1. INTRODUCTION

Grammars are a formal notation for describing the structure of elements and processes. The most common application of grammars is the description of natural and computer languages. Other applications include describing the structure of line drawings, the outline of human chromosomes, and the growth of simple organisms.

The structures best described by grammars are constructed from elements using a set of rules. These elements are structures made up of simpler elements, again using some set of rules. This whole process begins with the lowest level, or primitive, element in the structure. A primitive element is an element which cannot be further divided.

Each rule specifies how a set of elements are to be combined in order to produce a larger element. In the case of written language the primitive elements are words. The rules are used to combine the words into phrases, and the phrases into sentences. The primitive elements in a grammar are called terminals, and the rules are called productions. The larger elements which result from applying productions are called non-terminals. A grammar is a collection of productions.

2. A TUTORIAL INTRODUCTION TO PROD

PROD is a computer program for music composition which uses grammars to describe the music to be generated. The input to PROD is a grammar which is supplied by the musician. The output from PROD is a musical work which conforms to the grammar used as input. Depending upon the form of the productions more than one score can be derived from the grammar. When more than one musical structure, or "sentences" can be derived from a set of productions, PROD uses a stochastic process to determine the result. We will see later how this non-deterministic behavior can be controlled by the musician.

In this section we look at how grammars are constructed for use with PROD.

Since most musical compositions have notes as their most primitive component, one of the terminals we will use is called "note". Note this class of terminals will be used for all notes in the composition. A production in a PROD grammar is made up of two parts: a left side and a right side. The right side is the role for producing new elements. While

---

Author's Current Address: Dept. of Computer Science, McMaster University, Hamilton, Ontario, Canada, L8S 4K1.

1 Elements are often referred to as "objects" in the literature. We have not used that term so as to avoid confusion with ISISalanbe from which one element may be an object.
the left side is the name of the new symbol, or non-terminal which results from applying the rule. The two sides of the production are separated by a ‘,’ and the end of the production is signaled by a ‘;’. As an example, the following rule states that the non-terminal "ex" is composed of a single note.

```
ex: note
```

This example is complete in itself and results in a one-note score. If the score is played, it will be noticed that the note has somehow assumed a pitch, duration, and loudness. These are derived automatically from "default" values, which are assumed for parameters left unspecified by the user.

If three notes in a row were desired the following production would be used.

```
ex: note note note
```

Again, the result can be heard. However, with all three notes the same, the result is not especially interesting. Thus, we need some way of changing the characteristics of the notes produced by PROOD. In the SSSP system, there are six aspects of a note which can be controlled by the composer (Rusio, 1978). They are:

• Timbre, which is specified by the (alphabetical) name of the "instrument" which plays the note;
• Frequency/Pitch, specified in either cycles-per-second or Decibels of Pure Tones or American pitch notation (e.g., C4, G3);
• Duration, specified as a numerical value in units of thirty-second notes;
• Volume, specified as an integer between 0 and 255, with 120 being about mf.
• Channels: one of the four audio output channels.
• Entry Delay, specified in same way as duration, represents the time delay before the start of the note which follows, thereby controlling rhythm.

These six parameters can be specified for each note by appending a list of their values, enclosed in parentheses, to the "note" symbol. Thus, we can repeat our first example, but specifying all aspects of the note this time. One point to note in the example is that the attributes of the note must be specified in the order given above.

```
ex: note(chime, c2, 32, 190, 0, 32);
```

"Ex3" will have a duration of a whole note, since a duration of 32 was specified.

Not all attributes of a note need be explicitly specified by the user. For example a quarter note played by a trumpet at a frequency of 440 hertz would be specified as

```
ote(trumpet,440.0)
```

For parameters left unspecified, the corresponding attributes of the preceding note are carried over. Thus, the following production will generate a three-note sequence, with each note having the same duration, and orchestrations (trumpet).

```
ex: note(trumpet,c4,4,150) note440 note440
```

102
In the example, note that where parameters are left unspecified, the leading commas must be included so as to enable PEGO to know for what parameter a particular value is intended.

