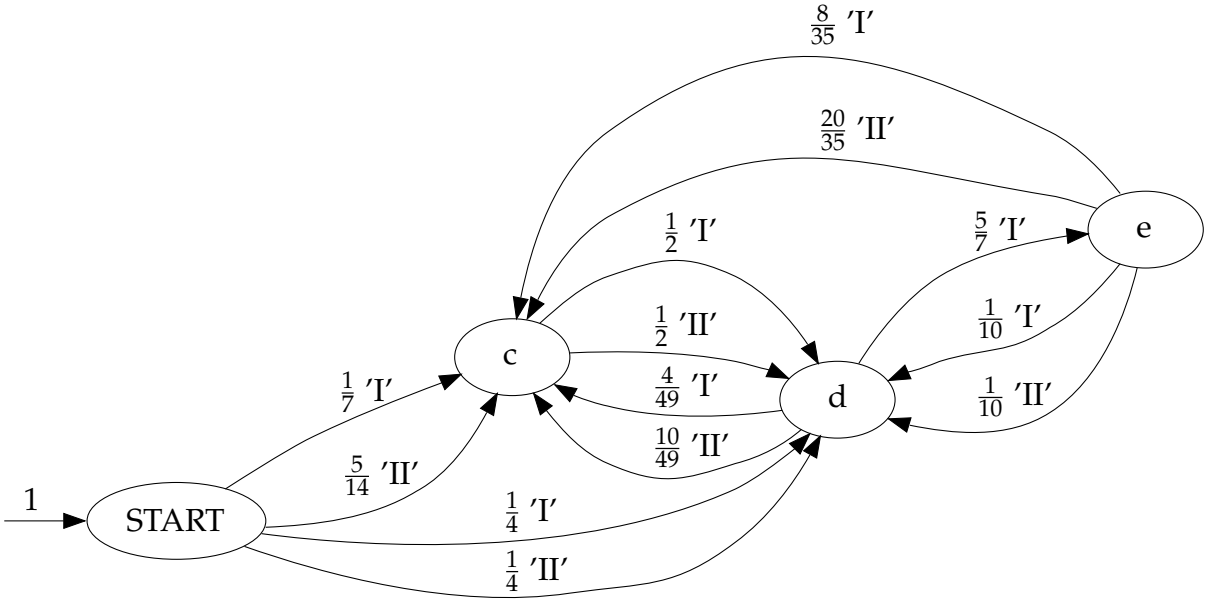


Figure 3.8: A probabilistic finite automaton

Note that a hidden Markov model can be converted to a PFA which determines the same probability distribution. When converting an HMM to a PFA, we use an identical set for Σ and the state space becomes $Q' = Q \cup \{q_{start}\}$. We obtain the transition probability function of the PFA by calculating it over $q_i \in Q'$, s $\in \Sigma$ and $q_j \in Q$ as shown below:

$$p(q_i, s, q_j) = \begin{cases} \pi(q_j) \times b(q_j, s) & \text{if } q_i = q_{start}; \\ a(q_i, q_j) \times b(q_j, s) & \text{otherwise.} \end{cases}$$

We define the initial probabilities of the equivalent PFA as $\pi(q_{start}) = 1$, and all other states have intial probability zero. Figure 3.8 shows a PFA equivalent to the HMM in Figure 3.5.

3.10 Final States

Final or acceptance states in an automaton require the automaton to be in one of these states after the processing of a string, in order to accept the given string. The automata described in the previous sections did not include a final state, since they only modelled strings of a fixed length. This

implies that their distributions were normalised over strings with the same length. For example, all the probabilities of the strings generated from the Markov chain in Figure 3.1, of length two, sums to one, as shown below:

$$\begin{array}{rcl} P(cd) & = & \frac{1}{2} \times 1 \\ P(dc) & = & \frac{1}{2} \times \frac{2}{7} \\ P(de) & = & \frac{1}{2} \times \frac{5}{7} \\ \hline & & 1.0 \end{array}$$

The introduction of a final state normalises the probability distribution of generated strings over all lengths. This is necessary when generating strings of various lengths. The definition of a probabilistic finite automata with a final state is given below.

