Wind in the Willows

Johanna Högberg
Department of Computing Science, Umeå University
S–901 87 Umeå, Sweden
johanna@cs.umu.se

Abstract. We implement a rule-based system for algorithmic composition. This system, that we call Willow, resides in the TREEBAG environment and consists of a sequence of formal devices, familiar from the field of tree grammars and tree transducers. Since these devices are well studied, we can apply known results to derive the descriptive complexity of the system as a whole.

Copyright © 2005

UMINF 05.13 ISSN 0348-0542

1 Introduction

The aim of this paper is to generate music in an algorithmic fashion, thus contributing to the field of computer aided music generation. For other work in this area, see e.g. [Roads, 1979], [Baroni, 1983], [Lerdahl and Jackendoff, 1977], [Prusinkiewicz, 1986], and [Lindblom and Sundberg, 1970]. Our first task shall be to compile a corpus of precepts for musical composition. At the point when this is complete, we proceed to draw – guided by the corpus – the components of a rule-based system for algorithmic composition. In this transition – from precepts to formal rules – we will put semantics aside and concern ourselves only with issues of syntax. In other words, our rules attempt to capture typical structural aspects of the music to be generated (i.e. its syntax), but disregard its meaning (i.e. semantics). This is in line with the following remark that was made in [Dempster, 1998].

> ... while music typically has very elaborated and regular structures – much like language – these structures do not apparently originate from nor are they in the service of the need to encode meanings – exactly unlike language.

In its original context, the quotation attempts to refute not only the presence of semantics in music, but also of syntax. The implication that, because music does not have semantics, it cannot be described by a grammar, is questioned in a convincing manner in [Wiggins, 1998]. He writes:

> The work of [Bel and Kippen, 1992] and [Ponsford et al., 1999] (among many others) indicates clearly that grammars of different kinds *can* successfully capture conventionally agreed musical structure, rhythmically and harmonically. No further argument, surely, is needed.

Wiggins also discusses what happens to the syntax–semantics dichotomy, when imported into a music theoretic context: Although there is a clear distinction between syntax and semantics in the linguistic sense, this distinction may be blurred or altogether disappear when we apply these concepts to music. He concludes that it might be that music's meaning is *in* the structure, rather than being *carried by* its structure. If this conjecture proves correct, then we can safely forget about semantics in the traditional, linguistic, sense, as it is implicitly treated along with the syntax.

Our origin is set in Computer Science, and within this field, the investigation of formal methods for the specification and manipulation of syntax is considered part of Formal Language Theory (or FLT, when abbreviated). The conventional way to represent syntax is by means of a tree structure (e.g. as a parse or a derivation tree). The object whose syntax is to be studied is deconstructed in a hierarchical manner, resulting in a set of constituents, arranged into levels of increasing detail. To clarify, we show how the deconstruction process can be applied to a musical extract.

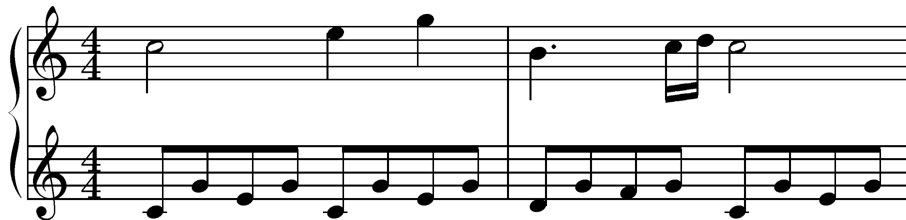


Figure 1: The first two bars of the sonata in C-major KV545 by Mozart.

Example 1.1 Figure 1 shows the first two bars of the sonata in C-major KV545 by Mozart [Egler et al., 2003]. We project a hierarchy onto the piece; at the topmost level we place the whole of the extract, immediately below it, two voices of two measures each. The measures in turn are composed of (concatenations of) notes of various durations. At the lowest levels there are pitches. This hierarchy, together with a discretisation of certain attribute values (i.e. we only permit a

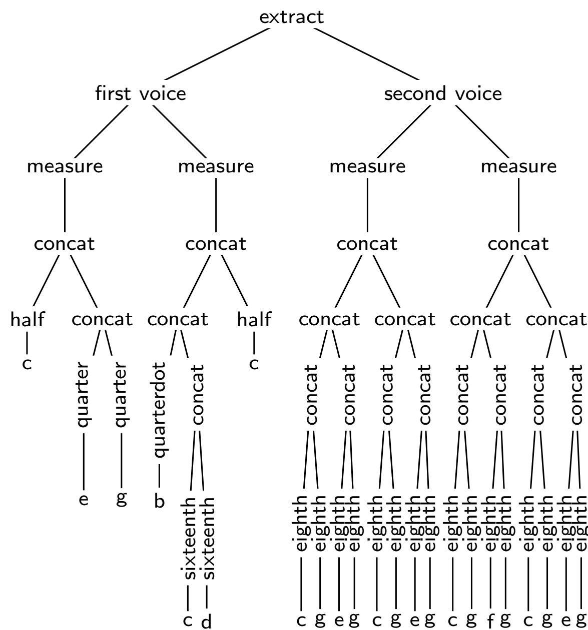


Figure 2: Score 1, expressed as a tree.

finite number of pitches, durations, meters etc.) allows us to represent the piece as the (ordered, labelled) tree of Figure 2. This particular parse is clearly a simplification, but it is hoped that the reader finds the example realistic enough to accept the basic idea.

The branch of FLT that investigates algorithmic systems whose purpose it is to generate and transform trees, is commonly known as the theory of tree grammars and tree transducers. Here, we find a large variety of well-studied devices, united in their modus operandi – with few exceptions, they all generate or transform by iteratively applying a finite set of production or rewrite rules. The applications of three different formal devices are discussed in Examples 4.9, 4.11 and 4.14.

The theory of tree grammars and tree transducers has to some extent been implemented in TREEBAG [Drewes, 2005], a system that allows for tree grammars and tree transducers to be combined, in order to produce trees. We shall use this system to connect a sequence of formal devices, and interpret the resulting trees as representations of music. The devices included in the sequence have been kept as simple as possible. As will be argued in Sections 3 and 5, if the system becomes too complex, it ceases to be plausible in the sense that it could illustrate how a human listener perceives music. An overly intricate system for algorithmic composition also has little, if any, scientific value; few would be surprised if a simple tune could be generated by, for example, a Turing machine.

The sequence of devices that constitute Willow can be compared to an assembly line, the workers along the line being familiar from formal language theory: regular tree grammars and top–down tree transducers. Since these devices are all well studied, we can use known results to analyse and control their work, and since the workers are separate entities, they can be replaced one at a time (a modified truth, see Section 5). Say for example that one worker, perhaps Mr Jazz-transducer, assigns chords mimicking Lillian Hardin, but that the user prefers pop. Then the user could simply substitute Mr Justin-transducer for Mr Jazz-transducer and have things her way. Similarly Mr Walz could substitute Mr March, Ms Guitar Mrs Piano, and Lady Choir old Sir Solo.

Another benefit of this system, which we call Willow, is that the generated music could take the rôle of raw material for a human composer. A computer executing the system could generate an endless supply of themes and tunes, without hesitation or embarrassment. It would then be up to the composer to pick and choose among the material as he pleases, and hopefully there will be some parts that appeal to him. These parts could then be fed into the next step in the construction line; the computer could for example suggest an accompaniment or a decoration. Again the human composer would be able to have his say-so, providing the computer with guidance and feedback.

1.1 Resources

The www site `http://www.cs.umu.se/~johanna/willow/` is designated to project Willow, and provides the following resources:

1. Music generated by Willow, in the form of scores and audio files. The scores are available in postscript and pdf format, and can be viewed with Ghostview and Acrobat Reader, respectively. The audio files are in midi format, and can be played by programs such as TiMidity and Windows Media player.
2. The Treebag system, as a `.jar` but also the source files are available, including the new classes `ScoreDisplay` and `IgnorantTransducer`.
3. This technical report in postscript and pdf format.

1.2 Outline

This paper is structured as follows. In the subsequent pages that constitute Section 2, we introduce basic concepts and notational conventions from the field of music theory. Section 3 gives an overview of previous work, while Section 4 provides preliminaries concerning formal language theory. Then follows Section 5, describing the actual implementation of Willow – an effort to model precepts for music composition using formal devices. Theoretical issues such as descriptive

complexity and statistical distribution are addressed in Section 6. The topic of the last section of this paper, Section 7, is future work.

2 Music theory

Contrary to what many believe, exactly what constitutes a tone, a scale or a melody is quite arbitrary and differs between cultures. Though our impression is that all varieties have their own attractions, we have chosen to study the principles used in western tradition, quite simply because it is closest at hand. These precepts (together with an additional few, for various reasons withheld until Section 5) constitute the corpus that Willow was built upon. For those who wish for a more thorough introduction to music theory in general, we would like to recommend [Kerman and Tomlinson, 2000]. There are also many helpful resources on the Internet; [Alvira, 2004], [Blood, 2004], and [Adams, 2004], to mention a few. The reader already familiar with the foundations of music theory, including the triads of the major scale, chord progressions, meter and rhythm, may wish to skip this section.

2.1 Notes

In the context of this paper, a *tune* will be a sequence of notes. A note is characterised by its *duration*, *accent*, *timbre* and *pitch*. The first of these properties, duration, is the ratio of the length of a note to the length of an arbitrary, but fixed, reference note. Furthermore, a note whose duration is equal to that of the reference note, is called a *whole* note. A note whose duration is but half of that of the reference note, is called a *half* note, and so forth. Accent is also a relative property; the note is accented if it is played, for example, louder or longer than any surrounding note, but more on this in Section 2.5.

The timbre, or tone colour, of a note is the subjective quality which (easily put) lets us distinguish between instruments. When a narrator wants to convey the aromas of a fine wine or the colours of an impressionist painting, she may resort to a quite imaginative vocabulary. This is also the case when timbre is described; flutes are often said to have bright timbre, while clarinets have a deep, rich timbre. The sound of a Stradivarius has even been said to ‘move like candlelight’.

The pitch (or tone) of a note is a frequency in the *audible interval* – the range of frequencies spanning from 20 to 20.000 Hz. The ratio between two tones is called an *interval*. Naturally there are an infinite number of tones, but only a few are (deliberately) used. In western culture these happen to be the tones in the *chromatic scale*. When the tone of a note is its only relevant attribute, we may simply substitute tone for note.

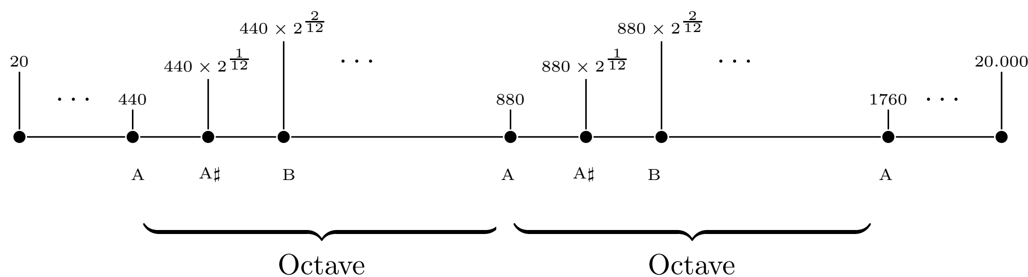


Figure 3: The audible frequency interval spans from 20 to 20.000 Hz, here divided into octaves.

2.2 Scales

A *scale* is a set of tones, and just as there are an infinite number of tones, so are there an infinite number of conceivable scales. But then again, only some of these are actually used in practice. The most important of these is probably the chromatic scale, which is constructed as follows: First a reference tone is ordained, and normally this would be the so-called *a above middle c*, an alias for 440 Hz (for convenience, we henceforth restrict ourselves to chromatic scales built around *a* above middle *c*). By doubling this pitch, the tone one *octave* higher is found, and by dividing it by two, the the tone one octave lower. The whole of the audible interval is split into octaves, and every octave is divided into 12 tones, in such a way that the ratio between two consecutive tones amounts to $2^{1/12}$. The construction is illustrated in Figure 3.

There are two conventional ways to address the tones of an octave. One can either use absolute names and refer to the twelve tones as a, $a\sharp$, b, c, $c\sharp$, d, $d\sharp$, e, f, $f\sharp$, g, and $g\sharp$, respectively, or one can use relative names. In the latter case, a reference tone is selected (this can be any tone in the chromatic scale), and the remaining tones it are named after the intervals that they are positioned at. Figure 4 (a) and (b) illustrate the absolute and the relative approach, respectively. Mind that the reference tone used Figure 4 (b) is apparently c, but could be any tone in the chromatic scale. If, instead, the rôle of reference tone is assigned to $f\sharp$, then all names would be shifted seven steps to the right. To find names for the remaining tones of the chromatic scale, the absolute names can simply be transposed any number of octaves. As for relative names; either one can keep counting – octave, minor ninth, major ninth, minor tenth, and so forth – or simply add a prefix, e.g. octave, octave minor second, octave major second, octave minor third, and so on. Aside: In Figure 4, the tone names are set against a keyboard, indicating which key corresponds to which tone. A reader not familiar with the layout of a keyboard, may ignore this and simply focus on the names and sequential order.

Conventionally, only a subset of the chromatic scale is used within a single tune.

How this subset is selected varies greatly, but in this article we will only discuss the selection that results in a *major scale*: First, an arbitrary tone from the chromatic scale is chosen, together with the tones that follow at intervals major second, major third, tritone, perfect fifth, major sixth and a major seventh. The tones of the major scale are also referred to as tonic, supertonic, mediant, subdominant, dominant, submediant and subtonic, respectively. Figure 5 shows the scale of C major. It is but a coincidence that only white keys are used; C major is in fact the only major scale that completely lacks black keys.

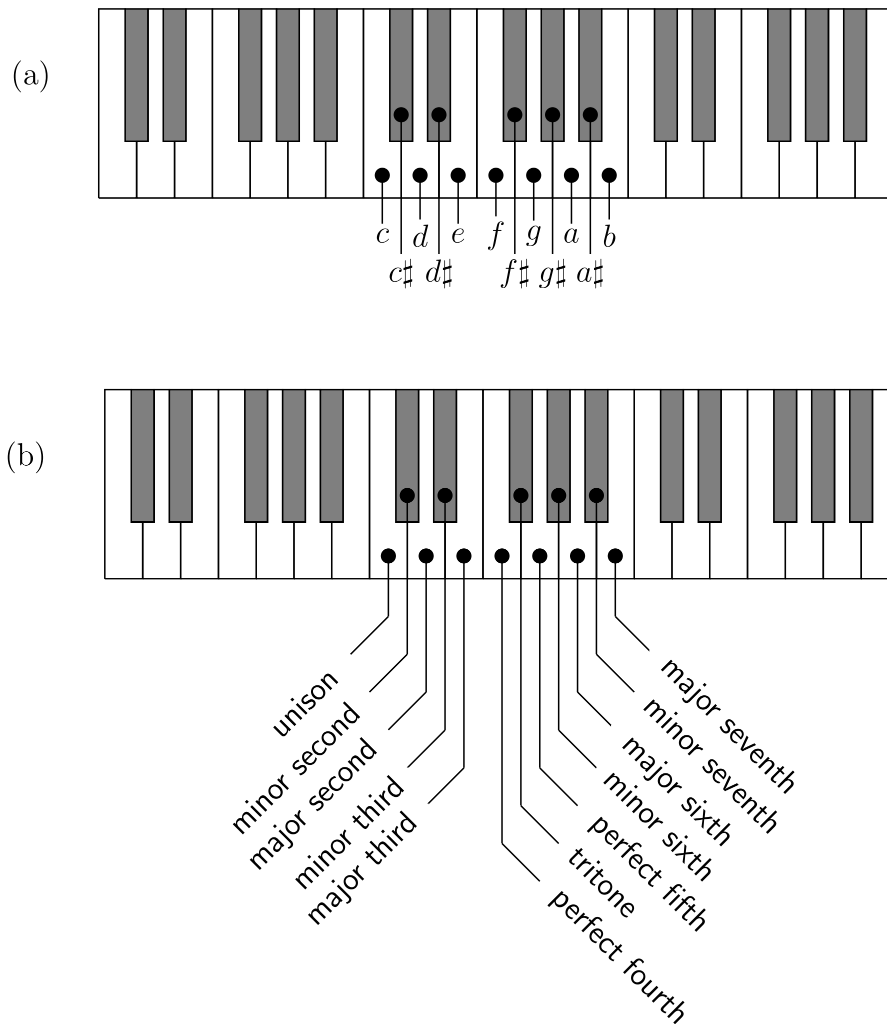


Figure 4: The absolute tone names of Figure (a) are repeated for every octave. When relative tone names are used, as in Figure (b), one simply counts the interval to a fixed reference tone.