The entry delay parameter is used for building chords and melodies. This parameter specifies the time delay between the start of the current note and the start of the note immediately after. If the entry delay is zero a chord is formed. If the entry delay is the same as the duration of the note then a melody is formed. A two note chord is produced by the following production

```
chord : note(trumpet.c4,.8,100.,0) note(c4,0,0);
```

Pitches can be specified in relative, as well as absolute, terms. This is shown in the next example, which is functionally identical to the previous one.

```
chord : note(trumpet.c4,.8,100.,0) note(third(c4),0,0);
```

The pitch specification of the second note is given as an interval name followed by the reference pitch (that is, the second note will have a pitch a major third above c4). The reference pitch must be enclosed in parentheses.

A non-terminal is the name of a particular collection of notes. This name can be used in other productions. Each time a non-terminal is used it is replaced by the collection of notes it represents. Non-terminals and the productions in which they appear can be used to represent the structure of a musical composition. For example, a musical composition with the structure A-B-A can be represented by the following production:

```
composition : A S A;
```

Obviously, this production is not complete in itself; however, it does permit the basic structure to be succinctly defined. Both A and B are non-terminals which represent the basic sections of the score, and other productions must be specified in order to derive their actual notes. If the A part of the composition is to be three C major chords, the following productions would be used

```
A : cmajor cmajor cmajor;
```

```
cmajor : note(trumpet.c4,.8,100.,0) note(third(c4)) note(third(c4),0,0);
```

Likewise, if the B part is to have three G7 chords then the following productions would be used

```
B : ...
```

103
The underlying (or "deep") structure of this particular composition is shown in Figure 2. Diagrams like this can be constructed from the productions in any grammar. This type of diagram shows how the whole composition is divided into parts, and the parts into sub-parts. Each non-terminal in a grammar is described in terms of the terminals and non-terminals from which it is constructed. In the shown grammar the non-terminal "composition" is described in terms of the non-terminals "A" and "B". Each of these lower level non-terminals is in turn described in terms of other non-terminals. In the case the non-terminals "major" and "sevenenth". Description of this type which are based on different levels of detail are called hierarchically.

In previous examples we used productions to generate chords. In these productions the base note of the chord was fixed. That is, if we wanted a major chord with 4 as its base we would have to write a separate production for it. This is not a very satisfactory arrangement. To get around this problem we use parameterized productions. In these productions the values which change each time the production is used are marked with special symbols. A production for the non-terminal "major" which generates a major chord for any base note can be constructed in the following way. Starting with a production like "major" we replace every occurrence of the pitch of the base note by the special symbol "B1". This symbol will last be replaced by the actual pitch of the base note. This replacement process gives the following.

Figure 1

\[ B : \text{seventh} \cdot \text{seventh} \cdot \text{seventh} : \]
\[ = \text{enth} : \text{note(trumpet,4.6.100,3)} \]
\[ \text{note(third(3))} \]
\[ \text{note(seventh(9)),...9);} \]

The above five productions form a grammar which can be used as input to PROBO. The result is a musical score starting with three C chords, followed by three G7 chords, and ending with three C chords. The actual score produced by PROBO is shown in Figure 1.
production.

major: note(trumpet,$1$.8.1000.0)
note(chord($1$))
note(mi($1$),...,$6$);

Each time the non-terminal "major" is used in a production the pitch of the base note must be specified. A parameter, such as the pitch of the base note, can be specified by appending its value, enclosed within parentheses, to the non-terminal symbol. Note the similarity between specifying parameters for notes and for non-terminals. An example of the use of the non-terminal "major" is:

series: major($4$) major($8$) major($6$);

Each of the pitches used with "major" in the above example will replace the symbol "$1$" when it comes to generating a chord. This will give three different chords all produced by the same production.

Using the above two productions as input P$9002$ produces the score shown in Figure 3.

A more general chord production would also include parameters for object, duration, and volume of the chord. The following is such a production:

Major: note($1$, $1$.2.4.$3$, $4$.0)
note(chord($2$2$2$))
note(mi($2$),...,$2$);

The parameters to this production are (in the order in which they must be specified) object, root, duration, and volume. The number following the "$1$" symbol identifies the
parameter value to be used. Thus the symbol "P" stands for the first parameter, and
the symbol "D" stands for the third parameter. This production could be used in the
following way.

Series: major(trumpet,cs3,128)
major(trumpet,gs,120)
major(trumpet,gs,8,160);

Another terminal which can be used in PROD grammars is called "score". This ter-

minal is used to include pre-composed scores in the composition described by a gram-
marr. This terminal has one parameter, the name of the score to be included. If we had
two scores, "para" and "partb", and we wanted to use them to produce a new score
with the structure A B A the following grammar could be used.

composition : A B A;
A: score("para");
B: score("partb").