Definition 3.10.1 (Probabilistic finite automata with final state probabilities (Thollard *et al.*, 2005)) **-** A probabilistic finite automaton with final state probabilities (FPFA) is a 6-tuple $\langle$ Q,Σ,τ,π,p,f $\rangle$ where:

- Σ is a finite input alphabet;
- Q is the state space;
- $\tau \subseteq Q \times \Sigma \times Q$ is a set of transitions;
- π : Q $\rightarrow [0, 1]$ defines initial-state probabilities;
- p : $\tau \rightarrow [0, 1]$ defines transition probabilities;
- f : Q $\rightarrow [0, 1]$ defines the final-state probabilities.

The following constraints must be satisfied:

- $\sum_{q \in Q} \pi(q) = 1$
- for $q_i \in Q$ we have that $f(q_i) + \sum_{s \in \Sigma, q_j \in Q} p(q_i, s, q_j) = 1$

The behaviour of a single acceptance state can be simulated without adjusting the definitions and algorithms described in the previous sections. This is done by appending all training sequences with a final state and emission. This implies an implicit extension of the respective state space and alphabet. In Figure 3.9 an FPFA equivalent to the HMM Figure 3.5 is shown.

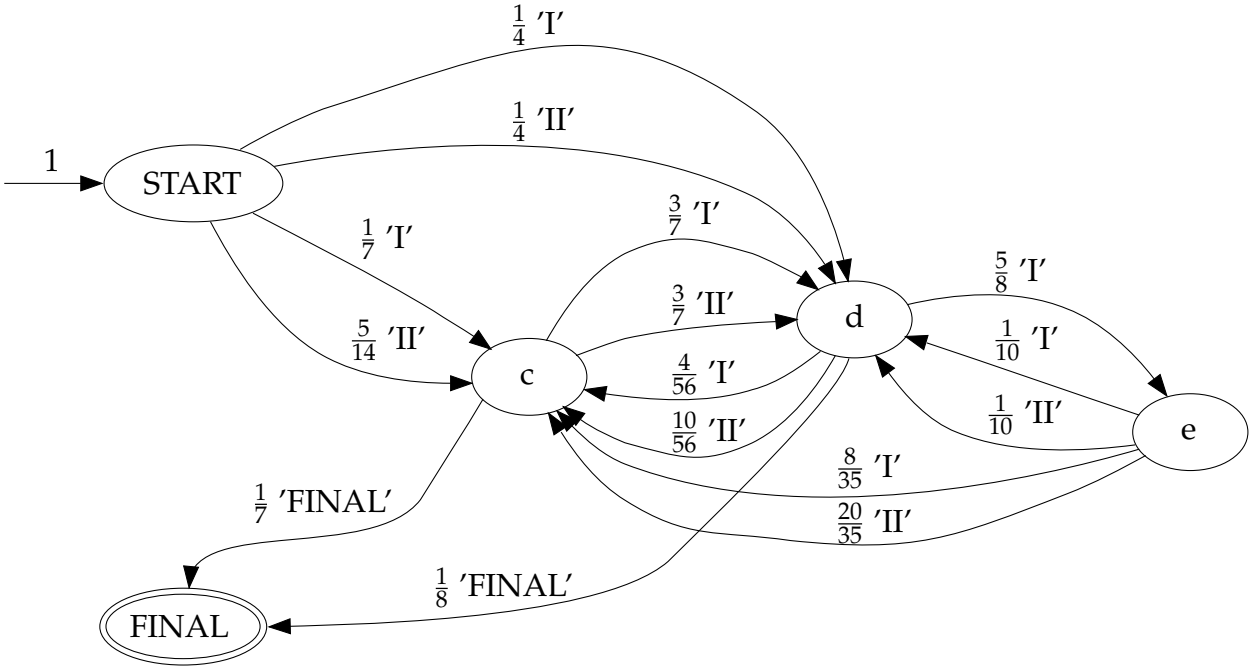


Figure 3.9: A probabilistic finite automaton with a final state

3.11 Conclusion

This chapter provided the required background on probabilistic automata and hidden Markov models. Our music generation/imitation system allows the user to use first, higher and mixed order Markov models. Our first approach to music generation uses first order Markov chains. Later we discuss how to use hidden Markov models to model the relationship between the melody and harmony of a composition.

Chapter 4

Literature Survey

Chapter 5

Implementation

Chapter 6

Evaluation

Chapter 7

Conclusion

Chapter 8

Future Work

Appendices

Appendix A

Tree Languages

A.1 Trees

A Tree is defined to be a connected acyclic graph in graph theory. In Computer Science trees are usually labelled, ordered and rooted. Rooted trees single out a node as being the root and any two connected nodes in the tree have an inherent parent-child relationship. In an ordered tree, the children of each node have a specific ordering.