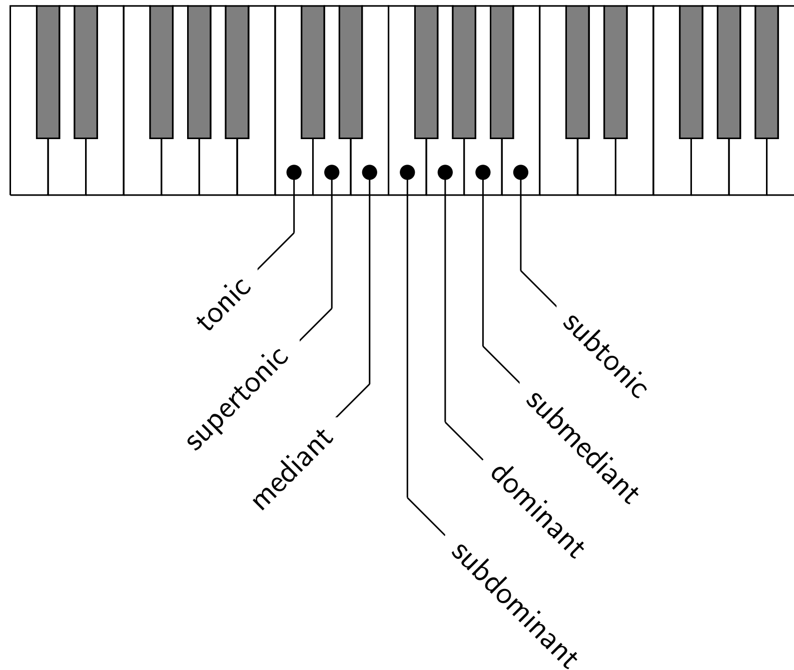


Figure 5: The tones that constitute the scale of C major.

2.3 Chords

A set of two or more notes, sounded simultaneously (or nearly so), constitutes a *chord*. For tunes written in a major scale there are a some conventional sets of 'allowed' chords. The most basic of these sets consists of the *triads in the major scale*, in which each of the seven tones in the major scale has been extended to a chord called a *triad*. In analytic notation the triads are referred to by roman numbers, for example, the triad that stems from the tonic (the first tone of the scale) is written I, while the triad that stems from the dominant (the fifth tone) is written V.

A triad consists, as the name suggests, of three tones, out of which the lowest tone belongs to the major scale and is called the *root* of the triad. The other two tones are named after their interval relative to the root. There are three varieties, as illustrated in Figure 6. Suppose that the root is followed by a major third and a perfect fifth, then the triad is a *major triad*. If there is instead a minor third and a perfect fifth, then we are looking at a *minor triad*. Finally, a *diminished triad* is a root followed by a minor third and a tritone. Both the analytic notation, i.e. the roman number, of minor and diminished triads are written in lower case, and if the triad is diminished then a degree sign is added. The triads of the major scale are: major, minor, major, major, major, minor, and diminished. In analytic notation I, ii, III, IV, V, vi, vii°.

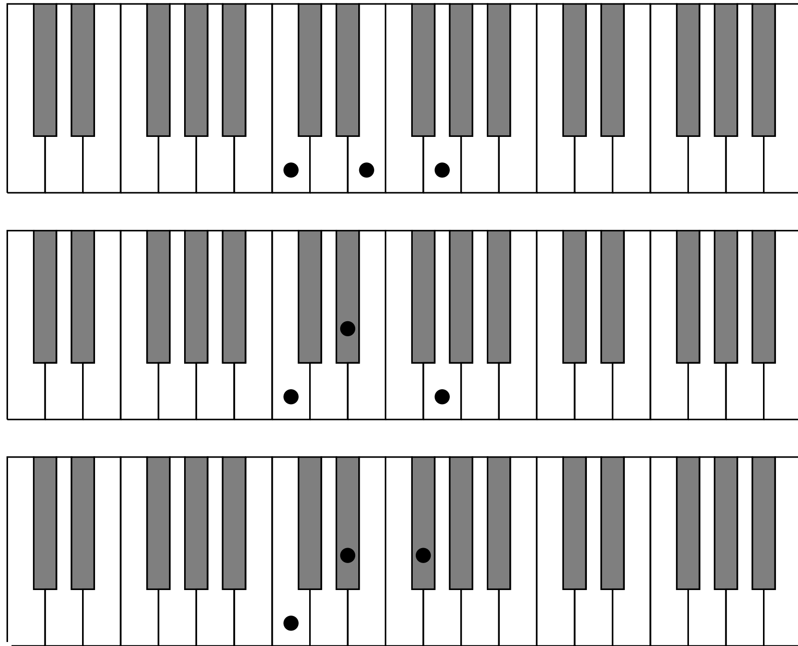


Figure 6: A major, minor and diminished triad, respectively.

Using only the triads of the major scale in musical composition is a bit like preparing a meal without adding any spices: A piece whose every chord belongs to the allowed set is likely to sound dull, and for this reason human composers often add a few extrinsic chords to give their tune a certain mood or flavour.

2.4 Chord progressions

A *chord progression* is a successive sequence of chords. Though the chords in a chord progression can be any triads in the scale, depending on the genre, some progressions are more likely than others. Some common chord progressions for a variety genres are collected and represented as graphs in Figure 7 on the next page. The reader may envision loops on all nodes – a tune may linger for a while on one chord before proceeding to the next. A common feature of these chord progressions is that they tend to start at the tonic, search their way towards the dominant while creating tension, and then fall back to the tonic, resolving the tension. Konecky [Konecky, 2004] conjectures that:

> Much of our enjoyment of music comes from the interplay between tension and relaxation or the resolving of tension.

This cycle of increasing and then decreasing tension is called a *phrase*. In fact, the relaxation of tension at the end of the phrase is so important that it has been given a name; the chord progression that ends a phrase is called a *cadence*.

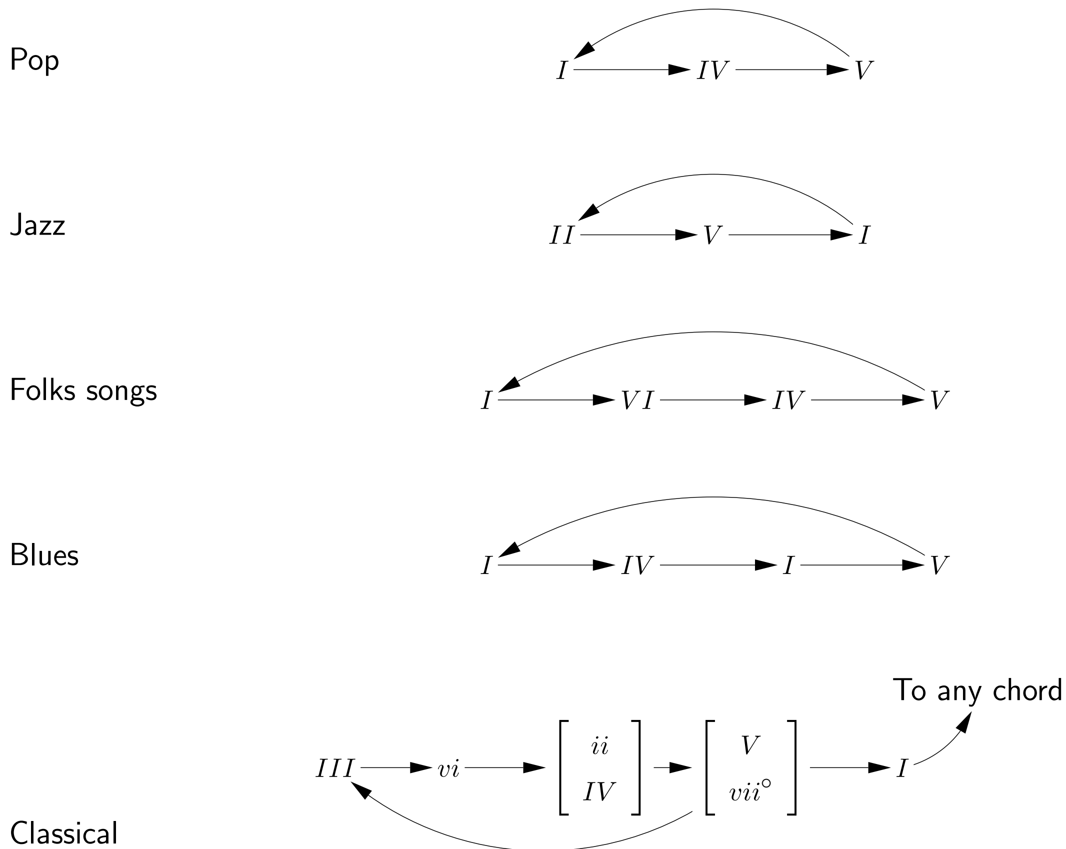


Figure 7: Common chord progressions in different genres.

2.5 Rhythm and meter

The *rhythm* of a tune is a description of how the tones and chords are accented and arranged in time. When governed by rule, it is called *meter* [Wik, 2005]. Although we have saved this concept for the last part in this section, it is probably the most prominent feature of music.

A meter is, simply put, a straightforward repetition of a *measure*; a short sequence consisting of one accented note and one or more less accented notes. As mentioned earlier, a note can be accented in several different ways: There can be one or more prior, shorter notes, the accented note can be played as a tonic or a dominant, or quite simply louder than its neighbouring notes. Within a

tune all measures have the same number of notes, defined by the *time signature*. The time signature also determines of what type these notes should be. By convention, each time signature is associated with one or more patterns of accent [Benward and Carr, 1999]. These patterns, can be read for beats of 2, 3 and 4 in Figure 8. To the left of each pattern is a small graph, and the reader is encouraged to pick one and move his or her hand in the air, in accordance with the arrows of that graph. Doing so might give an inkling of how the corresponding beat pattern should be accented – the greater the descent, the stronger the accent.

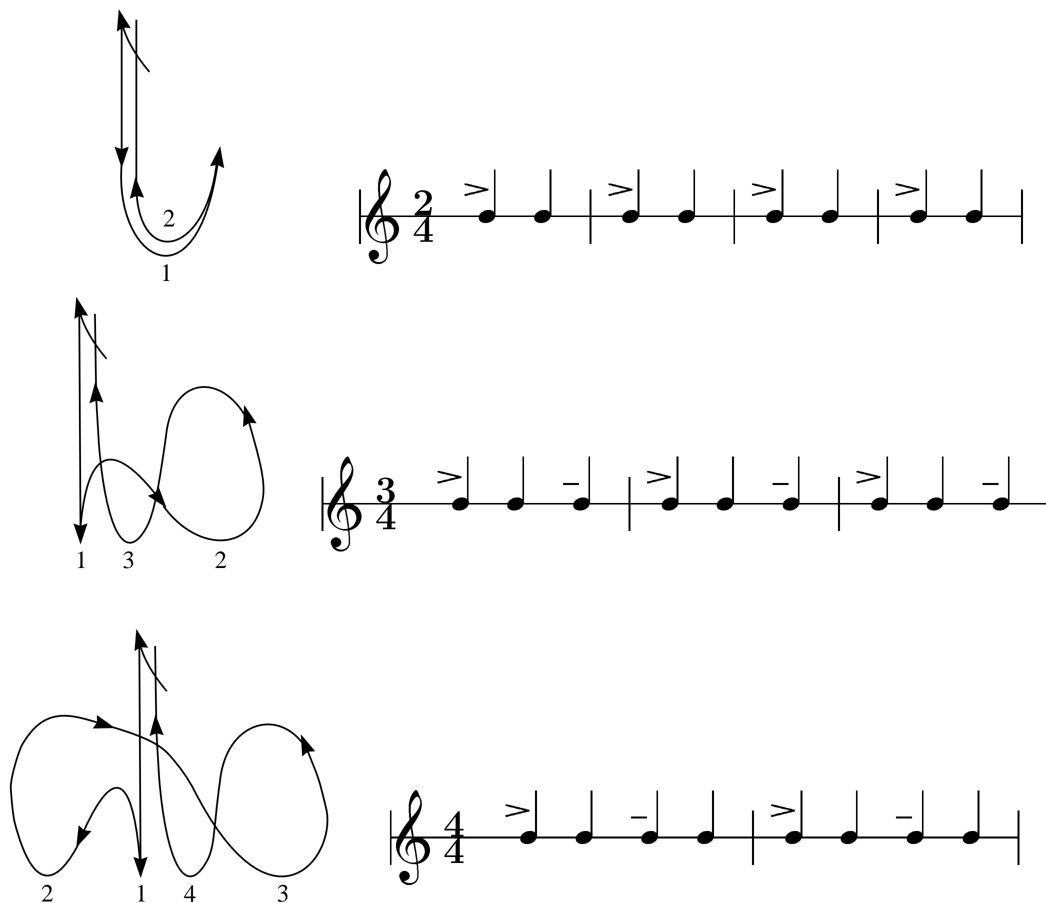


Figure 8: Accented notes in beat of 2, 3 and 4, respectively. If a note has the superscript $>$ it is strongly accented, $-$ medium accented, and if it has no superscript at all, then it is weakly accented.

3 Algorithmic Composition and Analysis

Wolfgang Amadeus Mozart – prodigy, musical genius, and incidentally, designer of a composing algorithm. The algorithm, 'Mozart's musikalisches Würfelspiel', is in fact a musical one-player game. The participant rolls a 12-sided dice to select a sequence of measures from a collection of snippets of sheet music. When the measures are assembled, they form a complete minuet, giving the participant the wonderful feeling of having composed a hereto unheard piece of music[1].

When a round of 'Mozart's musikalisches Würfelspiel' has resulted in a new minuet, who created the piece, the participant or Mozart himself? To find the answer, let us think of the sequence of dice rolls as an index into a huge library containing every minuet with thirty-two measures, each measure one of Mozart's original twelve. From this angle, the player seems to be little more than an explorer in a world of minuets, Mozart being its true creator.

