AUTOMATIC SPECIES COUNTERPOINT

by

Bill Schottstaedt

Research sponsored by
System Development Foundation

CCRMA
DEPARTMENT OF MUSIC
Stanford University
Stanford, California 94305
AUTOMATIC SPECIES COUNTERPOINT

by

Bill Schottstaedt

Species counterpoint as presented by J. J. Fux in Gradus Ad Parnassum appears to be a ready made case for a rule based "expert system". In programs of this sort, knowledge is encoded as a list of IF..THEN statements. These attributes can easily be defined in such a manner that a computer program can use them to find acceptable solutions to species counterpoint problems. In this paper we present a program that can write counterpoint of this sort. Rather than get bogged down in circumlocutions, we provide the actual code, and the reader can if he so desires re-implement the entire program. The language used is SAIL, an Algol-like language fully described elsewhere. The original implementation was done in Fls, an offshoot of SAIL also described elsewhere. We translated everything to SAIL to speed up execution. This file is itself an executable program which solves species counterpoint problems as described in the text.

This research was supported by the System Development Foundation under Grant SDF #345. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of Stanford University, any agency of the U. S. Government, or of sponsoring foundations.
© Copyright 1984
by
Bill Schottstaedt
## Table of Contents

### Automatic Species Counterpoint
1

### Assumptions
1

### Basic Definitions
2

### Data Representation
5

### Basic Rules
10

### Species Definition
18

### Multi-Part Counterpoint
22

### Searching Methods
25

### Acknowledgments
35
Species Counterpoint

Automatic Species Counterpoint

Species counterpoint as presented by J. J. Fux in "Gradus Ad Parnassum" appears to be a ready made case for a rule based "expert system". In programs of this sort, knowledge is encoded as a list of IF-THEN statements. Obvious examples from Fux are:

IF Melodic-Leap > Octave THEN Try-Something-Else.

or

IF Interval-With-Bass = Fourth AND Species = First-Species THEN Try-Something-Else.

These attributes can easily be defined in such a manner that a computer program can use them to find acceptable solutions to species counterpoint problems. In this paper we present a program that can write counterpoint of this sort. Rather than get bogged down in circumlocutions, we provide the actual code, and the reader can if he so desires re-implement the entire program. The language used is SAIL, an Algol-like language fully described elsewhere. The original implementation was done in Pla, an offshoot of SAIL also described elsewhere. We translated everything to SAIL to speed up execution. This file is itself an executable program which solves species counterpoint problems as described in the text.

Assumptions

The octave is divided into 12 semitones. We can therefore give a unique integer to every possible pitch. In our examples, this value is the distance in semitones from the given pitch to low C (16 Hz).

Rhythms can be represented as a certain number of eighths.

Notes within a voice do not overlap. Each note's onset time can therefore be represented as a certain number of eighths from time 0. Of course a prior assumption is that there are things called voices that can be assigned a unique succession of notes. Each note has a duration, an onset time, and a pitch. If more than one voice exists, there is also a notion that a "vertical" interval exists between these voices equivalent to the melodic interval necessary to jump from one voice to the other.

The cantus firmus and the starting note of each counterpoint voice are specified by hand. We also currently assume that the cantus firmus is entirely in whole notes, but it would not be difficult to remove this restriction.
Basic Definitions

We must first define our terms. These include the interval names, consonance and dissonance classifications, and so on. To define the intervals:

```
BEGIN "tune"
REQUIRE 20000 SYSTEM POL;
DEFINE l="COMMENT";
DEFINE Unison 0;
MinorSecond 1;
MajorSecond 2;
MinorThird 3;
MajorThird 4;
Fourth 5;
Fifth 6;
MinorSixth 8;
MajorSixth 9;
MinorSeventh 10;
MajorSeventh 11;
Octave 12;
```

Each interval is defined by the number of semitones within it. These intervals are impervious to octave interpolations and transpositions.