Next we describe trees that are labelled and ranked. We denote the set of non-negative integers by $\mathbb{N}$. A ranked alphabet Σ is a finite alphabet that is partitioned into disjoint subsets Σ_k, for $k \in \mathbb{N}$. Thus $\Sigma = \cup_{k \in \mathbb{N}} \Sigma_k$ and $\Sigma_i \cap \Sigma_j = \emptyset$ if $i \neq j$. The rank of a node is the number of children of the given node. In labelled ranked trees each node of rank k is labelled by a symbol in Σ_k. Most Extensible Markup Language (XML) documents can be represented by unranked trees, except when links are used.

The tree in Figure A.1 has the signature {s : 2, 𝅗𝅥 : 0, ♩ : 0}, and it describes one measure in $\frac{4}{4}$ time. This tree represents a quarternote followed by a halfnote followed by a quarternote. Using the notation $a[t_1, \ldots, t_k]$ to denote a tree with root node labelled by the symbol a of rank k, and with subtrees $t_1, \ldots, t_k$, and using simply a if $k = 0$, the tree in Figure A.1 is denoted by s [s[♩, 𝅗𝅥], ♩]. In essence one can interpret nodes labelled by s in Figure

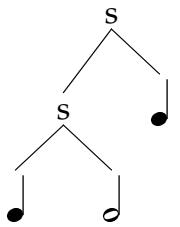


Figure A.1: A sample tree over the ranked alphabet {s: 2, 𝅗𝅥 : 0, ♩ : 0 }

A.1 as a way to convert unranked trees, with notes at the leaves, to ranked binary trees.

Definition A.1.1 (Set of trees over Σ, or T_Σ) **-** (Drewes, 2006) Let Σ be a signature. The set T_Σ of all trees over Σ is the smallest set of strings such that $t_1, \ldots, t_n \in T_\Sigma$ implies that $f[t_1, \ldots, t_n] \in T_\Sigma$, for every $f \in \Sigma^{(n)}$.

Note that all trees consisting of only a root node labelled by a symbol of rank 0 in Σ, is by definition in T_Σ. The trees $t_1, \ldots, t_n$ are called direct subtrees of $f[t_1, \ldots, t_n]$. The generation of trees by grammars are discussed next.

A.2 Regular Tree Grammars

Context-free grammars, the formal equivalent of the Backus-Naur formalism (Ford, 2004), are well-known to most Computer Scientists. The Backus-Naur formalism is often used to describe the syntax of programming languages. A context-free grammar is a finite set of rules that generates a language of strings. This formalism was proposed by Noam Chomsky (Chomsky, 1956). In a similar way, a regular tree grammar (RTG) is a finite set of rules that generates a language of trees (Drewes, 2006).