The score referenced by a "score" terminal can be any score in the MUSIS system. It
need not be one that was previously generated by PROD.

Each of the grammars that we have seen up until now describes only one score.
That is, every time one of these grammars is used as input to PROD the same score is
produced as output. We call this type of generation process deterministic. It is possi-
ble to construct grammars which describe more than one score. When one of these
grammars is used as input to PROD one of the scores it describes will be generated. A
different score may be produced each time the grammar is used. This is referred to as
a random or non-deterministic generation process.

In order to construct a grammar which describes more than one score we use pro-
ductions with right sides consisting of several alternatives. Each of these alternatives is
made up of terminals and non-terminals and is similar to the right sides of productions
we have seen so far. When PPROD encounters a production which has several alterna-
tives one of them is chosen at random. The symbol "|" (or bar) is used to separate the
alternatives which make up the right side. Thus, a production with two alternatives
would be written as

\[ A \rightarrow B | C \]

This production states that when the non-terminal "A" is found in a production it
is replaced by either A or B.

To show how this type of production is used consider the following example. We want
to produce 6 bars of chords. The first bar must contain four C chords and the last bar
four D7 chords. The remaining bars can have either chord, but all the chords in a bar
must be the same. The following grammar will generate 6 bars which satisfy this
description.

\[
\text{c5} : \text{oobar bar bar bar bar gobar} ; \\
\text{obar} : \text{obar gobar} ; \\
\text{major} : \text{major(C4) major(C4) major(C4) major(C4)} ; \\
\text{gobar} : \text{seven(H4) seven(G5) seven(G5) seven(G5)} ; \\
\text{major} : \text{note(trumpet,1,2,100,0)} \\
\text{note,flat(H1)} ; \\
\text{seven} : \text{note(trumpet,1,2,100,3)} \\
\text{note,flat(H1)} ; \\
\text{note,seventh(H1)} ;
\]

One of the scores produced by PPROD when given this grammar as input is shown in Fig-
ure 4.

In the above grammar the non-terminal "bar" is just as likely to generate a bar of C
chords as it is to generate a bar of D7 chords. In this case we say that the bar of C
chords and the bar of D7 chords have an equal probability of occurring. This may not be
what we want. We may want the C chords to be three times as likely as the D7 chords.
In this case the following production would be used in our example grammar

\[
\text{bar} : \text{obar | oobar | gobar} ;
\]

In this production "obar" occurs in three alternatives and "gobar" occurs in only one. Thus,
"obar" is three times more likely. If this production was used in the input gram-
mar we would expect that three of the four middle bars would have C chords. It is pos-
sible to have only two bars of C chords or all four bars of C chords. The above produc-
tion says that on average there will be three bars of C chords. In our example produc-
tion "obar" has a probability of 3/4 and "gobar" has a probability of 1/4.

There is a shorter way of writing the last production. Instead of repeating the alter-
native we append the number of repetitions - "closedit in brackets, to the end of it.
Thus, we can rewrite the previous production as

\[ \text{bar : orf[2]} | \text{bar} \]

In PROD the musician has control over where randomness will occur and the degree of this randomness. Only those productions which have more than one alternative can give rise to randomness. By specifying the probability for each alternative the musician can control how often an average that alternative will appear in the composition.

A non-terminal can be used in the right side of the production which defines it. This is known as a recursive production. However, if such a production has only one alternative on the right side problems will arise. Consider the following example.

\[ A : A B \]

This production says that A is to be replaced by itself followed by B. Since there are no other alternatives which can be used to replace A this replacement process will continue forever. On the other hand, if a production with more than one alternative has at least one alternative which does not contain the non-terminal being defined then an infinite sequence of replacements will not be produced. The sequence will stop whenever one of the alternatives that does not contain the non-terminal being defined is used in a replacement.

This type of production can be used for producing sequences of notes or chords which are of an arbitrary length. In a previous example we presented a grammar which produced 5 bars of chords. This grammar can be modified to produce an arbitrary number of bars by changing the production for "exit" to
exit : char middle gbar :
middle : bar middle (3) bar :

If this grammar is used we would expect the middle part to be an average 4 bars long. This is because the first alternative has probability 3/4 and each time it is selected the production will be repeated. This grammar could generate a middle part as short as one bar long. The probability of this is 1/4, which means it would happen on average every four times the grammar is used. It could also generate a very long middle part, but this would happen very rarely.