Continuing along this line of reason, let us think of the set that consists of *all* pieces of music. We do not need to solve the labyrinthine problem of how to define music, to see that this set would be very large. Consider, on the other hand, the set that consist of all pieces of sound – music or otherwise. As extensive as the first set might be, the second set would be ever so much larger. In these terms, composition in the traditional sense (complete with muse, quill, and angst) is equivalent to locating a sound that is also music, whereas algorithmic composition is equivalent to selecting a subset of all pieces of sound, such that every piece therein is also music. When Mozart wrote his Würfelspiel he did not only compose a small number of snippets of music, but also the 34182189187166852111368841966125056 minuets that can (and if the player is persistent, will) result from his dice-throwing algorithm.

We might ask ourselves; for the hard up artisan, would an algorithm for composition not be something of a goose with golden eggs? There is a catch. Assume that somebody wrote an algorithm G, and that G could generate a superset of all possible sounds of length ten minutes or less, that could be achieved by playing with two hands on a piano. If certain limitations were allowed, e.g. that all characteristics of a human piano performance are discrete, this algorithm G could certainly be implemented in practice. With little doubt this G would, among other things, compose a piano-version of every short piece of contemporary western music. Not bad. But it would also produce a whole lot of noise, in fact, since there are so many more ways of using a piano to make noise then there are of playing proper music, the amount of actual music produced by G would be completely negligible[2]. Even Mozart's game, a very predictable algorithm by a great musician, may at times produce minuets that are closer to noise than

[1]For a digital implementation, see Mozart's musikalisches Würfelspiel [Chuang, 1995]

[2]'However many ways of being alive there are, it must be true that there are infinitely more ways of being dead' — Richard Dawkins

mastery. It must be concluded that although algorithms for musical composition can be very powerful in terms of productivity, this generative power will not help when control is lacking – we want the algorithms to compose music, but equally important, not to compose that which is not music.

Algorithmic composition, as a field of research, has drawn inspiration from a great variety of sources, one of these being cognitive linguistics. In the 1950s and 1960s Noam Chomsky conjectured that a natural language sentence has two structural levels: a deep structure carrying the semantic relations the underlies the sentence, and a surface structure that closely follows the phonological form of the sentence. Chomsky also defined the transformational grammar – a mapping from the deep structure to the surface structure, and postulated that no matter how different two languages may appear when ostensibly compared, their deep structures will be quite similar. A decade later these ideas had seeped into music analysis, making a (universal) grammar of music something of a Grail.

Whether a universal grammar of music exists, was (and still is) a question under much debate. If it does exist, then it would be the embodiment of most, if not all, of music analysis. The hypothetical grammar is expected to assign a structural description to any (anonymous) piece of music, but with this presumption trouble arises, for rarely do two people hear the same piece in precisely the same way. Nonetheless, it is argued in [Lerdahl and Jackendoff, 1977], there is normally a substantial agreement on what are the most natural ways to hear a piece. Hence, a grammatical theory of music must not only assign structural descriptions to musical pieces, but must also indicate how far these descriptions are from the 'preferred way' of hearing the piece. To avoid this assessment, one could settle for a weaker grammar, capable of generating every structural description that correspond to some piece of music, but that does not assign to each piece a unique structural description. This restriction, along with others that will be mentioned in the following, was applied to make the construction of Willow tractable.

Lerdahl and Jackendoff also require that the nature of the structural descriptions generated must be psychologically plausible, in that their complexity must result from the interaction of a fairly small number of processes, each of which taken in isolation is relatively simple. We do believe that the inner-workings of Willow adhere to this principle; surly the less then twenty components must be considered a small number, and td transducers are, as we shall see in Section 4, exceedingly simple devices.

Further constrictions on a generative grammar are identified in [Roads, 1979]. For practical purposes the fact that it is 'theoretically possible' to generate a certain structure does not suffice; there is always a sharp pragmatic distinction between what can be done with some elegance and what requires ad hoc patchwork to be accomplished. To be of real use a grammar must have an explanatory quality to facilitate analysis, while retaining enough expressive power to cope with a broad range of compositions. To bestow Willow with this explanatory quality, each

component is designed to add a distinct attribute (e.g. rhythm, scale, voices), and the devices used are as simple as possible (without defining new devices). These precautions render a system that as a whole is transparent and – it is hoped – comprehensible.

The construction of a grammar of music is discussed in [Baroni, 1983]: The author points out that whereas Chomsky's generative-transformational linguistics rests upon a firm foundation of structural linguistics (embodied in a large catalogue of phonetic and morphological features in the language), musicology lacks a comparable empirical base. In an attempt to attack this problem, we build Willow on a small, but traditionally accepted corpus, aware that in doing so we forfeit every claim on universality.

The grammar that Baroni constructs is divided into *micro structure* which concerns rule-based generation of phrases, and *macro structure* which describes a musical piece as a whole. At micro level he selects a set of pertinent features to define the basic units of his corpus: pitch (in the diatonic scale), note length (measured in semi-quavers), metrical position (out of a set of fixed positions), degree (one of the seven diatonic steps) and tonality (extracted from each choral). These five features, the author claims, are necessary and sufficient to establish a system of grammatical rules. Properties such as dynamics and timbre are left out on purpose.

The attempts described above have all been made of music theorists searching for grammars to describe music, but it is quite possible to approach the problem from the opposite direction: The generation of musical scores by means of L-systems – devices well-known from formal language theory – is the topic of [Prusinkiewicz, 1986], a paper motivated by issues of interest first and foremost to the FLT community. In the following sections, we take a similar point of view. Using well-known devices from FLT, we explore how and to what extent these can be used to generate music. In particular, this leads to insights regarding the descriptional complexity of different aspects of the generation of music scores.

4 Tree grammars and tree transducers

After two introductory chapters on music theory and algorithmic composition, we now turn our attention to the abstract structure *tree*, and some of the devices that can be used to generate and transform trees. For a more thorough introduction to the field, see for example [Gécseg and Steinby, 1984], [Gécseg and Steinby, 1997], [Fülöp and Vogler, 1998] or Appendix A of [Drewes, 2005]. The reader already familiar with trees, regular tree grammars and top–down tree transducers may wish to proceed directly to the next chapter.

4.1 Trees

To illustrate the formal definitions, we will use a tree made out of sticks and balls as a cognitive model. This analogy will however prove too clumsy in more complex argumentation, so the reader is advised to use it only to decrypt the formal definitions (and not per se).

Imagine a little plastic ball with a label, a hole drilled into its top and a couple of sticks protruding from its lower half (zero or more of these sticks). If you had a number of these balls, then you could build a bigger structure by inserting a stick coming out of one ball into the hole at the top of another. If the structure that you built had no loose stick ends, then you would be allowed to call your creation a tree. To build a proper tree you would of course need at least one ball with no protruding sticks at all, because there is a rule which says that you are not allowed to remove a stick that is already attached to a ball.

In the formal definition below, a labelled ball is called a *symbol* and the number of sticks protruding from the lower half is the *rank* of the symbol, a natural number. A *signature* would be a set of these balls, where you only had a finite number of different types of balls (with regards to labels and number of protruding sticks), but there would be an infinite number of each type that you *did* have. If you are given a signature Σ, then *the trees over* Σ refers to the set of all trees you could build using only labelled balls from Σ. A *tree language* is a subset of the trees over Σ. In the formal definition, we shall write $[n]$, where n is a natural number, to denote the set $\{1, \ldots, n\}$.

Definition 4.1 (Signature and rank) A signature $\Sigma = \bigcup_{k \in [n]} \Sigma^{(k)}$, $n \in \mathbb{N}$, is a finite set of symbols, partitioned into pairwise disjoint subsets $\Sigma^{(k)}, k \in [n]$. The symbols in $\Sigma^{(k)}$ are said to have *rank* k. When the rank of a symbol is important, we will add it as a superscript, i.e. if $f \in \Sigma^{(k)}$, we may write $f^{(k)}$.

Definition 4.2 (Trees and tree languages) The set T_Σ of trees over Σ is defined inductively:

1. $\Sigma^{(0)} \subseteq \mathrm{T}_\Sigma$, and
2. for $k \geq 1$, $f \in \Sigma^{(k)}$, and $t_1, \ldots, t_k \in \mathrm{T}_\Sigma$, the tree $f[t_1 \ldots t_k]$ belongs to T_Σ.

A subset of T_Σ is called a *tree language*.

Suppose that in addition to your original set of balls and sticks, which are all green, your friend gives you a set of ready-built trees made out of balls and sticks that were all red (let us call this set S). The set of trees *indexed by* S contains all the different trees you could build with your green balls and sticks, with the option of including some of the trees your friend built (but you cannot break your friends trees for components). Again, formally:

Definition 4.3 (Indexed trees) Let Σ be a signature and let S be a tree language. Then the set of trees over Σ indexed by S, denoted by $\mathrm{T}_\Sigma(S)$, is defined inductively as follows;

1. $S \cup \Sigma^{(0)} \subseteq \mathrm{T}_\Sigma(S)$ and
2. for $k \geq 1$, $f \in \Sigma^{(k)}$ and $t_1, \ldots, t_k \in \mathrm{T}_\Sigma(S)$, the tree $f[t_1 \ldots t_k]$ belongs to $\mathrm{T}_\Sigma(S)$.

Note that $\mathrm{T}_\Sigma(\emptyset) = \mathrm{T}_\Sigma$.

The *size* of a tree, in ball terminology, is simply the number of balls used to build it. If you took hold of the top-most ball (which we call the *root*) and held the whole thing up in front of you, then at the bottom of the tree you would find the balls with no protruding sticks, and if you read their labels in sequence starting from the left-most ball and reading them one by one, you would have the *yield* of the tree. The *height*, finally, is the longest number of sticks you could pass if you started at the root and were only allowed to move downwards. These definitions be formalised as in the following:

Definition 4.4 (Yield, height, and size) Let t be at tree in T_Σ. The yield of t, written $\mathit{yield}(t)$ is defined as follows.

1. If $t = f$, for some $f \in \Sigma^{(0)}$, then $\mathit{yield}(t) = f$.
2. Otherwise $t = f\,[t_1, \ldots, t_k]$ for some $k > 0$, and $\mathit{yield}(t)$ is the concatenation of $\mathit{yield}(t_1), \ldots, \mathit{yield}(t_k)$.

The height of t, written $\mathit{height}(t)$, is defined as follows.

1. If $t = f$, for some $f \in \Sigma^{(0)}$, then $\mathit{height}(t) = 0$.
2. Otherwise $t = f\,[t_1, \ldots, t_k]$ for some $k > 0$, and $\mathit{height}(t)$ is equal to $1 + \mathit{max}(\mathit{height}(t_1), \ldots, \mathit{height}(t_k))$.

The size of $t = f[t_1, \ldots, t_k]$, written $|t|$, is $1 + \sum_{i=1}^{k} |t_i|$.

The set of subtrees of a (ball) tree consists of the tree itself, together with the set of all other trees that could be built by pulling out one stick from the top of any single ball in the original tree, and then discarding the structure that is not a tree.

Definition 4.5 (Subtree) The set $\mathit{subtrees}(t)$ of a tree $t = f[t_1, \ldots, t_k]$ is recursively defined as $\{f[t_1, \ldots, t_k]\} \cup \bigcup_{i=0}^{k} \mathit{subtrees}(t_i)$. A tree s is a subtree of t if $s \in \mathit{subtrees}(t)$.

Suppose that we are building a stick-and-ball tree together with a friend; we are constructing the topmost part of the tree, while our friend is working on some of the lower subtrees. To coordinate our work we add yet another set of balls, and to distinguish these from those that we already have, we require that all new

balls are white (in contrast to the old ones, which are never white). We then use the white balls as place markers to indicate to our friend where in our tree she should attach her trees. We will henceforth think of the white balls as *variables*, and when the tree that we are constructing changes because at the place of a variable a new subtree is attached, then we shall say that the new subtree has been *substituted* into the tree. The reader is advised to be attentive, because the formal definition allows that many variables are substituted simultaneously:

Definition 4.6 (Substitution) Let $X = \{x_1, x_2, \ldots\}$ be a set of special symbols, so-called variables, all of rank zero, that is disjoint with every other signature in this paper. If $t \in \mathrm{T}_\Sigma(X)$ for some arbitrary signature Σ, then we denote by $t[\![t_1, \ldots, t_k]\!]$ the tree that results when each occurrence of x_i in t is replaced by t_i, $i \in [k]$. When we only wish to talk about a subset $\{x_1, \ldots, x_k\}, k \in \mathbb{N}^+$ of X, we refer to the subset as X_k.

4.2 Regular tree grammars

We now leave the ball-and-stick trees behind us, and turn to the operations that can be performed on formal trees. This paper centres around the idea that trees can be used to represent tunes and their constituents. There are an infinite number of combinations of meters, tunes, scales, pitches etc. and we could not possibly find place to write down the corresponding trees explicitly. Instead, we are going to work with finite representations of these infinite sets, and in doing this the so-called *regular tree grammar* will prove useful.

A regular tree grammar is a formal device g consisting of four parts:

1. a signature of symbols of rank zero called nonterminals. The nonterminals will that act as gs memory, and will occur when g is in the process of generating a tree, but not in the final tree;
2. a signature of symbols (disjoint with the nonterminals) called terminals. These are the building blocks for the trees produced by g;
3. a set of rules – instructions that tell g how to build a tree; and
4. a start symbol, the initial symbol in each generation.

A more formal definition of the same device follows next.

Definition 4.7 (Regular tree grammar) A regular tree grammar (rtg) is a quadruple $g = (N, \Sigma, P, S)$ consisting of

1. a signature N of nonterminals of rank 0;
2. a signature Σ of terminals, with $N \cap \Sigma = \emptyset$;
3. a finite set of rules of the form $A \to t$, where $A \in N$ and $t \in T_{N \cup \Sigma}$; and
4. an initial nonterminal S.

A derivation of a regular tree grammar $g = (N, \Sigma, P, S)$ begins with a tree of size one, whose single node is labelled S. The process that follows is iterative: In each step, a rule $r = A \to t \in R$, such that A occurs in the current tree, is selected and t (the right-hand side of r) is substituted for an occurrence of A (the left-hand side). This is repeated until there are no more nonterminals left in the tree.

Definition 4.8 (Derivation and derivation step) Let $g = (N, \Sigma, P, S)$ be a regular tree grammar, and $s,\ s' \in \mathrm{T}_{\Sigma \cup N}$. We say that there is a *derivation step* from s to s' and write $s \Rightarrow_g s'$ (or simply $s \Rightarrow s'$ if g follows from the context) if and only if

1. $s = t_0[\![A]\!]$ for a tree t_0 containing exactly on occurrence of x_1,
2. there is a rule $A \to t \in P$, and
3. $s' = t_0[\![t]\!]$.

We denote the transitive reflexive closure of $\Rightarrow_g$ by $\overset{*}{\Rightarrow}_g$ and say that there is a derivation from s to s' if $s \overset{*}{\Rightarrow}_g s'$. The *generated language* of a regular tree grammar $g = (N, \Sigma, P, S)$, written $L(g)$, is the set of trees $\{t \in \mathrm{T}_\Sigma \mid S \overset{*}{\Rightarrow}_g t\}$.