Definition A.2.1 (Regular Tree Grammars (RTG's)) **-** (Drewes, 2006) A regular tree grammar is a tuple G = (N,Σ,R,S) consisting of

- a finite alphabet N of nonterminals of rank 0;
- a finite output alphabet Σ, disjoint from N, whose elements are called terminals;

- a finite set R of rules of the form $A \rightarrow t$, where $A \in N$ and $t \in T_{\Sigma \cup N}$; and
- an initial nonterminal $S \in N$.

Regular Tree Grammars generate trees by using the rules of the grammar in a sequence of derivation steps.

Definition A.2.2 (Regular Tree Grammar Derivations) **-** (Comon *et al.*, 2007) Let G = (N, Σ, R, S) be a regular tree grammar. For trees s, s′ $\in T_{\Sigma \cup N}$, there is a derivation step s $\Rightarrow_G$ s′ (or simply s $\Rightarrow$ s′), if:

- s is a tree which has at least one leaf node labelled by a nonterminal A.
- there is a rule A $\rightarrow t \in R$, and
- s′ is the same tree as s, except that one of the leaf nodes labelled by A is replaced by the tree t.

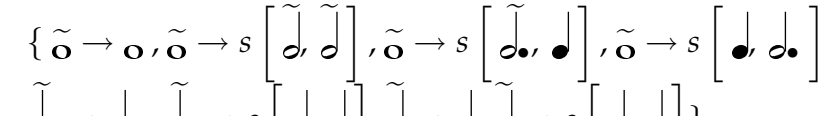
A sequence $t_0 \Rightarrow t_1 \Rightarrow \cdots \Rightarrow t_n$ of n derivation steps ($n \in \mathbb{N}$) is denoted by $t_0 \Rightarrow^n t_n$ and derivations of any length by $t_0 \Rightarrow^* t$. The regular tree language generated by G, denoted by $L(G)$, is the set of trees $\{t \in T_\Sigma | S \Rightarrow_G^* t\}$.

In Example A.2.1 we give an example of a derivation in an RTG.

Example A.2.1 (RTG derivations) **-** In this example we give an RTG which generates the tree in Figure A.1.

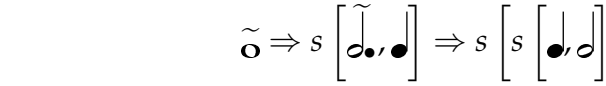
$N =$ $\{$  $\}$

$\Sigma =$ $\{$ $s : 2$,  $: 0$,  $: 0$,  $: 0$,  $: 0$ $\}$

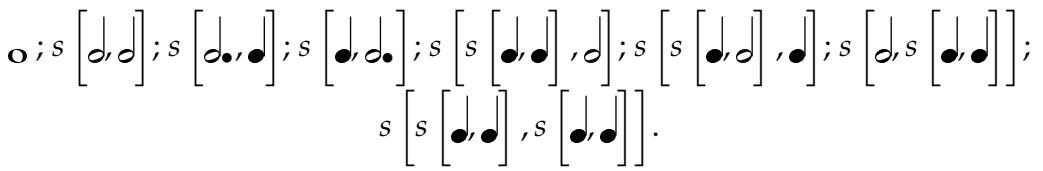
$R =$ $\{$  $\}$

$S =$ 

The RTG above can generate the tree in Figure A.1 as follows:



This is not the only tree that can be generated from this grammar. Here are all the other possibilities:



A.3 Top-Down Tree Transducers

Ranked trees, regular tree grammars and top-down tree transducers play an essential role in the music generation system Willow (Högberg, 2005). Top-down tree transducers are the tree analogues of string transducers. Thus a top-down tree transducer takes a tree as input and produces a tree or nothing as output. Tree transducers (TDs) have both input and output alphabets.