3. THE USE OF PROD IN COMPOSITION

PROD is one of the many tools available to the composer using the SISYPH music system. Some of the composing tools that are available include an interactive score editor, an object editor, score transformation programs, score orchestration programs, and other stochastic music composition programs. These tools are described in detail elsewhere (Butx, 1993), so no further mention will be made of them here. It is not our intention that the composer would only use PROD and produce one large grammar for his composition. Instead he would only use PROD for those parts of his composition for which it is best suited. PROD does not provide all the facilities necessary for computer composition since most of these facilities are already provided by the other tools in the system. Instead PROD concentrates on two particular composing tasks.

The first of these tasks is generating parts or sections of scores. These sections may be used with pip- or composing tools or combined with other parts of the score which were produced by other composing techniques. The score sections produced by PROD may vary from one or two bars long to major components of the piece. Examples of how these score sections are generated have been presented in section 2.

The second composing task involves putting together the sections of a composition to produce the final score. Some musicians develop their compositions as a number of separate sections. This could be the result of using different composing tools or techniques on each section or writing the sections at different times. Through the use of the "score" terminal PROD can be used to organize these different sections into a final score. The grammar for putting together the sections can be stored on a file and called up whenever a copy of the final score is required.

4. RELATION TO FINITE STATE MACHINES AND MARKOV CHAINS

Much of the early work on composing programs, such as that described in Experimental Music (Hiller & Lewinson, 1959), was based on the theory of finite state machines. Finite state machines are equivalent in power to type 3 grammars. A type 3 grammar is a grammar in which all the productions have at most one non-terminal on the right side. Obviously, PROD will accept any type 3 grammar. Any composition which has been produced by a finite state machine based composition program can also be realized with PROD.

A number of stochastic composition programs have been based on the theory of Markov chains. The most notable of these is the work of Tzemakh (1971). Markov chains can be looked upon as finite state machines with non-deterministic state transition runs. The non-deterministic productions in PROD behave in the same way as the random state transitions in Markov chains. The same basic probability model applies in both cases. The state transition probabilities are the same as the probabilities that are attached to the different alternatives of a production. Thus we can see that the Markov chain based composition techniques are also covered by PROD.

109
5. OTHER GRAMMAR FORMALISMS

Other grammar-based composition programs have appeared in the literature. Most of these other systems are more comprehensive than PROS. This is mainly because some of the features provided by these composing programs are already provided by other programs in the SSEP system. Therefore, they need not be included in PROS.

A good example of this is the GUGL system (Wolsten, 1979). This program has facilities for performing score transformations such as inversion and for generating sound parameters used by the synthesis hardware. The SSEP music system already has standard tools for performing these functions; therefore, PROS need not provide them. The GUGL system is more powerful than PROS in that it accepts a wider range of grammars. Using this system it is quite possible to write one grammar to produce a complete composition. This never was an important goal in the design of PROS. Instead, PROS was aimed at producing small sections of scores. One of the major results of this design goal is that PROS is relatively easy to learn and use. This makes it ideal for use by the composer naive musician. It is easy for him to make use of PROS in his routine compositional work. This is one of the major advantages of a simple system like PROS.

Seeds (1977) has done some interesting work on using grammars in musical composition. The unique part of this work is the parallelism which is interpreted into his grammars. The grammar formalism developed by Seeds allows several lines of music to be developed simultaneously. This is particularly interesting since most music involves several parallel streams of activity. The only way a similar effect could be achieved with PROS is to develop a separate grammar for each of the lines and then combine the resulting scores. Parallelism is a significant feature which is worth consideration in future computer composition programs.

6. ACKNOWLEDGEMENTS

I would like to thank W. Burton for reading several drafts of this paper. Thanks also goes to the musicians who have used PROS in their work. Their feedback has made PROS a better program. The support of the Computer Systems Research Group at the University of Toronto is gratefully acknowledged. Finally, this work was undertaken as part of the SSEP. This research is funded by the Social Sciences and Humanities Research Council of Canada. This support is gratefully acknowledged.

7. REFERENCES