We classify each interval as either a perfect consonance, imperfect consonance, or dissonance:

```
BOOLEAN ARRAY PerfectConsonance[(s):12];
PerfectConsonance[Unison]=TRUE;
PerfectConsonance[Multip] = TRUE;
PerfectConsonance[Octave]=TRUE;

BOOLEAN ARRAY ImperfectConsonance[(s):12];
ImperfectConsonance[MinorThird]=TRUE;
ImperfectConsonance[MajorThird]=TRUE;
ImperfectConsonance[MinorSixth] = TRUE;
ImperfectConsonance[MajorSixth] = TRUE;

BOOLEAN ARRAY Dissonance[(s):12];
Dissonance[MajorSecond] = TRUE;
Dissonance[MinorSecond] = TRUE;
Dissonance[MinorSeventh] = TRUE;
Dissonance[MajorSeventh] = TRUE;
Dissonance[Fourth] = TRUE;
Dissonance[Tritone] = TRUE;
```

To ascertain whether a given interval is a dissonance, we merely use that interval as an index into the Dissonance array. The other arrays are used similarly. Under certain circumstances, the fourth is considered an imperfect consonance, but we will leave that complication until later. (In SAIL all Boolean variables are initialized to False, so we need only set the ones that should be True. These assignments take place at initialization time before the main program is executed).

We must also define the basic scales used in species counterpoint. The representation chosen here takes advantage of the fact that the scales are only an octave in extent, so octave transpositions can be removed before a decision is made as to whether a given pitch is in a mode. However, we do have to take into account transpositions by some interval other than an octave. In the definition given here, we assume that the transposition (if any) has also been removed - from this code's viewpoint every mode starts on pitch 0 and has only 12 pitches to choose from. Any pitch that is in the mode has a value of 1 in the corresponding mode array. A value of 0 means that the given pitch is illegal (or at least a chromatic alteration) in that mode.
Species Counterpoint

`DEFINE ReAllan1, Dorian2, Phrygian3, Lydian4, Mixolydian5; Ionian6, Lorian7;
PRESET_1,0,1,0,1,0,1,0,1,0,1,0,1; INTEGER ARRAY _Ionian[8:11];
PRESET_1,0,1,0,1,0,1,0,0,1,0,1; INTEGER ARRAY _Dorian[8:11];
PRESET_1,0,0,0,1,0,0,1,0,1,0,1; INTEGER ARRAY _Phrygian[8:11];
PRESET_1,0,0,0,1,1,0,0,1,0,0,1; INTEGER ARRAY _Lydian[8:11];
PRESET_1,0,1,0,1,0,0,0,1,0,0,0; INTEGER ARRAY _Mixolydian(8:11);`

`BOOLEAN PROCEDURE InMode(INTEGER Pitch, NODE); BEGIN
  INTEGER Pit;
  Pit=PITCH[8:12]; i remove any lingering octave information;
  CASE NODE OF
    BEGIN
    (Ionian) RETURN _Ionian(Pit); [etc.]
  END;
END;`

Although defined here, the Ionian and Lorian modes have not actually been used during testing and may contain unnoticed bugs.

Certain intervals are not allowed melodically because they are considered hard to sing:

`BOOLEAN ARRAY Bad旋律.Interval(8:12);`

`SIMPLE PROCEDURE BadInit; BEGIN
  Bad旋律.Interval[Tritone]=TRUE;
  Bad旋律.Interval[MajorSixth]=TRUE;
  Bad旋律.Interval[MinorSeventh]=TRUE;
  Bad旋律.Interval[MajorSeventh]=TRUE;
END; REQUIRE BadInit INITIALIZATION;`

Other "bad" intervals such as the augmented second can be avoided by forbidding chromatic alterations within the mode. The latter are necessary unfortunately at cadences, so more elaborate checks will be made later. We must also avoid a leap greater than an octave and a leap of a minor sixth down. The leap of a minor sixth up is acceptable.

`BOOLEAN PROCEDURE Bad旋律 INTERVAL; RETURN 1LESS(INV) Octave OR
  1 more than an octave either way;
  Bad旋律.Interval[AND (In1) OR 1 tritone or major sixth or seventh;
  Interval-MinorSixth] OR 1 downward minor sixth;
  We will often need to distinguish a "skip" (or leap) from a "step". These are not explicitly defined in my version of Fux, but it is clear that a skip is any interval greater than a major second, and that a step is an interval that is not a skip and also not a unison.

`BOOLEAN PROCEDURE Skip(INTEGER INTERVAL);`

`RETURN (ABS (INTERVAL) > MajorSecond);

BOOLEAN PROCEDURE Step(INTEGER INTERVAL);
RETURN (ABS (INTERVAL) > MinorSecond) OR (ABS (INTERVAL) > MajorSecond));`

The intervals are further divided up into the groups second, third, sixth, seventh, and so on. We therefore define the obvious procedures:
Boolean Procedure AThird

Boolean Procedure ASeventh

Boolean Procedure AOctave

Boolean Procedure AThenth

The last procedure (AThenth) is used to implement only one rather obscure rule.