Definition A.3.1 (Tree transducer) **-** (Comon *et al.*, 2007) A top-down tree transducer (TD) is a tuple td = (Q, Σ, Δ, R, q_0), where

- Q is a finite ranked alphabet of states, all of rank one;
- Σ and Δ are finite ranked input and output alphabets, respectively;
- R is a finite set of *rewrite rules* of the form $q[a[x_1, \ldots, x_k]] \rightarrow t$, where $a \in \Sigma_{(k)}$, $q \in Q$, and $t \in T_\Delta(Q(X_k))$;
- $q_0 \in Q$, is the *initial state*.

We briefly explain the notation $T_\Delta(Q(X_k))$ used in the definition above. Firstly, $Q(X_k)$ denotes the set of trees consisting of a state in Q as root node and a variable in X_k as only child, where X_k is the set of variables $\{x_1, \ldots, x_k\}$. A tree t in $T_\Delta(Q(X_k))$ is obtained by taking a tree s in T_Δ and replacing (perhaps) some (or even all) of the leaf nodes of s by trees in $Q(X_k)$.

Next we describe the mechanism with which a top-down tree transducer td=(Q,Σ,Δ,R,q_0) computes output trees from input trees. Let $s \in T_\Sigma$ be an input tree. The computation starts with the tree $q_0[s]$. By $q_0[s]$ we mean the tree with the state q_0 as root and with the tree s as the only child of the root node. Assume that s is given by $a[t_1, \ldots, t_k]$. Next we take any rule

in R of the form $q_0[a[x_1, \ldots, x_k]] \rightarrow t$, where a is the label of the root node of s, and k is the rank of a. If no such rule exists, the transducer does not produce any output when given the input tree s. Note that in the special case where $k = 0$, we have that $t \in T_\Delta$. Since $t \in T_\Delta(Q(X_k))$, the tree contains possibly some of the variables $x_1, \ldots, x_k$ at the leaf nodes. A given variable may also appear at more than one leaf node. When we apply the rule $q_0[a[x_1, \ldots, x_k]] \rightarrow t$ to the tree $q_0 s$, we obtain the tree $t[\![t_1, \ldots, t_k]\!]$. We denote by $t[\![t_1, \ldots, t_k]\!]$ the tree that is obtained by replacing x_i in t by t_k. Note that when we replace x_i by t_i in t, we obtain a tree with a state in Q above each t_i. At each node that is labelled by a state in $t[\![t_1, \ldots, t_k]\!]$, we repeat the rewriting process that was used at the root of $q_0 s$. We repeat this process until we obtain a tree t in T_Δ. We denote the computation that takes s as input and produces t as output by $s \Rightarrow_{td} t$. Also, by $td(s)$ we denote the set $\{t \in T_\Delta | q_0 s \Rightarrow_{td} t\}$. In other words, $td(s)$ is the set of all possible trees in T_Δ that can be obtained if we start with $q_0 s$ and apply the rules in R until we obtain a tree in T_Δ.

In the next example we show how a given transducer transforms the tree in Figure A.1 in such a way that it contains multiple phrases. The parents of the leaf nodes of the trees obtained as output from the tree transducer are also marked by pitch values.

Example A.3.1 - This example gives a non-deterministic total top-down tree transducer which transforms the tree from Figure A.1 to have pitch and multiple phrases. This transducer creates two copies of the input tree with phrase as root. Next it places the pitches C, E or G on the yield of the tree, always starting with C.