Example 4.9 Suppose that we want to generate the measures of a tune in time signature three halves. This requires that the total note value of each measure corresponds to a whole dot, although possibly divided over a number of actual notes. We now construct the regular tree grammar $g_{\text{div}} = (N, \Sigma, P,$ (𝅝.) $)$, that takes a whole dot note and divides it in various ways into halves, quarters etc. Plain note symbols are used as terminals, while note symbols enclosed in circles stand for nonterminals.

Note	𝅝.	𝅝	𝅗𝅥.	𝅗𝅥	♩.	♩	♪.	♪
Duration	$1 + \frac{1}{2}$	1	$\frac{1}{2} + \frac{1}{4}$	$\frac{1}{2}$	$\frac{1}{4} + \frac{1}{8}$	$\frac{1}{4}$	$\frac{1}{8} + \frac{1}{16}$	$\frac{1}{8}$

Table 1: When a dot is added to a note, its length is extended by 50%. For example, 𝅝 is the reference note, and 𝅝. lasts one and a half time as long.

Table 1 lists the duration of different notes, including 𝅝., which has length three halves and is consequently used as start symbol. The terminal symbols $\circ^{(2)}$ and $\circ^{(3)}$ can be thought of as a binary and a ternary concatenation operator, respectively, whereas the remaining components of g_{div} are defined as follows.

$N = \{$ ⟨𝅝.⟩, ⟨𝅗𝅥.⟩, ⟨𝅗𝅥⟩, ⟨♩.⟩, ⟨♩⟩, ⟨♪.⟩, ⟨♪⟩ $\}$

$\Sigma = \{$ $\circ^{(2)}, \circ^{(3)}$,
𝅝.$^{(0)}$, 𝅗𝅥.$^{(0)}$, 𝅗𝅥$^{(0)}$,
♩.$^{(0)}$, ♩$^{(0)}$, ♪.$^{(0)}$, ♪$^{(0)}$ $\}$

$P = \{$
⟨𝅝.⟩ → 𝅝., ⟨𝅝.⟩ → ∘[⟨𝅗𝅥.⟩, ⟨𝅗𝅥.⟩], ⟨𝅝.⟩ → ∘[⟨𝅗𝅥⟩, ⟨𝅗𝅥⟩, ⟨𝅗𝅥⟩],
⟨𝅗𝅥.⟩ → 𝅗𝅥., ⟨𝅗𝅥.⟩ → ∘[⟨♩.⟩, ⟨♩.⟩], ⟨𝅗𝅥.⟩ → ∘[⟨♩⟩, ⟨♩⟩, ⟨♩⟩],
⟨𝅗𝅥⟩ → 𝅗𝅥, ⟨𝅗𝅥⟩ → ∘[⟨♩⟩, ⟨♩⟩],
⟨♩.⟩ → ♩. ⟨♩.⟩ → ∘[⟨♪.⟩, ⟨♪.⟩], ⟨♩.⟩ → ∘[⟨♪⟩ , ⟨♪⟩ , ⟨♪⟩],
⟨♩⟩ → ♩, ⟨♩⟩ → ∘[⟨♪⟩ , ⟨♪⟩],
⟨♪.⟩ → ♪.,
⟨♪⟩ → ♪ $\}$

One possible derivation of g_{div} is shown in Figure 9, but there are, of course, many others. By inspection of P we can tell that g_{div} can derive more than sixty distinct trees, each corresponding to a subdivision of the note value three halves, and where the shortest notes would be equal or longer than an eighth. The reader is encouraged to follow a number of derivations of g_{div} through.

Sixty trees are not that many – certainly not anywhere near infinity. Yet, neither is *any* finite number. We originally argued that the need for tree grammars stems from the existence of infinite sets of trees, but by now it should be clear that they become useful as soon as the sets that we want to generate are sufficiently large. It would perhaps be silly to claim that this is the case for $L(g_{\text{div}})$, but restrictions such as 'there is not space enough to define more trees than there are atoms in this universe' can reasonably be imposed (and for this particular restriction, it is clear that a set as large as the number of atoms in the universe is sufficiently large for tree grammars to be useful).

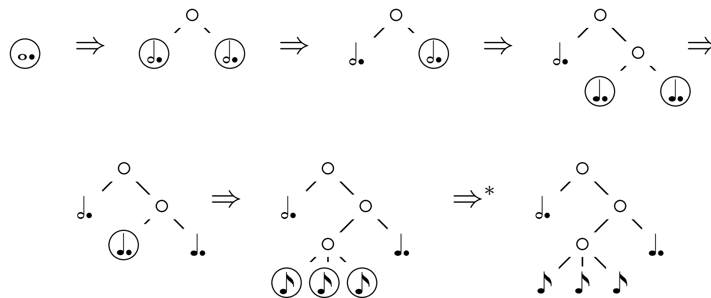


Figure 9: The regular tree grammar g_{div} subdivides a note value of three halves into a rhythmic pattern.

4.3 Tree transducers

The regular tree grammar is a device for *generating* trees. What it outputs is completely determined by the definition of the tree grammar, and nothing 'external' can affect the result. In contrast, our approach to generating music takes after an assembly line. First, we generate a tree t_m to represent the meter of our tune, and this can be done with a regular tree grammar. In the following steps we augment the meter with melody and accompaniment. This cannot be done using a regular tree grammar, for the obvious reason that a regular tree grammar could not add anything to t_m; only produce completely new trees. What we need is a formal device which can *transform* trees, and we find this in the *top–down tree transducer*.

The top–down tree transducer (or td transducer, for short) can be compared to a regular tree grammar modified to consume an input tree, while simultaneously producing an output tree. A td transducer consists of five components, the first of these are an *input signature* and *an output signature*, the building blocks for the input trees and output trees respectively. The third component is a signature of so-called *states*, all of rank one. The states correspond to the nonterminals of an rtg and act as the td transducers memory. Then follows a set of rules, which are similar to, but differ from, the rules of an rtg. The fifth and last component is the *initial state*.

Recall that the rules of an rtg have the form $A \to t$, where A is a nonterminal and t is a tree over the (output) signature. A rule of this type is applicable if the following condition is met: the nonterminal A occurs in the derivation tree, and there is a rule $A \to t$. If a rule $A \to t$ is applicable in a derivation tree, then we may replace an occurrence of A in the derivation tree by t.

The rules of a tree transducer, on the other hand, have the form $q[f[x_1, \ldots, x_k]] \to t$, where q is a state, f is an input symbol of rank k, and $x_1, \ldots, x_k$ are variables in X_k. The tree t also has a particular form – it is a tree over the output signature, indexed by states, and below each state is a variable in X_k. A rule belonging to a td transducer is applicable if the following, more restrictive, condition is met: at some place in the tree that is being transformed, there is a state q, and directly below it, the subtree $f[t_1, \ldots, t_k]$, and there must also be a rule $q[f[t_1, \ldots, t_k]] \to t$. If a rule $q[f[x_1, \ldots, x_k]] \to t$ is applicable, we may replace an occurrence of a subtree $q[f[t_1, \ldots, t_k]]$ in the tree that is being transformed by $t[\![t_1, \ldots, t_k]\!]$.

Definition 4.10 (Top-down tree transducer) A top–down tree transducer is a quintuple $td = (\Sigma, \Sigma', Q, R, q_0)$, where

1. Σ is an input signature,
2. Σ' is an output signature,
3. Q is a signature of states of rank 1, such that $Q \cap (\Sigma \cup \Sigma') = \emptyset$,

4. R is a finite set of rules, and
5. $q_0 \in Q$ is the initial state.

Every rule in R has the form

$$q[f[x_1, \dots, x_k]] \to t,$$

where $k \in \mathbb{N}$, $q \in Q$, $f \in \Sigma^{(k)}$ and $t \in \mathrm{T}_{\Sigma'}(Q(X_k))$. To improve legibility, we henceforth write $q\, f[t_q, \dots, t_k]$, rather then $q[f[t_q, \dots, t_k]]$, when $q \in Q$.

For trees $s, s' \in \mathrm{T}_{\Sigma'}(Q(\mathrm{T}_\Sigma))$, there is a *transduction step* $s \mapsto_{td} s'$ from s to s' if

1. $s = s_0[\![q\, f[t_1, \dots, t_k]]\!]$ for some tree s_0 containing x_1 exactly once, $q \in Q$, and $f \in \Sigma^{(k)}$,
2. there is a rule $q\, f[x_1, \dots, x_k] \to t \in R$, and
3. $s' = s_0[\![t[\![t_1, \dots, t_k]\!]]\!]$.

We denote the transitive closure of $\mapsto_{td}$ by $\overset{*}{\mapsto}_{td}$ and say that there is a transduction from s to s' if $s \overset{*}{\mapsto}_{td} s'$. Finally, $td(s) = \{s' \in \mathrm{T}_\Sigma \mid s \overset{*}{\mapsto}_{td} s'\}$.

To convey the fairly simple mechanism of a td transducer, we include an example.

Example 4.11 When two voices are played simultaneously, it is often the case that when one voice is very active, the other voice rests, and vice versa. This implies a need for synchronisation of attributes (such as tempo, volume, melodic arc et cetera) between the voices. One way to accomplish this is to first generate a single voice, which is then duplicated and the attributes of the copy adjusted (or rather, inverted). This is a typical task for which the td transducer is a suitable device.

In this example we shall only consider one attribute, and that is the tempo. Suppose that the original voice is given as a tree $t = \mathsf{voice}[t']$, such that the symbol voice does not occur in t'. The yield of t is a sequence of paces – which can be slow, medium or fast – whereas that the internal nodes of t only hold structural information. In the following, we give the components of the td transducer $td = (\Sigma, \Sigma', Q, R, q_p)$.

$$\Sigma = \{ \quad \mathsf{voice}^{(1)}, \circ^{(2)}, s^{(0)}, m^{(0)}, f^{(0)} \quad \}$$
$$\Sigma' = \Sigma \cup \{ \mathsf{voices}^{(2)} \}$$
$$Q = \{ \quad q_a, q_p \quad \}$$

$R = \{$	$q_p\,\mathsf{voice}\,x_1 \to \mathsf{voices}[q_p\,x_1,\; q_a\,x_1]$	*Duplicate the original voice and modify the right-most copy.*
	$q_p\;\circ[x_1, x_2] \to \circ[q_p\,x_1, q_p\,x_2]$	*Rules that pass over the first voice, which is left unaltered.*
	$q_p\,s \to s$	
	$q_p\,m \to m$	
	$q_p\,f \to f$	
	$q_a\,\mathsf{voice}[x_1] \to \mathsf{voice}[q_a\,x_1]$	*This rules is a dummy – due to the restriction on the input trees, it will never be applicable.*
	$q_a\;\circ[x_1, x_2] \to \circ[q_a\,x_1, q_a\,x_2]$	*When working on the second voice, first proceed towards the leaves.*
	$q_a\,s \to f$	*Second voice moves fast, when first voice moves slow.*
	$q_a\,m \to m$	*Second voice moves at medium pace, if first voice does.*
	$q_a\,f \to s$	*Second voice moves slow, when first voice moves fast.* $\}$

If we want to cover every combination of a state and an input symbol, the set of rules cannot have less then $|\Sigma| \times |Q|$ entries. The set R – dummy rule included – has indeed exactly ten rules. A transduction of *td* may proceed as in Figure 10. It is suggested that the reader verify that every transduction step in the figure is made in accordance with R (this also holds for those transduction steps that are not explicitly shown). As the reader may have guessed, a td transducer with many states and/or a large input signature also needs an extensive transition table to cope with all combinations of states and input symbols. This basic type of td transducer that covers *every* possibility is called a *total* td transducer. However, there are many transductions that are better described by a so-called *partial* td transducer – a td transducer whose rules only cover a subset of all possible left-hand sides. If, at some point in a transduction of a partial td transducer, an undefined combination of state and input symbol arises, then the transduction is aborted and the td transducer produces no output. A partial definition coupled with so-called nondeterminism can at times be very useful, and we will return to this subject towards the end of this section.

Many of the td transducers that constitute Willow only make minor changes to their input trees – most input symbols are left untouched, or rather, are represented by the same symbol at the corresponding position in the output tree. The work of adding all the rules that just transfer a symbol in the input tree to the output tree seemed somewhat pointless, so we introduced the idea of an *ignorant* td transducer. Just as a partial td transducer, an ignorant td transducer has an

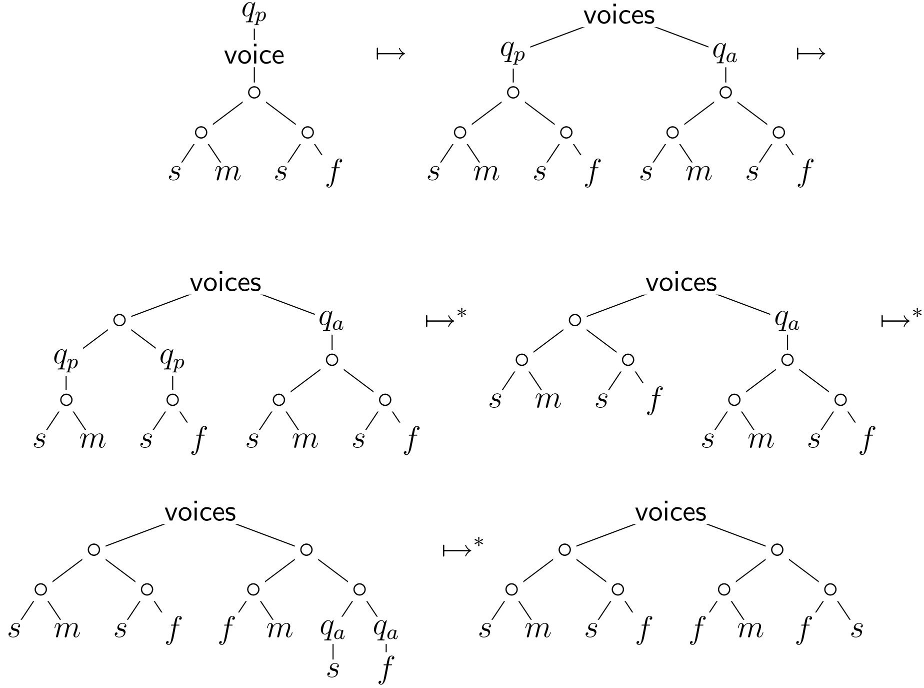


Figure 10: The td transducer *td* copies a voice and then inverts the tempo of the duplicate.

incomplete set of rules. The difference is that when, in a transduction of an ignorant td transducer, an undefined combination of a state q and an input symbol of rank k arises, the symbol is simply transferred to the output tree and the td transducer continues working (in state q) on each of the k subtrees.

Each type of td transducer – total, partial and ignorant – has certain advantages and disadvantages, and for this reason they are all represented in the sequence of devices that constitutes Willow. Although this sequence may vary slightly in length, depending on the configuration of the system, it usually contains some twelve to fifteen td transducers. The reader may wonder if they are all really necessary – perhaps there is a clever way to combine two or more of them into one? In some cases a composition is possible, but to recognise these situations, we need to know whether the td transducers in question are *linear*, *deleting* and/or *deterministic*. We shall have more to say on the topic of composition of td transducers in Section 6.

Definition 4.12 (Linear, copying and deleting) Let $td = (\Sigma, \Sigma', Q, R, q_0)$ be a td transducer. We call *td* a *linear* td transducer if, in every rule, each variable occurs at most once in its right-hand side. A td transducer that is not linear is *copying*. We say that *td* is a *deleting* td transducer if there is a rule r in

R whose left-hand side contains a variable that does not occur in the right-hand side.