Now we define the four kinds of "motion": direct, oblique, contrary, and no motion:

```plaintext
<table>
<thead>
<tr>
<th>DirectMotion</th>
<th>ContraryMotion</th>
<th>ObliqueMotion</th>
<th>NoMotion</th>
</tr>
</thead>
</table>
```

Two other procedures will be useful later when we apply rules involving direct motion to a perfect consonance and consecutive leaps:

```plaintext
Boolean Procedure DirectMotion

Boolean Procedure PerfectConsonance
```
Species Counterpoint

```plaintext
BOOLEAN PROCEDURE ConsecutiveSkipsInSameDirection(
    INTEGER Pitch1,Pitch2,Pitch3;
RETURN ((Pitch1>Pitch2 AND (Pitch2>Pitch3)) OR
        (Pitch1<Pitch2 AND (Pitch2<Pitch3)) AND
        NotSkipped(Pitch2-Pitch1) AND ।
        NotSkipped(Pitch3-Pitch2)); ।
```

The extreme high and low pitches in a melody define its overall range. Fux emphasizes on many occasions that this range should be constrained to fit human vocal ranges.

```plaintext
DEFINE HighestSemitones=72;    \ this is a high C (1825 Hz);
DEFINE LowestSemitones=24;     \ this is a low C (65 Hz);

BOOLEAN PROCEDURE OutOfRange(INTEGER Pitch);
RETURN (Pitch>HighestSemitones) OR (Pitch<LowestSemitones));

BOOLEAN PROCEDURE ExtremeRange(INTEGER Pitch);
RETURN(Pitch>HighestSemitones-3) OR Pitch<LowestSemitones+3));
```

These procedures give a rather liberal range check. We also need code to find the overall melody range so that it normally does not exceed an octave and a fifth. This code, however, depends on the representation we choose for voice data.

Data Representation

For the purposes of this description we will use a simple representation which uses parallel arrays for onset time, duration, and pitch. First we have integers for the transposition (BasePitch), the current mode (Mode), and the length of the cantus firmus in eighth notes (TotalTime).

```plaintext
DEFINE MostNotes=128;  \ most notes any one voice can have;
DEFINE MostVoices=8;

INTEGER BasePitch,Mode,TotalTime;
```

We arbitrarily limit the longest melody to 128 notes and the number of voices to 8 (once again, these are purely for array allocation purposes and are not built into any of the counterpoint writing procedures). We need a place to hold the pitch, duration, and onset time data for each voice, including the cantus firmus.

```plaintext
INTEGER ARRAY CtrPt,Onset,Dur[[MostNotes,MostVoices];
INTEGER ARRAY TotalNotes[128,MostVoices];
```

The cantus firmus is voice 0 in this scheme, but is not otherwise distinguished from the other voices. Each note of voice $V$ keeps its note information for note number $N$ in these three parallel arrays. OnSet($V,N$) is the begin time in eighth notes of the note, $Dur(V,N)$ is its duration (in eighth notes), and $CtrPt(V,N)$ is the pitch of the note (in semitones above the low C). Next we need a variety of procedures to access the data in these arrays.
We also need a procedure that returns the total range of the voice \( V \) at the note \( CN \) (counting from the beginning of the melody), given the current pitch \( CP \).

```
INTEGER Procedure TotalRange(INTEGER \( C_n, C_p, V \));
BEGIN
    \( \text{I return total range of melody so far \( \text{including CP} \)} \);
    INTEGER MaxP, MinP, MaxP = \( CP \);
    MinP = \( CP \);
    FOR \( i = 1 \) STEP \( . \) UNTIL \( C_n - 1 \) DO
        \( \text{BEGIN} \)
        \( \text{MaxP = MinP = \_ Uset(v, v)} \); \( \text{MaxP = MaxP = \_ Uset(v, v)} \); \( \text{END} \);
        \( \text{END} \);
        \( \text{Return} \( \text{MaxP - MinP} \); \( \text{END} \);

    \text{We present most of the data analysis procedures in this form. In words, we have a proposed pitch \( CP \) in a voice \( V \) at the point \( CN \) in that voice, and need a decision about the acceptability of that pitch.}

Since we assume for simplicity's sake that the cantus firmus is always in whole notes, it is easy to write a procedure which returns the cantus firmus note at any given eighth note beat \( N \).