$$
\begin{aligned}
\Sigma = \quad & \{\, s:2,\ \mathbf{o}:0,\ \mathbf{d}.:0,\ \mathbf{d}:0,\ \mathbf{q}:0 \,\} \\
\Delta = \quad & \{\, \mathsf{phrase}:2,\ s:2,\ \mathbf{o}:0,\ \mathbf{d}.:0,\ \mathbf{d}:0,\ \mathbf{q}:0,\ c:1,\ e:1,\ g:1 \,\} \\
Q = \quad & \{START, C, E, G\} \\
R = \quad & \{START[x_1] \rightarrow \mathsf{phrase}[C[x_1], G[x_1]]\,, \\
& C[s[x_1,x_2]] \rightarrow s[C[x_1], E[x_2]]\,, \\
& E[s[x_1,x_2]] \rightarrow s[E[x_1], G[x_2]]\,, \\
& G[s[x_1,x_2]] \rightarrow s[G[x_1], G[x_2]]\,, \\
& G[s[x_1,x_2]] \rightarrow s[G[x_1], C[x_2]]\,, \\
& C[\mathbf{o}] \rightarrow c[\mathbf{o}]\,,\quad C[\mathbf{d}] \rightarrow c[\mathbf{d}]\,,\quad C[\mathbf{d}.] \rightarrow c[\mathbf{d}.]\,,\quad C[\mathbf{q}] \rightarrow c[\mathbf{q}]\,, \\
& E[\mathbf{o}] \rightarrow e[\mathbf{o}]\,,\quad E[\mathbf{d}] \rightarrow e[\mathbf{d}]\,,\quad E[\mathbf{d}.] \rightarrow e[\mathbf{d}.]\,,\quad E[\mathbf{q}] \rightarrow e[\mathbf{q}]\,, \\
& G[\mathbf{o}] \rightarrow g[\mathbf{o}]\,,\quad G[\mathbf{d}] \rightarrow g[\mathbf{d}]\,,\quad G[\mathbf{d}.] \rightarrow g[\mathbf{d}.]\,,\quad G[\mathbf{q}] \rightarrow g[\mathbf{q}]\} \\
q_0 = \quad & START
\end{aligned}
$$

A sample transformation of a tree with this top-down tree transducer is shown below:

$$
\begin{aligned}
& s\left[s\left[\mathbf{q}, \mathbf{d}\right], \mathbf{q}\right] \\
\Rightarrow\ & START\left[s\left[s\left[\mathbf{q}, \mathbf{d}\right], \mathbf{q}\right]\right] \\
\Rightarrow\ & \mathsf{phrase}\left[C\left[s\left[s\left[\mathbf{q}, \mathbf{d}\right], \mathbf{q}\right]\right], G\left[s\left[s\left[\mathbf{q}, \mathbf{d}\right], \mathbf{q}\right]\right]\right] \\
\Rightarrow\ & \mathsf{phrase}\left[s\left[C\left[s\left[\mathbf{q}, \mathbf{d}\right]\right], E\left[\mathbf{q}\right]\right], s\left[G\left[s\left[\mathbf{q}, \mathbf{d}\right]\right], C\left[\mathbf{q}\right]\right]\right] \\
\Rightarrow\ & \mathsf{phrase}\left[s\left[s\left[C\left[\mathbf{q}\right], E\left[\mathbf{d}\right]\right], e\left[\mathbf{q}\right]\right], s\left[s\left[G\left[\mathbf{q}\right], G\left[\mathbf{d}\right]\right], c\left[\mathbf{q}\right]\right]\right] \\
\Rightarrow\ & \mathsf{phrase}\left[s\left[s\left[c\left[\mathbf{q}\right], e\left[\mathbf{d}\right]\right], e\left[\mathbf{q}\right]\right], s\left[s\left[g\left[\mathbf{q}\right], g\left[\mathbf{d}\right]\right], c\left[\mathbf{q}\right]\right]\right]
\end{aligned}
$$

A.4 Conclusion

This Appendix gave a short introduction to tree languages. A more detailed introduction can be found in Drewes (2006). We discussed how trees can be generated by regular tree grammars and transformed by top-down tree transducers. Willow (Högberg, 2005) uses tree grammars and tree transducers to implement a rule-based system for algorithmic composition. XML documents, which are used in our music generation system, can often be considered as trees.