Another property of td transducers that we must pay attention to is determinism. Simply put, if td has two rules with identical left hand sides, then td is a *nondeterministic* td transducer, as opposed to a *deterministic* td transducer (in which case all left hand sides are distinct).

Definition 4.13 (Deterministic) A td transducer $td = (\Sigma, \Sigma', Q, R, q_0)$ is *deterministic* if distinct rules have distinct left-hand sides.

To illustrate how nondeterminism can be applied to certain types of problems, we include an example. This also concludes the section on tree grammars and tree transducers.

Example 4.14 Suppose that we have a tree t, whose yield is a sequence of quarter notes, and that we wish to mark the second-to-last and last notes with a d (for dominant) and a t (for tonic), respectively. After some consideration, we realize that this cannot be accomplished by a deterministic td transducer: There are decisions that must made early in the transduction, but in accordance with information that can only be obtained towards the end of the transduction. A nondeterministic partial td transducer, on the other hand, can guess by making a nondeterministic choice. If the guess is erroneous, then the transduction will fail to terminate and no output tree will be produced; but if the guess is correct, the objective is accomplished and the last two notes of the output tree turned into a cadence. We construct a nondeterministic partial td transducer $td = (\Sigma, \Sigma', Q, R, q_s)$ as follows.

$\Sigma = \{ \ \circ^{(2)}, ♩^{(0)} \ \}$

$\Sigma' = \Sigma \cup \{ \ d^{(1)}, t^{(1)} \ \}$

$Q = \{ \ q_s, q_d, q_t, q_p \ \}$

$R = \{$

Rule	Comment
$q_s \circ[x_1, x_2] \rightarrow \circ[q_p \, x_1, q_s \, x_2]$	*Search for the parent of the last note.*
$q_s \circ[x_1, x_2] \rightarrow \circ[q_d \, x_1, q_t \, x_2]$	*Guess that we are at the parent of the last note.*
$q_d \circ[x_1, x_2] \rightarrow \circ[q_p \, x_1, q_d \, x_2]$	*Stay to the right while looking for the second to last note.*
q_d ♩ $\rightarrow d[$♩$]$	*Place the d marker for 'dominant'.*
q_t ♩ $\rightarrow t[$♩$]$	*If state q_t encounters anything but a leaf, the transduction fails.*
$q_p \circ[x_1, x_2] \rightarrow \circ[q_p \, x_1, q_p \, x_2]$	*Transfer the rest of the tree to the output.*
q_p ♩ $\rightarrow$ ♩	

$\}$

One possible transduction is shown in Figure 11. At the very first step, *td* guesses that it has found the parent of the last note. Clearly, this is not the case. After a number of transductions steps the rightmost tree of the figure has been reached. At this point the transduction must abort, because there are no rules with left-hand side $q_t \circ [x_1, x_2]$.

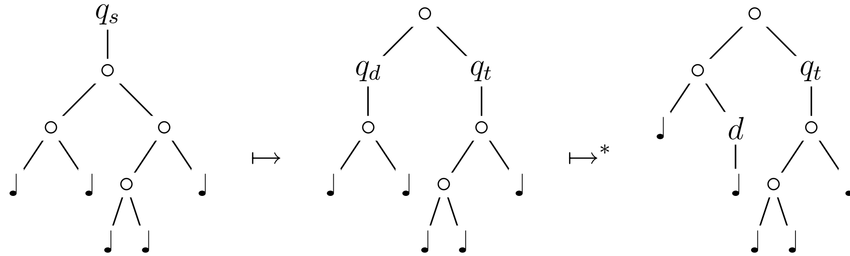


Figure 11: Because of an incorrect earlier guess, the nondeterministic td transducer fails to complete the transduction.

An alternative transduction is shown in Figure 12. Here, the td transducer waits a number of steps before guessing nondeterministically that it has found the sought parent. This time it has guessed correctly, so the transduction succeeds in turning the last two notes into a cadence and terminating.

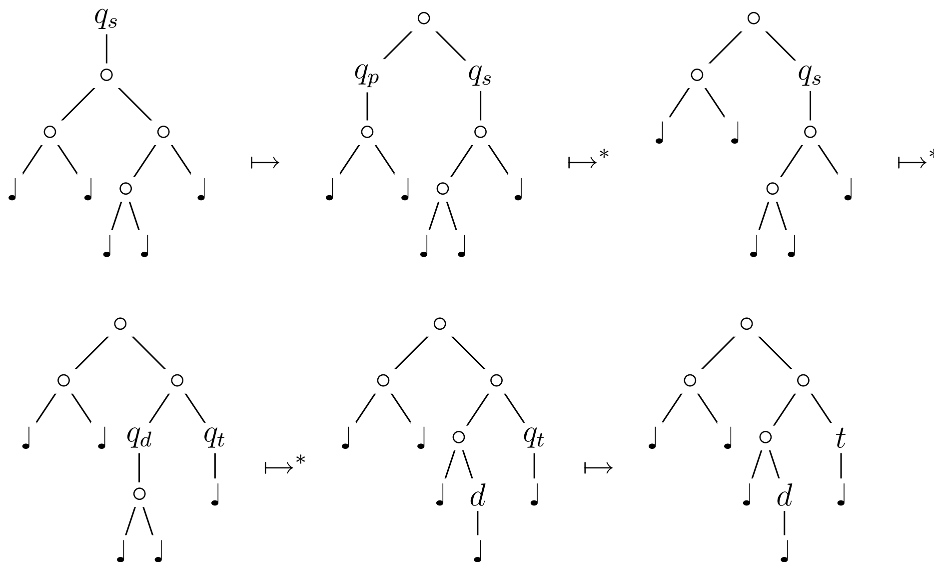


Figure 12: After a series of correct guesses the nondeterministic td transducer terminates.

5 Willow's assembly line

The music generating device Willow is implemented in the TREEBAG environment, and consists of some fifteen components, mainly a regular tree grammar, a number of td transducers, and a score display. For technical reasons, a so-called free term algebra is added. The generative process starts when the regular tree grammar produces a tree, which is then processed by the td transducers before it is finally written to a file by the display. It is not completely correct to talk about *a* tree that changes through the process, when it really is a sequence of distinct trees. However, as input tree and output tree of most td transducers differ only slightly, it is easier to think of the sequence of trees as a single tree that is being developed. Figure 5 shows the TREEBAG graphical user interface with Willow loaded. If the icon for the Score display is clicked, a rectangular menu becomes visible. When the command 'Print mup file' of this menu is selected, the tree representation of the generated tune (which can be manipulated by reordering and/or replacing td transducers) is converted to mup format and written to a file.

Three principles have guided the implementation of Willow. First, the system should be loosely coupled: Each td transducer is designed to add a specific attribute, affecting the work of previous transducers as little as possible. This allows the user to relatively freely exchange and re-order the td transducers. Secondly, every attribute added by a td transducer should be firmly rooted in music theory. It is hoped that this requirement makes the generation process

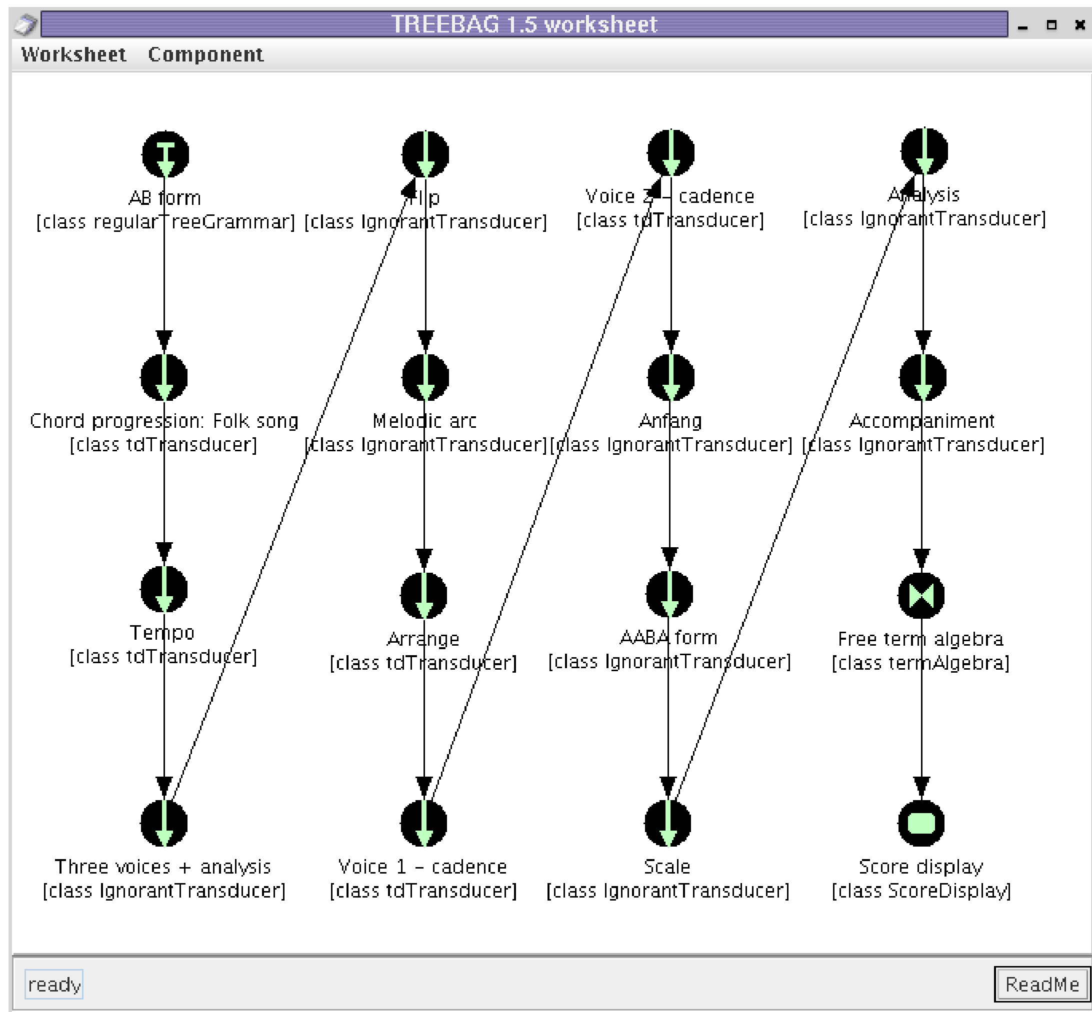


Figure 13: Willow loaded into TREEBAG.

straightforward, not only to the formal language theorist, but also to the music theoretician. Thirdly, each component should be kept as simple as possible. By simple we mean two things; the component should be the weakest possible device sufficient, and the definitions of the devices should be kept as short and clear as possible. The principle of simplicity has occasionally been given precedence over the principle of fidelity, because correctness cannot be assured without it: As the size of the definitions grow, so does the number of errors. More importantly; no one will be very surprised if a basic tune on AABA form can be generated by something almost as powerful as a Turing machine. The system simply looses its scientific value when it becomes too complex.

5.1 Treebag

Since Willow has been implemented in the Treebag environment, the resources offered by Treebag, in terms of available devices and control mechanisms, have guided the development towards its present state, and are likely to maintain their influence also in the future. Therefore, a presentation of Willow could not be complete without an introduction to Treebag.

Treebag was designed and implemented by Frank Drewes. He describes his system as follows:

> Treebag is a system that allows to generate, transform, and display objects of various types, where generation and transformation is done using tree grammars and tree transducers. The basic principle is that tree grammars produce trees over symbols that are interpreted by appropriate algebras as operations on some domain. Thus, every tree is viewed as an expression that denotes one of the objects of interest. These objects can be visualised using appropriate displays.

There are several types of component classes available for generation, transformation, interpretation and display. As the implementation of Treebag is object oriented, there is a straightforward way of writing new classes. Furthermore, the components themselves can be loaded and un-loaded independently, their interactions being controlled by the user: The system has a graphical user interface in which each component is represented by an icon. The user can connect two components with an arrow, thus directing data flow. For a technical report on Treebag, see [Drewes, 1998]. Newer information can be found in [Drewes, 2005].

Example 5.1 The form known as AABA form was very popular during the first half of the 20th century, and remains the most common form in jazz standards. It is divided into four sections, first two identical verse sections A and A, then a contrasting section B (the bridge) and finally a concluding A. To see how a rhythmic passage of AABA form can be generated by Willow, consider Figure 5.1. The Treebag environment is shown with components loaded that belong to the classes regular tree grammar, ignorant transducer, term algebra and score display. The regular tree grammar Ab form generates a tree structure to represent a sequence of measures of two four time, divided into two equal length phrases. The ignorant transducer Aaba form rearranges its input tree by adding a copy of the first phrase at the beginning and end of the rhythmic passage. The term algebra called free term algebra is needed because Treebag displays only accept input from algebras. However, in this case the algebra simply returns as its output tree the input tree, so that we may disregard this component in the following.

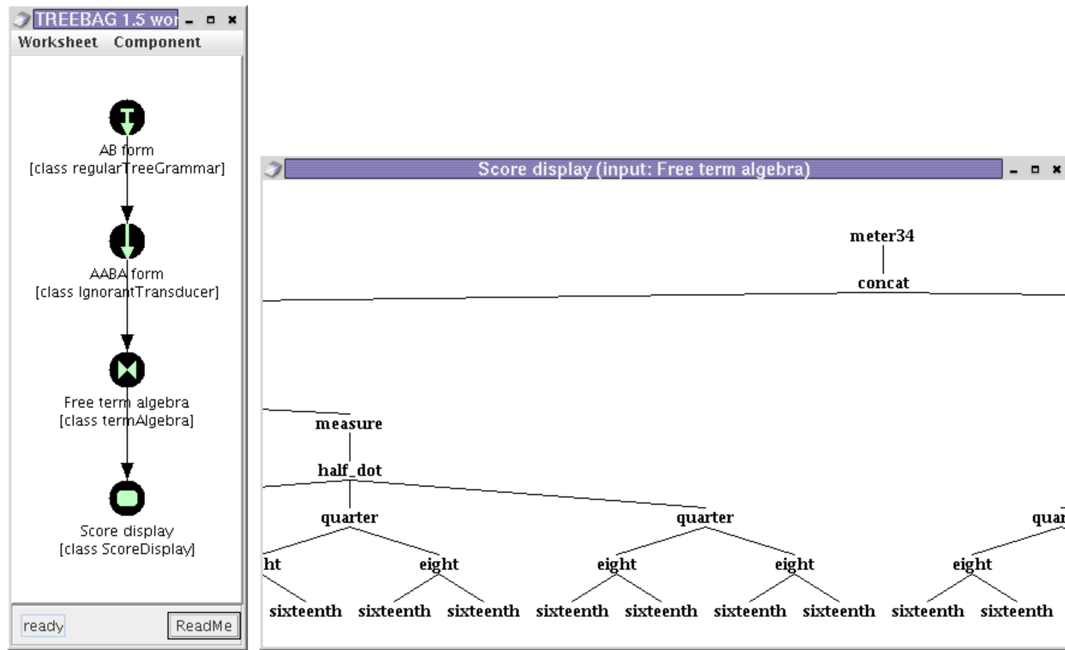


Figure 14: TREEBAG with components from classes Regular tree grammar, Ignorant transducer, Free term algebra and Score display loaded.