```
INTEGER Procedure Canus(INTEGER \( n, v \));
RETURN(Crirp(\( \text{Uset}(n, v) \) DIV 8, 1));

```
A similar procedure is needed to return the current pitch in some other voice (we could use this procedure for the cantus firmus also because it is just voice 0, but as long as we are assuming the cantus moves in whole notes we can handle it separately to save some time):

```
INTEGER Procedure VIndex(INTEGER \( Time, VNumb )\); BEGIN \( i \);
FOR \( i = 1 \) STEP \( 1 \) UNTIL \( \text{TotalNotes(VNumb)} \) DO
    IF \( \text{Uset(i, VNumb) = Time AND Uset(\( i, VNumb + \text{Dur(i, VNumb)} \) = Time THEN} \text{DONE} \);
    RETURN(i); \( \text{END} \);

INTEGER Procedure Other(INTEGER \( C_n, C_p, V)\); RETURN(Crirp(VIndex(\( \text{Uset}(C_n, V) \)), V));

INTEGER Procedure Baseline(INTEGER \( C_n, v \)); BEGIN \( i \);
INTEGER \( j \), LowestPitch;
LowestPitch = Canus(Cn, v);
FOR \( j = 1 \) STEP \( 1 \) UNTIL \( V - 1 \) DO
   LowestPitch = LowestPitch \( \text{Min} \{ \text{Other}(C_n, v) \} \);
RETURN(LowestPitch); \( \text{END} \);
```

```

Data Representation
This is not an optimal search procedure, but normally there are not too many notes in a melody. We can therefore search laboriously from the beginning for the note in voice \( P \) that sounds at the same time as the note \( CN \) in voice \( V \).

We must also define various rhythmic values:

```plaintext
DEFINE WholeNote = 0,
     HalfNote = 4,
     DottedHalfNote = 6,
     QuarterNote = 2,
     EighthNote = 1;
```

A measure contains 8 eighth notes (triple time is not supported).

```plaintext
INTEGER PROCEDURE Beat(INTEGER n);
RETURN (n MOD 8); // first beat, 7=last;
```

A downbeat occurs on the first (0-th) beat of the measure.

```plaintext
BOOLEAN PROCEDURE DownBeat(INTEGER n, v);
RETURN (Beat(n, v) = 7);
```

An upbeat can be considered to be any beat that is not a downbeat. There are several cases where beat 4 (the second half note beat) is a downbeat, but these will be handled separately.

```plaintext
BOOLEAN PROCEDURE UpBeat(INTEGER n, v);
RETURN (NOT DownBeat(n, v));
```

Fux mentions that it is desirable to maintain "variety" in the melodies. This variety seems to entail a mix of melodic intervals coupled with an avoidance of too many repetitions of any one pitch. To check the latter condition we define a procedure which looks for repeated pitches within a melody:

```plaintext
INTEGER PROCEDURE PitchRepeats(INTEGER Cn, Cp, V);
BEGIN
   INTEGER i, k1;
   i = 8;
   FOR k1 = 1 STEP 1 UNTIL Cn-1 DO IF UpBeat(i, V) THEN i = i+1;
   RETURN(i);
END;
```

We also need to encourage the melody to contain a nice mixture of intervals (in third and fifth species). We keep track of how many times a given interval type has been used so far, then check to see if others have been used nearly as many times. The interval type ignores distinctions of major and minor, but does not ignore direction (a rising minor second is considered the same interval type as a rising major second, but not the same as a descending minor second).
DATA REPRESENTATION

1

DEFINE One=0, Two=2, Three=3, Four=4, Five=5, Six=6, Eight=8;

INTEGER PROCEDURE SIZE(INTEGER Neln); BEGIN INTEGER ActInt,IntTyP;
ActInt=ABS(Neln);
CASE ActInt OF
(union) IntTyP=One;
(minorSecond)(majorSecond) IntTyP=Two;
(minorThird)(majorThird) IntTyP=Three;
(fourth) IntTyP=Four;
(fifth) IntTyP=Five;
(minorsixth) IntTyP=Six;
(octave) IntTyP=Eight;
ELSE PRINT("illegal melodic intervals","Neln"); END;
RETURN(If Neln=8 THEN IntTyP ELSE -IntTyP); END;

BOOLEAN PROCEDURE TooManyD(Interval(INTEGER Cn,Cp,v)) BEGIN INTEGER ARRAY Intas[v]; MINCLT(Intas); FOR i=1 STEP 1 UNTIL Cn DO BEGIN k=Size(CharPt(i,v)-CharPt[i-1,v]); Intas[i]=Intas[i-1] k=Size(CharPt(i-1,v)); MINL=MINL+1; FOR i=1 TO 7 STEP 1 UNTIL 8 DO IF i AND Intas[i]=Intas[MINL] THEN MINL=i; RETURN(Intas[i]>Intas[MINL]+1)); END;