List of Figures

Nomenclature

Abbreviations

ATTLIST	attribute list
bpm	beats per minute
CDATA	character data
DOM	Document Object Model
DPFA	Deterministic Probabilistic Finite Automata
DTD	Document Type Definition
FPFA	Probabilitic Finite Automata with a final state
HMM	Hidden Markov Model
Hz	Hertz
MC	Markov Chain
MIDI	Musical Instrument Digital Interface
PAC	Probably Approximately Correct
PCDATA	parsed character data
PDF	Portable Document Format
PFA	Probabilistic Finite Automata
PS	PostScript
PSA	Prediction Suffix Automata
PST	Prediction Suffix Tree
RTG	Regular Tree Grammar
TD	Tree Transducer
URL	Uniform Resource Locator
UTF	Unicode Transformation Format

wfsa	weighted finite state automata
wfst	weighted finite state transducer
XHTML	Extensible Hypertext Markup Language
XML	Extensible Markup Language
XPATH	Extensible Markup Language Path Language
XSL	Extensible Stylesheet Language
XSL-FO	Extensible Stylesheet Language Formatting Objects
XSLT	Extensible Stylesheet Language Transformation

List of References

Adams, R. (2000). Musictheory.net. `http://www.musictheory.net`.

Chomsky, N. (1956). Three models for the description of language. *IEEE Transactions on Information Theory*, vol. 2, no. 3, pp. 113–124.

Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S. and Tommasi, M. (2007). Tree automata techniques and applications. Available on: `http://www.grappa.univ-lille3.fr/tata`. Release October 12th, 2007.

Drewes, F. (2006). *Grammatical Picture Generation – A Tree-Based Approach*. Texts in Theoretical Computer Science. An EATCS Series. Springer.

Dupont, P., Denis, F. and Esposito, Y. (2005). Links between probabilistic automata and hidden markov models: probability distributions, learning models and induction algorithms. *Pattern recognition*, vol. 38, no. 9, pp. 1349–1371.

Ford, B. (2004). Parsing expression grammars: A recognition-based syntactic foundation. In: *Symposium on Principles of Programming Languages*, pp. 111–122. ACM Press.

Hart, P., Nilsson, N. and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107.

Högberg, J. (2005). Wind in the willows. Report UMINF 05.13, Umeå University.

Huang, X., Acero, A. and Hon, H. (2001). *Spoken language processing: A guide to theory, algorithm, and system development*. Prentice Hall PTR Upper Saddle River, NJ, USA.

Levitin, D.J. (2006 August). *This is Your Brain on Music*. 1st edn. Dutton, a member of Penguin Group (USA) Inc.

McNab, R.J., Smith, L.A., Witten, I.H. and Henderson, C.L. (2000 April). Tune retrieval in the multimedia library. *Multimedia Tools and Applications*, vol. 10, no. 2-3, pp. 113 – 132.

Ottman, R.W. (1983). *Elementary Harmony: Theory and Practice*. 3rd edn. Englewood Cliffs, N.J. : Prentice-Hall.

Parsons, D. (1975 January). *The directory of tunes and musical themes*. Spencer Brown.

Rabiner, L. (1990). A tutorial on hidden Markov models and selected applications in speech recognition. *Readings in speech recognition*, vol. 53, no. 3, pp. 267–296.

Randel, D.M. (2003 November). *The Harvard Dictionary of Music*. 4th edn. Belknap Press of Harvard University Press.

Ron, D., Singer, Y. and Tishby, N. (1996). The power of amnesia: Learning probabilistic automata with variable memory length. *Machine learning*, vol. 25, no. 2, pp. 117–149.

Schwardt, L. (2007 December). *Efficient Mixed-Order Hidden Markov Model Inference*. Ph.D. thesis, University of Stellenbosch.

Thollard, F., de la Higuera, C. and Carrasco, R. (2005). Probabilistic Finite-State Machines-Part I. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 7, pp. 1013–1025.

Valiant, L. (1984). A theory of the learnable. *Communications of the ACM*, vol. 27, no. 11, p. 1142.

Ziv, J. and Lempel, A. (1978). Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, vol. 24, no. 5, pp. 530–536.