5.2 Rhythm and meter

The meter of a tune is not unlike a skeleton, as it sketches out the rudimentary features of the tune at the same time as it distributes place markers over time for tones and cadences. Also, the meter implicates a boundary of sorts on the rhythm. For all these reasons, generating a meter seems a reasonable first step in generating a whole tune. To do this, we represent the meter as a tree, where the label of the root tells us what time is used. Each measure is represented as a subtree, and the note values at the leaves of such a subtree comply, when aggregated, with the time declared in the root label. A meter tree of this type is illustrated in Figure 15. The reader may question why we need the root label at all, when then the total note value of a measure can be deduced from the leaves. The reason for this is that although two different times, e.g. two two and four four, have measures of equal note value, they are traditionally accented differently. A time signature is more than a note value.

We have already met the component that generates the meter – it is the rtg AB form presented in Example 5.1. At present this component is restricted to generating meters in two two, two four, three four and four four time. Also, the phrases generated will have two, three or four measures each. The types of meters generated could very well be increased, but it does not seem that further insight will be gained by doing this. The length of the phrases, on the other

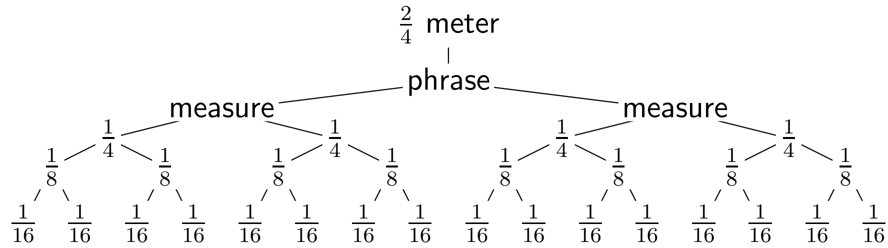


Figure 15: A meter is represented as a tree, in which the root label declares the time signature and the sum of the note values in each measure complies.

hand, cannot be increased arbitrarily much, because the length of the phrases is an important factor in the descriptional complexity of Willow.

Chord transitions and tempo changes accent the notes they occur on, and thus influence the rhythm. The rewrite rules of the td transducers that assign chord progression and tempo are weighted in such a way that allocations are most likely to be made at the level of measures, then at the level of half measures, and so forth. It follows that changes are most likely to take place on the first note of a measure, then the first of the second half of the measure, et cetera, which is in accordance with the accent patterns of Figure 8.

5.3 Chord progression

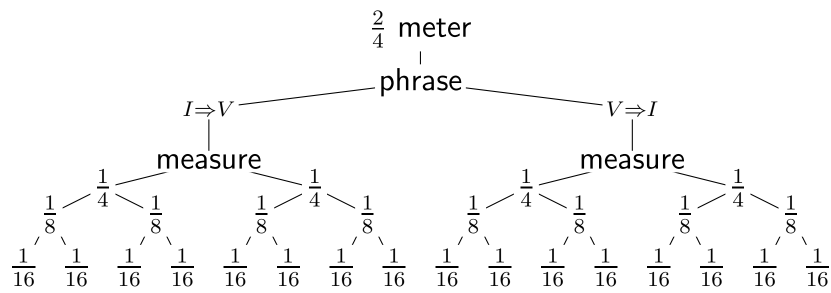
Consider the chord progression of popular music in Figure 7 on page 10. What we would like to do is to assign chords to the lower nodes of our meter tree, in such a way that when the assigned chords are read from left to right they follow the chord progression. We do not want all chords to be assigned to nodes at the same level, because this would correspond to a tune where the chord changes every n-th whole note, and that is not desired behaviour. With flexibility in mind we define the td transducer Chord, shown in Figure 16 on the following page (the vertical dots indicate that a number of lines have been omitted).

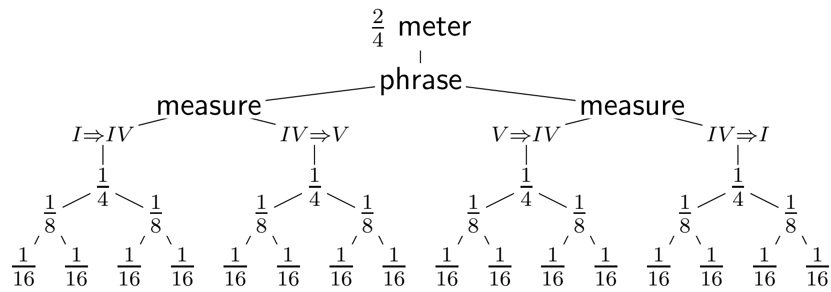
How Chord transforms an input tree representing a meter of two four time is shown in Figure 17 on page 32, where the first two steps have been omitted in order to fit the transformation on one page, and the nonterminals are written as, e.g. $I \Rightarrow V$ rather then I_V.

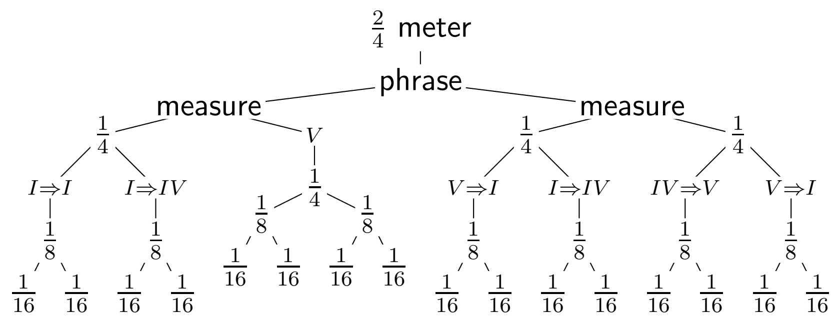
The states of Chord correspond to chord transitions. Given that the pop progression in question uses three chords, we would expect there to be nine possible transitions. However, Chord has nineteen states, and this requires an explanation. The state P means 'pass' and simply replicates the input tree to output. In the first of the tree terminating rules below, the state 'transition from chord I to

```
generators.tdTransducer("Chord progression: Pop"):
(
  { #include(labels) },
  { #include(labels) },
  { #include(nonterminals) },
  {
    I_I[phrase[x1,x2,x3]]         -> phrase[I_I[x1], I_I[x2], I_I[x3]],
    I_I[phrase[x1,x2,x3]]         -> phrase[I_I[x1], I_IV[x2], IV_I[x3]],
    I_I[phrase[x1,x2,x3]]         -> phrase[I_I[x1], I_V[x2], V_I[x3]],
    I_I[phrase[x1,x2,x3]]         -> phrase[I_IV[x1], IV_I[x2], I_I[x3]],
    I_I[phrase[x1,x2,x3]]         -> phrase[I_IV[x1], IV_IV[x2], IV_I[x3]],
    I_I[phrase[x1,x2,x3]]         -> phrase[I_IV[x1], IV_V[x2], V_I[x3]],
    I_I[phrase[x1,x2,x3]]         -> phrase[I_V[x1], V_I[x2], I_I[x3]],
    I_I[phrase[x1,x2,x3]]         -> phrase[I_V[x1], V_IV[x2], IV_I[x3]],
    I_I[phrase[x1,x2,x3]]         -> phrase[I_V[x1], V_V[x2], V_I[x3]],
    I_TI[phrase[x1,x2,x3]]        -> phrase[I_I[x1], I_I[x2], I_TI[x3]],
    I_TI[phrase[x1,x2,x3]]        -> phrase[I_IV[x1], IV_I[x2], I_TI[x3]],
    I_TI[phrase[x1,x2,x3]]        -> phrase[I_V[x1], V_I[x2], I_TI[x3]],

    I_I[phrase[x1,x2,x3,x4]]      -> phrase[I_I[x1], I_I[x2], I_I[x3], I_I[x4]],
    I_I[phrase[x1,x2,x3,x4]]      -> phrase[I_I[x1], I_I[x2], I_IV[x3], IV_I[x4]],
    I_I[phrase[x1,x2,x3,x4]]      -> phrase[I_I[x1], I_I[x2], I_V[x3], V_I[x4]],
    I_I[phrase[x1,x2,x3,x4]]      -> phrase[I_I[x1], I_IV[x2], IV_I[x3], I_I[x4]],
    I_I[phrase[x1,x2,x3,x4]]      -> phrase[I_I[x1], I_IV[x2], IV_IV[x3], IV_I[x4]],
    I_I[phrase[x1,x2,x3,x4]]      -> phrase[I_I[x1], I_IV[x2], IV_V[x3], V_I[x4]],
    I_I[phrase[x1,x2,x3,x4]]      -> phrase[I_I[x1], I_V[x2], V_I[x3], I_I[x4]],
    I_I[phrase[x1,x2,x3,x4]]      -> phrase[I_I[x1], I_V[x2], V_IV[x3], IV_I[x4]],
    I_I[phrase[x1,x2,x3,x4]]      -> phrase[I_I[x1], I_V[x2], V_V[x3], V_I[x4]],
    I_I[phrase[x1,x2,x3,x4]]      -> phrase[I_IV[x1], IV_I[x2], I_I[x3], I_I[x4]],
    I_I[phrase[x1,x2,x3,x4]]      -> phrase[I_IV[x1], IV_I[x2], I_IV[x3], IV_I[x4]],
    I_I[phrase[x1,x2,x3,x4]]      -> phrase[I_IV[x1], IV_I[x2], I_V[x3], V_I[x4]],
    I_I[phrase[x1,x2,x3,x4]]      -> phrase[I_IV[x1], IV_IV[x2], IV_I[x3], I_I[x4]],
    I_I[phrase[x1,x2,x3,x4]]      -> phrase[I_IV[x1], IV_IV[x2], IV_IV[x3], IV_I[x4]],
    I_I[phrase[x1,x2,x3,x4]]      -> phrase[I_IV[x1], IV_IV[x2], IV_V[x3], V_I[x4]],
    I_I[phrase[x1,x2,x3,x4]]      -> phrase[I_IV[x1], IV_V[x2], V_I[x3], I_I[x4]],
    I_I[phrase[x1,x2,x3,x4]]      -> phrase[I_IV[x1], IV_V[x2], V_IV[x3], IV_I[x4]],
    I_I[phrase[x1,x2,x3,x4]]      -> phrase[I_IV[x1], IV_V[x2], V_V[x3], V_I[x4]],
    I_I[phrase[x1,x2,x3,x4]]      -> phrase[I_V[x1], V_I[x2], I_I[x3], I_I[x4]],
    I_I[phrase[x1,x2,x3,x4]]      -> phrase[I_V[x1], V_I[x2], I_IV[x3], IV_I[x4]],
    I_I[phrase[x1,x2,x3,x4]]      -> phrase[I_V[x1], V_I[x2], I_V[x3], V_I[x4]],
    I_I[phrase[x1,x2,x3,x4]]      -> phrase[I_V[x1], V_IV[x2], IV_I[x3], I_I[x4]],
    I_I[phrase[x1,x2,x3,x4]]      -> phrase[I_V[x1], V_IV[x2], IV_IV[x3], IV_I[x4]],
    I_I[phrase[x1,x2,x3,x4]]      -> phrase[I_V[x1], V_IV[x2], IV_V[x3], V_I[x4]],
    I_I[phrase[x1,x2,x3,x4]]      -> phrase[I_V[x1], V_V[x2], V_I[x3], I_I[x4]],
    I_I[phrase[x1,x2,x3,x4]]      -> phrase[I_V[x1], V_V[x2], V_IV[x3], IV_I[x4]],
    I_I[phrase[x1,x2,x3,x4]]      -> phrase[I_V[x1], V_V[x2], V_V[x3], V_I[x4]],
    I_TI[phrase[x1,x2,x3,x4]]     -> phrase[I_I[x1], I_I[x2], I_I[x3], I_TI[x4]],
    I_TI[phrase[x1,x2,x3,x4]]     -> phrase[I_I[x1], I_IV[x2], IV_I[x3], I_TI[x4]],
    I_TI[phrase[x1,x2,x3,x4]]     -> phrase[I_I[x1], I_V[x2], V_I[x3], I_TI[x4]],
    I_TI[phrase[x1,x2,x3,x4]]     -> phrase[I_IV[x1], IV_I[x2], I_I[x3], I_TI[x4]],
    I_TI[phrase[x1,x2,x3,x4]]     -> phrase[I_IV[x1], IV_IV[x2], IV_I[x3], I_TI[x4]],
    I_TI[phrase[x1,x2,x3,x4]]     -> phrase[I_IV[x1], IV_V[x2], V_I[x3], I_TI[x4]],
    I_TI[phrase[x1,x2,x3,x4]]     -> phrase[I_V[x1], V_I[x2], I_I[x3], I_TI[x4]],
    I_TI[phrase[x1,x2,x3,x4]]     -> phrase[I_V[x1], V_IV[x2], IV_I[x3], I_TI[x4]],
    I_TI[phrase[x1,x2,x3,x4]]     -> phrase[I_V[x1], V_V[x2], V_I[x3], I_TI[x4]],


             ⋮


    I_I[measure[x1,x2]]       -> i[measure[P[x1], P[x2]]] weight 100,
    I_IV[measure[x1,x2]]     -> i[measure[P[x1], P[x2]]] weight 100,
    I_TI[measure[x1,x2]]      -> i[measure[P[x1], P[x2]]] weight 100,
    IV_IV[measure[x1,x2]]   -> iii[measure[P[x1], P[x2]]] weight 100,
    IV_V[measure[x1,x2]]     -> iii[measure[P[x1], P[x2]]] weight 100,
    IV_TIV[measure[x1,x2]]  -> iii[measure[P[x1], P[x2]]] weight 100,
    V_I[measure[x1,x2]]       -> v[measure[P[x1], P[x2]]] weight 100,
    V_V[measure[x1,x2]]       -> v[measure[P[x1], P[x2]]] weight 100,
    V_TV[measure[x1,x2]]      -> v[measure[P[x1], P[x2]]] weight 100,
    P[measure[x1,x2]]         -> measure[P[x1], P[x2]]
  },
    I_I
)
```

Figure 16: The rules of a td transducer CHORD, which assigns a chord progression common to pop music to the input tree.







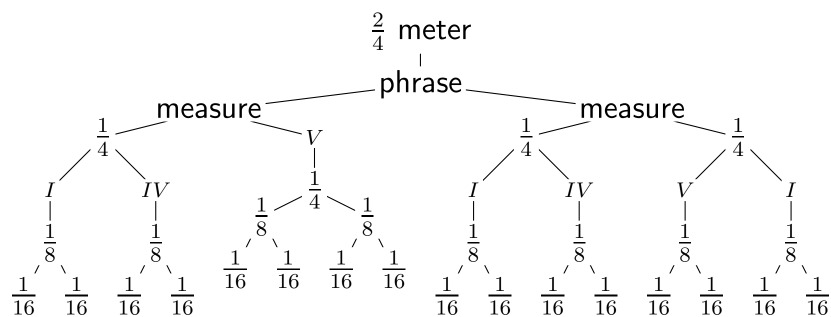


Figure 17: The td transducer CHORD assigns a chord progression common to pop music (the first two steps of the transduction have been omitted).

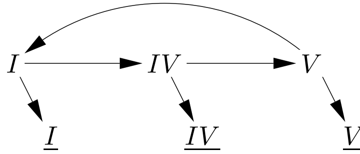


Figure 18: A graph of chord transitions augmented with sinks.

chord III' is rewritten as the terminal symbol for chord I. The state 'transition from chord III to chord V' as III and the state 'transition from chord V to chord I' as chord V. The implication is that if our initial state is $I{\Rightarrow}I$, the first chord in the generated tune is I, but there is nothing that says that the last chord must also be I.

```
I_III[measure[x1,x2]]   -> i[measure[P[x1],P[x2]]]   weight 100,
III_V[measure[x1,x2]]   -> iii[measure[P[x1],P[x2]]] weight 100,
V_I[measure[x1,x2]]     -> v[measure[P[x1],P[x2]]]   weight 100,
```

As we need to be able to specify not only our starting point, but also our destination, this problem needs to be solved. One remedy is to augment the progression graph with sinks. Compare the original graph for pop progression with the augmented graph in Figure 18.

The benefit of having sinks is that when some sink, suppose $\underline{I}$ (or TI as in Figure 16), occurs in a derivation, it will always be preceded by I. When the derivation terminates the $\underline{I}$ will disappear, leaving the I behind. So if we want our generated tune to begin with a V and and end with an I, and have a number of different chords in the middle, it suffices if we choose as our initial state the state $V{\Rightarrow}\underline{I}$. The additional chord-sink combinations account for the remaining states.

Assume now that we apply CHORD to an input tree. Obviously we do not want the output tree to represent an illegal chordprogression, that is, a sequence of chords hat does not correspond to a walk in the progression graph. There are two ways of doing this. One option is to remove every terminating rule with a left-hand state that corresponds to a forbidden transition and then use nondeterminism. This is the easy way to do it, because TREEBAG has a built-in look-ahead function that avoids dead end transductions. The drawback is that the definition of CHORD cannot be ignorant when nondeterminism is used, and will necessarily be very large (roughly 2000 lines in this case).

The other option is to use the fact that we now how low in a tree certain symbols can occur, and restrict the transduction to only derive intermediate states that will eventually be able to terminate. Suppose for example that the transition $I{\Rightarrow}V$ occurs at the level of phrases in a transduction. This is usually not a problem for there is plenty of time to fill out the gap with, for example a $I \Rightarrow III$ and a $III \Rightarrow V$ before the derivation terminates. But it is not a good idea to have rules like

```
I_V[eight[x1,x2]]  -> eight[I_V[x1], V_V[x2]],
```

as the symbol below an eighth is usually a sixteenth, which is a leaf, so any derivation using this rule can never terminate. With this option we do not only remove the terminating rules with forbidden transitions, but also every rule that can never lead to an allowed transition. This allows for a substantial trim down of CHORD, but even if we are able to remove half of the lines, we are still stuck with a file of 1000 lines. To write such a large definition is obviously an arduous and error prone task. And still we are only dealing with a chord progression of three out of the eight major triads; if there are n chords, then there will be $O(n^5)$ transitions.

Note that the version of CHORD discussed until now assigns a chord progression common to popular music, but that a corresponding td transducer could be built for any genre described in Figure 7. Since every such tree transducer would be as complex as the current version of CHORD, we implemented the perlscript `progression.pl` (available at [Högberg, 2005]). The script takes as input a plain text description of a progression graph and outputs the corresponding, nondeterministic, td transducer.

5.4 Tempo

After the chords have been set, the pace is decided. This is done in the same way as the chords were spread over the tree, but now three paces are substituted for the chords; *slow*, *fast*, and *medium*. We work with the assumption that it is easier for a listener to accept small tempo changes than big ones. As with non-harmonic tones, significant tempo changes do add character to a piece, but must applied carefully – a task beyond the capabilities of a top–down tree transducer. Instead, the only transitions allowed will be those in Figure 19. Sinks will be added before `progression.pl` is applied to the graph, yielding a tempo distributing tree transducer. The initial state of the tree transducer determines the pace at the start and end of the tune.

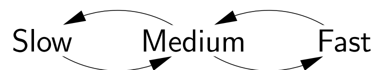


Figure 19: Allowed tempo transitions.

The tempos discussed here are relative, that is, if a part of the tree is supposed to be played quickly, in two two time this might mean a sequence of sixteenth notes and in four four time a sequence of eights. The relative values are later resolved by the td transducer ARRANGE, see Section 5.8.

5.5 Voices

By now the tree is equipped with markers for meter, chords and tempos. If there are to be multiple voices, then they must at least agree on these three properties. Otherwise the effect is far more likely to be that of a cacophony, rather than a chorus. This need for synchronisation is the reason why we could not generate the voices independently, avoiding copying tree transducers altogether.

In the current implementation of Willow there are three possible configurations of voices:

1. One voice.
2. One voice and accompaniment.
3. Two voices and accompaniment.

All configurations reserve an extra copy of some of the subtrees for analysis. The number of voices does not affect the efficiency much, but has an impact on the tree transducer that will later add cadences, see Section 5.10.

5.6 Invert voice tempo

An optional step after VOICES has been applied, is to invert the tempo of one voice. If there are two voices playing their own individual melodic line, the overall feeling may be clattered, unless they take turns resting and 'talking fast'. Before this step both voices move in identical tempo, so simply exchanging the fast and slow sections of one voice achieves the desired effect. Please note that although the voices no longer concord in tempo, there is still a synchronisation; the copying tree transducer of the previous section really is necessary. Example 4.11 showed the mechanisms of a td transducer that copies a voice and inverts the tempo of the duplicate.

5.7 Melodic arc

We have already decided on a chord progression, and on these chords we shall now weave a melody. When we assign the melody, our first principle will be to only move by, at most, four half steps a time. Too many large steps and the overall impression will be nervous and erratic. Our second principle is to use only notes that belong to the chords we committed ourselves to in the previous step. To use non-harmonic tones, as they are called, can add much to the music, but it should only be done with care. At this point we simply do not have the control mechanisms necessary to use them at the right places. Our last principle is that a phrase should always begin and end with the root note of the first and last chord, respectively.

It is easy to see that within a single chord we can always finds tones that are at most two whole steps apart. This follows directly from the definition of the triads of the major scale. But when we are trying to 'glue' two chords in different octaves together we have to turn to *inversions*. A chord is inverted if one or more of its constituent tones has been moved an octave up or down. If we also add the tones that belong to the inversions (and inversions of inversions) of our selected chords, then we will have a sufficiently dense set of tones to assure smooth transitions.

The melody assigned is relative; only the first note of every phrase is given as an absolute tone value. Thereafter follows a sequence of two operators, `up` and `down`, that determine the direction of the melodic arc. The `up` operator is taken to be an instruction to move as far upwards as is needed to find a tone in the current chord, while the `down` operator instead searches downwards for a suitable tone. If the melodic arc reaches further than three octaves below or above middle C, the display will print the warning 'Melodic arc is out of bounds.' and fail at writing the score. If the melody had instead been generated with absolute tone values, then nondeterminism could prevent a derivation that reaches too far from terminating. So why not use absolute values? One reason is that while there are only a bounded number of possible chords, there is an infinite number of possible tone values. Admittedly only a subset of these will actually come into consideration when we compose a tune, but even if this subset holds no more than 36 tones (mere three octaves), we would need hundreds if not thousands of states to handle all possible transitions. In Curtis Roads words, there is always a sharp pragmatic distinction between what can be done with some elegance and what requires ad hoc patchwork to be accomplished. Even if we did accept the inelegant solution of storing meter of a tune at two places in a derivation tree, we will desist from using hundreds or thousands of states when four states will do, as long as one accepts the occasional tune that cannot be played on regular instruments. One way of balancing a straying melodic arc is to use the optional transducer Flatten, which resets the octave at the beginning of each phrase.

5.8 Arrange

The td transducer Arrange compiles information concerning tempo, chord and melodic arc into relative tone values and absolute note lengths. It operates in a nondeterministic fashion – Arrange guesses that it has seen the last piece of relevant information before it writes a tone value or note length, and if it, as it traverses further down the tree, discovers that it has made a mistake, the transduction will fail to terminate. Despite the use of nondeterminism, the output trees produced by Arrange are uniquely determined by the input tree. Because of this, the functions performed by Arrange could be incorporated into Melodic Arc, but by placing them in a separate transducer, the definitions becomes cleaner and the system as a whole less coupled.

5.9 Anfang

As mentioned in Section 5.7, the melody is only restricted by the chord progression. Normally, a phrase should begin a the tonic or at least the root of some chord. This is where the td transducer ANFANG comes in – its purpose is to assure that the first note of every phrase is a root. It simply places an operator 'root' at the appropriate place, which should be interpreted as 'Find the next tone that is a root in whatever direction you are currently searching'.

5.10 Cadence

In a tune with only one voice (not counting accompaniment), cadences are laid out by a td transducer that uses nondeterminism to find the last two notes of a phrase, making the last note a tonic and the second-to-last note a dominant. This was discussed in Example 4.14. A problem arises when there are more than one voice whose phrases should end in a cadence. When this happens the tree transducer has to keep track of a separate 'guess' for each voice. As long as there is a small number of voices active, lets say less than ten, this is quite tractable. However, because the number of combinations of possible guesses is exponential in the number of voices, this solution does not scale. A td transducer can only 'remember' a finite number of things, so we are left with three options:

1. To ease the constraints on the cadence, allowing it to cover more notes then absolutely necessary. This is not a good idea when the phrases are short; there is a risk that they become little more than long cadences.
2. To give the td transducer that assigns cadences the possibility to split and merge notes, i.e. to adjust the rhythm to the guesses. Neither this second options is attractive; it clashes with the idea of a loosely coupled system where each module adds a specific attribute without affecting the work of previous modules.
3. To use a more powerful device, for example a random context tree transducer or an attributed tree transducer (as introduced in [Ewert et al., 2005] and [Fülöp and Vogler, 1998], respectively). This option has potential; the system would remain loosely coupled and, if the alternative is chosen carefully, also retain most of its simplicity.

In future versions of Willow, the course directed by the third option will be taken. However, in the current version with its maximum of three voices, the cadence is written by two nondeterministic td transducers (it is not necessary to modify the accompaniment). Generally, if k is a natural number, then k voices can be given cadences by k td transducers, which in theory can be collapsed into a single td transducer, but the standard construction leads to the increase in states, and it seems unlikely that this can be avoided.

5.11 AABA form

At this point in the generation process the tree represents two complete sections, A and B. The transducer Aaba form is the second and last copying tree transducer used. It performs the simple task of copying subtrees, so that the resulting tree represents a musical piece with two initial sections A and A, a bridge B and a concluding section A. How this can be done in the treebag environment was discussed in Example 5.1.

5.12 Scale

The transducer Scale assigns a major scale to the tune and, using the operators `heighten` and `lower`, adjusts the voices an octave up or down. Take the voice that shall play the accompaniment, for example. If the chosen scale is *c*-, *c*♯-, *d*- or *d*♯-major, then this voice is heightened. If the scale is *g*-, *g*♯-, *a*- or *a*♯-major, the voice is lowered, and in any of the four remaining cases the voice is kept unchanged. The reason for moving the registers is to keep the voices from tangling, in which case it would be difficult for the listener to separate the melodic arcs. In addition to the major scales, pure minor scales could be used, and with a slight alteration to the Score display, also melodic and harmonic minors (because the tones of the melodic and harmonic minors depend on whether the tune is ascending or descending).

5.13 Simple accompaniment

The final step before the tree is ready for interpretation is to add accompaniment. One benefit of the accompaniment is that it brings out the harmony of the melody and allows the listener to guess what will happen next. Another is that it adds a solid structure that stabilises a wandering melody.

Also the accompaniment is added by means of a td transducer, in this case Simple accompaniment. It takes the form of a simple reordering of the chord, exactly how this is done depends on whether the chord is a major, minor or diminished triad, but also on the meter of the tune. The tables of Figure 20 show the possible configurations. Suppose that a diminished triad shall be accompanied in a tune in three four time, then by the table marked Diminished triad we have that the accompaniment will be the root of the triad, the minor third (relative to the root), the diminished fifth and finally the minor third of the triad again.

The accompaniment of a chord is played so that one note in the accompaniment corresponds to one beat in the meter. This means that normally the accompaniment of one chord lasts precisely one measure. Sometimes when there is a rapid chord progression there is not enough time for this type of accompaniment; in those cases only the root of the triad is played on each beat.

	Major triad
$\frac{2}{2}$	root, perfect fifth
$\frac{2}{4}$	root, perfect fifth
$\frac{3}{4}$	root, major third, perfect fifth
$\frac{4}{4}$	root, major third, perfect fifth, major third

	Minor triad
$\frac{2}{2}$	root, perfect fifth
$\frac{2}{4}$	root, perfect fifth
$\frac{3}{4}$	root, major third, perfect fifth
$\frac{4}{4}$	root, major third, perfect fifth, major third

	Diminished triad
$\frac{2}{2}$	root, diminished fifth
$\frac{2}{4}$	root, diminished fifth
$\frac{3}{4}$	root, minor third, diminished fifth
$\frac{4}{4}$	root, minor third, diminished fifth, minor third

Figure 20: The accompaniment depends of whether the chord is a major, minor or diminished triad, and also on the time of the tune.

5.14 Score display

After the accompaniment has been added, the composition is done and what remains is to present the user with the resulting tune. For output an instance of the class `display.scoreDisplay` is invoked, with most methods inherited from from `display.treeDisplay`. What has been added is a new command 'Write mup file', which, when invoked, writes the tree to a file in mup format. To make the conversion, Score display keeps track of which key was played last, the current chord and the direction of the melodic arc.

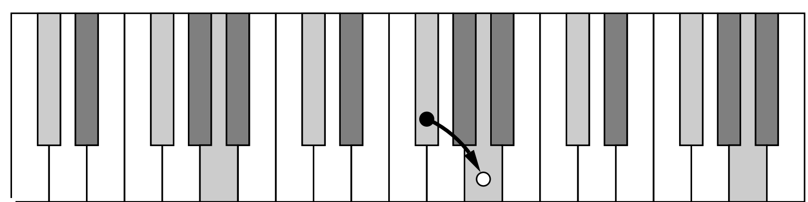


Figure 21: The black dot denotes the last key played, the light gray keys are those that belong to inversions of the chord *ii*, and the key with the white dot will be played next, as the melodic line is ascending.

Example 5.2 Suppose that the tune generated is in E major, that the last key played was an $f\sharp$, the current chord is *ii* and that the melodic line is ascending. Figure 21 illustrates the situation: The black dot marks the last key played, the light gray keys are those that belong to inversions of *ii*, and the key with the white dot is the key to be played next. Another situation arises when the tune reaches a cadence; it is no longer sufficient that the notes are played in the specific chord, they must be played as the root of the chord. This means that the possible keys become more scarce, forcing the melody to take greater leaps.

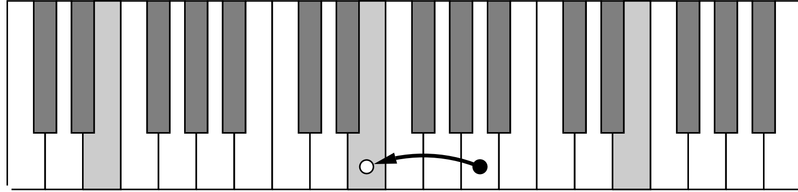


Figure 22: The black dot denotes the last key played, the light gray keys are transpositions of the root of chord I, and the key with the white dot will be played next, as the melodic line is descending.

Figure 22 illustrates such a situation: The black dot marks the last key played, the light grey are transpositions of the root of chord I, and the key with the white dot is the key to be played next.

5.15 Mup

After the tree has been written to file, the converter software Mup is used to produce a score in postscript (e.g. the score in Figure 23), and also a Midi file. The format that Mup accepts is described in [Arkkra, 2005]. Mup version 5.0 is executable on both $*$–nix and Windows machines and can be obtained from Arkkra Enterprises, `http://www.arkkra.com`.

6 Theoretical Aspects

This section discusses two aspects of Willow, descriptional complexity and statistical behaviour, which are of a more theoretical nature than the practical concerns of Section 5.

6.1 Descriptional complexity

The implementation of Willow discussed in this paper generates tunes with four phrases, each of length two to four measures. As this inflicts an upper bound on the lengths of the generated tunes, and since all attributes are discreet, the reader may rightfully draw the conclusion that the language of all trees produced by Willow is a finite tree language. However, by adding a single rule to the initial regular tree grammar, the phrases generated can be made arbitrary long – none of the tree transducers that follow require their input tree to be of a certain size, and neither does the Score display. For this reason we shall discuss the descriptional complexity of Willow as if the language that it generates was truly infinite.

Tree Symphonee

(A generated piece)

Author: J. Högberg

Tune: Willow

Figure 23: A score generated by Willow and converted to postscript by Mup.

An appealing property of Willow is the simplicity of its components (these are listed in Table 2); neither regular tree grammars nor top–down tree transducers are very powerful devices. We will now apply results concerning *composition* of transductions, and show that much of the simplicity is retained when it is the system as a whole that is under consideration.

Let us denote the composition of td transducers td_1 and td_2 by $td_2 \circ td_1$, i.e., $td_2 \circ td_1(t) = \bigcup_{t' \in td_1(t)} td_2(t')$. Theorem 6.1 on the following page is a result by Brenda S. Baker. The reader should note that the td transducers that constitute Willow are all nondeleting, which means that the second requirement of the theorem is always met.

Component	Type	Del.	Lin.	Det.
AB FORM	regular tree grammar	N/A	N/A	No
CHORD PROGRESSION	td transducer	No	Yes	No
TEMPO	td transducer	No	Yes	No
VOICES	td transducer	No	No	Yes
FLIP	td transducer	No	Yes	Yes
MELODIC ARC	td transducer	No	Yes	No
ARRANGE	td transducer	No	Yes	No
CADENCE	td transducer	No	Yes	No
ANFANG	td transducer	No	Yes	No
AABA FORM	td transducer	No	No	Yes
SCALE	td transducer	No	Yes	Yes
ANALYSIS	td transducer	No	Yes	Yes
ACCOMPANIMENT	td transducer	No	Yes	Yes

Table 2: The constituents of Willow, ordered as they occur in the generation process. The abbreviations del., lin., and det. stand for deleting, linear and deterministic, respectively.

Theorem 6.1 [Baker, 1979] The composition $td_2 \circ td_1$ of td transformations td_1 and td_2 is a td transformation if both of the following two conditions are satisfied:

1. td_1 is deterministic or td_2 is linear, and
2. td_1 is total or td_2 is nondeleting.

If, in addition, both td_1 and td_2 are linear, deterministic, total, or nondeleting, then $td_2 \circ td_1$ has the respective property as well.

First, we consider descriptional complexity at the level of phrases, that is, the complexity of the subtrees that represent phrases. If there is just one voice, then these subtrees can be generated by a regular tree grammar – only the td transducer VOICES modifies subtrees representing phrases by means of copying rules. Now, assume that the format of the tree is slightly modified, so that information for all voices are stored in the same subtree. This would allow any natural number of voices, whereas the subtrees representing phrases could still be generated by a regular tree grammar. Although a modification of this type will probably require a new set of nonterminals for each added voice, it can quite clearly be done. The reason why voices are stored separately in the trees that Willow processes, is that it increases legibility and thus eases debugging. However, the language of music that Willow generates is not regular, due to the AABA form of the tunes produced.

Recall that regular tree languages are closed under linear td transductions. As a consequence, we can combine the apparatus of AB FORM, CHORD PROGRESSION, and TEMPO in a single regular tree grammar. To establish the complexity of the system as a whole we also need Theorem 6.2. The abbreviation BST_k in the theorem refers to the class of languages that can be generated by a branching tree grammar of depth $k \in \mathbb{N}$ (as introduced in [Drewes and Engelfriet, 2004]). For an introduction to branching tree grammars, see also Appendix A of [Drewes, 2005].

Theorem 6.2 Let $n \in \mathbb{N}$. For every tree language L the following statements are equivalent:

1. $L \in BST_n$.
2. There are td transducers $td_1, \ldots, td_n$ and a regular tree language L_0 such that $L = td_n(\ldots td_1(L_0)\ldots)$.
3. There are total td transducers $td_1, \ldots, td_n$ and a regular tree language L_0 such that $L = td_n(\ldots td_1(L_0)\ldots)$.

Hence, BST_n is equal to the closure of the class of regular tree language under n td transformations.

According to Theorem 6.1, we may collapse a sequence of td transducers by substituting a single td transducer for every deterministic td transducer and its successor and every linear td transducer and its predecessor. After a number of iterated substitutions, the system has been reduced to a regular tree grammar and two td transducers (see Table 3), and cannot be simplified further using Theorem 6.1. Theorem 6.1 together with Theorem 6.2 thus implies that the language of all tunes generated by Willow is contained in BST_2.

Theorem 6.3 The tree language generated by Willow is an element of BST_2.

6.2 Statistical distribution

It is argued in [Baroni, 1983], that when working with a generative grammar, precautions must be taken so that if a certain feature occurs with frequency f in the repertoire, then the same feature occurs with frequency close to f in the language generated by the grammar. If Willow is ever re-written to mimic a specific repertoire, this will certainly be a relevant issue. To control the statistical distribution of a feature, one could take advantage of the fact that TREEBAG allows the user to specify an optional argument with every rule in an rtg or a td tree transducer, a rational number called its weight. The weight affects in the obvious way the probability that its associated rule will be selected during a nondeterministic derivation. This does not affect the fact that the generated language belongs to BST_2, for if a weight is zero, then a rule simply 'disappears',

Original Component	Component after collapse
Ab form	regular tree grammar
Chord progression	
Tempo	
Voices	copying nondet. td transducer
Flip	
Melodic arc	
Arrange	
Cadence	
Anfang	
Aaba form	copying nondet. td transducer
Scale	
Analysis	
Accompaniment	

Table 3: After repeated applications of Theorem 6.1, two td transducers remain.

and as long as there is a positive probability that a rule will be used at all, any tree that could be generated without the added weights will be generated with probability strictly greater than zero (since we consider only finite trees). Remember that the language of an rtg or td tree transducer is defined as the set, not the probability distribution, of all possible trees that can result.

7 Future work

The type of tunes produced by Willow is restricted in many ways; they are all in major scales, on the ABAA form, there are no extrinsic tones and no (intended) decorations. In the following, we discuss a few extensions that may lead to interesting future work.

7.1 The Fugue and macro tree transducers

The Fugue is a musical form associated with the Baroque era. Written for a fixed number of voices, it centres around a single *theme* (viz. a minimal tune). The theme is performed by the voices in an imitative fashion, much like a canon. Today Johann Sebastian Bach is considered to be the foremost ambassador of this strict form; in his famous *The art of Fugue* we find seventeen canons and fugues built on one single theme. It might be that the composition of a fugue could be done in an algorithmic fashion, using a regular tree grammar to generate

a theme, which is then extended to an entire tune using copying and modifying td transducers. There is, however, a formal device which at first glance seem much more appropriate for this type of composition. A *macro tree transducer* can be thought of as a td transducer, the states of which can carry a fixed number of parameters (trees) with them as they work their way down the input tree. These parameters are used in the rewrite steps as building blocks for the output tree [Fülöp and Vogler, 1998]. The idea is that one could generate a theme and construct a fugue based on this theme, using a single macro tree transducer.

7.2 Control of Accent

As discussed in Section 5.13, notes generated by the system described in this paper are accented by chord changes, and to a lesser degree by duration. In future versions we wish to also use other, more sophisticated forms of accentuation, such as peaks in melody, contrasting durations, or atonal notes. To do this without resorting to context sensitive devices could prove to be quite be a challenge, as the accent of a note is gauged relative to a background of surrounding notes, and not as an absolute value.

7.3 Extended Interaction

In the introduction we compared the generation process to an assembly line, where a human composer can monitor and direct the work. At present, the composer can only do this indirectly by selecting and arranging the formal devices. To further capacitate the composer, a midi synthesiser could be used to input themes and pieces into the computer. The midi files would then be converted to a structured text format (there are a number of open source systems available that do this). In addition, one would have to create a way to import trees into Treebag, where they could be inserted into the generation process. Another way to empower the user is to allow him or her to choose the applied rules explicitly. If the user is only interested in the resulting tunes, then this would of course be a nice feature. But if, on the other hand, the users interest is of a theoretical nature, then the possibility to manually select the rules to be applied may distort the understanding of the generated language.

7.4 Mid-tune key changes

A common feature of interesting human-composed music is for a change of key (a *transition*) to occur one or more times in a single tune. The effect is a corresponding mood change, but the transitions also divide the music into distinct parts. Modelling transitions will probably be easy from a theoretical point of view; we are only dealing with a finite number of possible transitions. However,

to implement a tree transducer that does this elegantly as well as efficiently, might be more demanding.

7.5 An algebra for music

As mentioned in Example 5.1, the free term algebra used in Willow is needed because Treebag displays only accept input from algebras – the free term algebra simply interprets a tree as the tree itself. The actual interpretation of the output trees is made by the Score display, while the realization of sound is left to external software. This is not a very elegant solution, as it disagrees with the general design of Treebag – algebras interprets the trees, while displays visualise (or in this case, perform) the results.

In future versions, it would be nice to replace the free term algebra by an algebra for music, and have Treebag both performing the generated pieces and displaying the scores in a postscript display (perhaps using Mup internally). It is not immediately obvious what an algebra for music would look like, but we believe that while searching for an appropriate definition many interesting questions can arise.

Acknowledgements

I would like to thank my supervisor Frank Drewes for his guidence and persistent encouragement. I also wish to express my gratitude to Per-Anders Sundbaum at the Department of Arts and Crafts for his contagious love of music, as well as to Claude Lacoursière for all the advice given.

References

[Wik, 2005] (2005). Wikipedia – The Free Encyclopedia. Internet resource. Available at `http://en.wikipedia.org/wiki/Rhythm`. Accessed Feb 2005.

[Adams, 2004] Adams, R. (2004). Musictheory.net. Internet resource. Available at `http://www.musictheory.net`. Accessed Jan 2005.

[Alvira, 2004] Alvira, J. R. (2004). Teoria — music theory web. Internet resource. Available at `http://www.teoria.com`. Accessed Jan 2005.

[Arkkra, 2005] Arkkra (2005). Mup user's guide. Internet resource. Available at `http://www.arkkra.com/doc/uguide.ps`. Accessed 25 Jan 2005.

[Baker, 1979] Baker, B. S. (1979). Composition of top–down and bottom–up tree transductions. *Information and Control*, (41):186–213.

[Baroni, 1983] Baroni, M. (1983). The concept of musical grammar. *Music Analysis*, 2(2):175–208.

[Bel and Kippen, 1992] Bel, B. and Kippen, J. (1992). Bol processor grammars. In Laske, O., Balaban, M., and Ebcioglu, K., editors, *Understanding Music with AI — Perspectives on Music Cognition*, pages 366–401. Cambridge, MA: MIT Press.

[Benward and Carr, 1999] Benward, B. and Carr, M. A. (1999). *Sightsinging Complete*. McGraw-Hill, sixth edition.

[Blood, 2004] Blood, B. (2004). Music theory and history online. Internet resource. Available at `http://www.dolmetsch.com/theoryintro.htm`. Accessed Jan 2005.

[Chuang, 1995] Chuang, J. (1995). Mozart's musikalisches Würfelspiel. Available at `http://www.worldvillage.com/jchuang/Music/Mozart/mozart.cgi`. Accessed Feb 2005.

[Dempster, 1998] Dempster, D. (1998). Is there even a grammar of music? *Musicæ Scienticæ*, 1(2):55–65.

[Drewes, 1998] Drewes, F. (1998). Treebag—a tree-based generator for objects of various types. Report 1/98, Univ. Bremen.

[Drewes, 2005] Drewes, F. (2005). *Grammatical Picture Generation – A Tree–Based Approach*. Springer.

[Drewes and Engelfriet, 2004] Drewes, F. and Engelfriet, J. (2004). Branching synchronisation grammars with nested tables. *Journal of Computer and System Sciences*, (68):611–656.

[Egler et al., 2003] Egler, A., Mitchell, R., and Taupin, D. (2003). Musixtex — using tex to write polyphonic or instrumental music. Manual. Available at `http://icking-music-archive.org/software/indexmt6.html`. Accessed 17 Jan 2005.

[Ewert et al., 2005] Ewert, S., Drewes, F., Högberg, J., van der Merwe, B., du Toit, C., and van der Walt, A. (2005). Random context tree grammars and tree transducers. Technical report, Computer Science, Umeå University.

[Fülöp and Vogler, 1998] Fülöp, Z. and Vogler, H. (1998). *Syntax-Directed Semantics: Formal Models Based on Tree Transducers*. Springer-Verlag New York, Inc.

[Gécseg and Steinby, 1984] Gécseg, F. and Steinby, M. (1984). *Tree Automata*. Akadémiai Kiadó, Budapest.

[Gécseg and Steinby, 1997] Gécseg, F. and Steinby, M. (1997). Tree languages. In Rozenberg, G. and Salomaa, A., editors, *Handbook of Formal Languages. Vol. III: Beyond Words*, chapter 1, pages 1–68. Springer.

[Högberg, 2005] Högberg, J. (2005). Willow. Internet resource. Available at `http://www.cs.umu.se/~johanna/willow/`.

[Kerman and Tomlinson, 2000] Kerman, J. and Tomlinson, G. (2000). *Listen*. Bedford, Freeman, and Worth, fourth edition.

[Konecky, 2004] Konecky, L. (2004). Basic music theory. Internet resource. Available at `http://www.alcorn.edu/musictheory/`. Accessed Jan 2005.

[Lerdahl and Jackendoff, 1977] Lerdahl, F. and Jackendoff, R. (1977). Toward a formal theory of tonal music. *Journal of Music Theory*, 21(1):111–171.

[Lindblom and Sundberg, 1970] Lindblom, B. and Sundberg, J. (1970). Towards a generative theory of melody. *Swedish Journal of Musicology*, (52):77–88.

[Ponsford et al., 1999] Ponsford, D., G, G. W., and C, C. M. (1999). *Journal of New Music Research*, 28(2):150–177.

[Prusinkiewicz, 1986] Prusinkiewicz, P. (1986). Score generation with L-systems. In Berg, P., editor, *Proceedings of the International Computer Music Conference 1986*, number 1, pages 455–457. Royal Conservatory, The Hague, Netherlands.

[Roads, 1979] Roads, C. (1979). Grammars as representations for music. *Computer Music Journal*, (3(1)):48–55.

[Wiggins, 1998] Wiggins, G. A. (1998). Music, syntax, and the Meaning of "Meaning".