Next we must define when a dissonance is legal in each species, and also when chromatic (altered) notes occur. The rules governing the cadential formulas are pretty clear, but the rules governing facts seem to consist mostly of handwaving and exceptions. We first define dissonance handling in each species:

BOOLEAN PROCEDURE Dissonance(Interval,Cn,Cp,v,Species);
IF Species=1 OR Dir(Cn,v)=wholenote THEN RETURN(Dissonance(Interval));

In first species no dissonances are allowed, so we simply return true in every case in which Interval is a dissonance. Similarly, we do not allow wholenotes to form a dissonance (in fifth species).

IF Species=2 THEN IF (Downbeat(Cn,v) OR NOT (NoteUp(Cp-Us(Cn-1,v)))) THEN RETURN(Dissonance(Interval)); ELSE RETURN(FALSE); Passing Tone

In second species, a dissonance is allowed only as a passing tone on an upbeat. Since we don't yet know what the next note will be, we must put off the rest of the dissonance check until later.
Species Counterpoint

ELSE
IF Species=3 THEN
BEGIN
INTEGER MelIn;
IF Beat8(DnSet(Cn,v))=0 OR FirstNote(Cn,v) OR LastNote(Cn,v) THEN Return(Dissonance(Interval));
MelIn=Ca-Uu(Cn-1,v);
IF NOT IsStep(MelIn)) THEN Return(Dissonance(Interval));
CORRECT
0 cannot be dissonant (downbeat) can be if passing either way, but must be approached by step.
1 can be if passing either way, but must be approached by step.
2 can be if passing 2 to 4 (both latter cases)
3 can be if passing but must be approached and left by step
RETURN(FALSE);
END

In third species, the downbeat cannot be a dissonance, but any of the other beats can if they are passing tones or cambiaturas. As with second species, the dissonance resolution is checked later.

ELSE
IF Species=4 THEN
BEGIN
INTEGER MelInt;
IF UpBeat(Cn,v) OR FirstNote(Cn,v) OR LastNote(Cn,v) THEN Return(Dissonance(Interval));
MelInt=Cp-Uu(Cn-1,v);
IF MelInt=0 THEN Return(Dissonance(Interval));
RETURN(FALSE); i.e. unison to downbeat is ok, but needs check later;
END

In fourth species, a dissonance is legal as a suspension. The suspension is handled here as a note repeated from the upbeat to the downbeat (consider it tied across the bar), so the melodic interval is obviously a unison.

ELSE
IF Species=5 THEN
BEGIN
IF Beat8(DnSet(Cn,v))=0 THEN first note;
IF Cp(Uu(Cn-1,v))=0 THEN so can be tied from previous to disson;
THEN se dissonance is ok here (check resolution later);
ELSE Return(FALSE);
ELSE Return(Dissonance(Interval));
ELSE not first beat;
IF NOT IsStep(Cp-Uu(Cn-1,v)) THEN cannot jump to dissonance;
THEN Return(Dissonance(Interval));
RETURN(FALSE);
END

Finally, in fifth species we have passing note dissonances, cambiaturas, and suspensions. Therefore, if the downbeat (beat 0) of the measure is dissonant, the note must have been tied across the bar (the melodic interval must be a unison). On any other beat the dissonance must be approached by step. We will check later for its proper resolution.

Lastly, in multi-part counterpoint we often need to check for various kinds of pitch doublings:
Data Representation

BEGIN
    BOOLEAN Procedure Doubled(INTEGER Pitch, En, v);
    BEGIN
        INTEGER v;v;
        FOR v=0 STEP 1 UNTIL v<1 DO;
            IF (either(En,v,v) MOD 12)=Pitch THEN RETURN(TRUE);
        END;
    END;

Basic Rules

The basic rules of species counterpoint describe how melodies are formed and combined. In most cases any given rule can be broken if other factors necessitate it. Fux repeatedly presents the rules as guidelines, not absolutes. In our implementation we define the relative importance of the rules by assigning each rule a penalty. The higher the penalty, the worse it is to break the associated rule. The relations between these penalties determine what kind of solutions the program will find.

Our penalty types and the associated penalty values are:

```plaintext
DEFINE Infinity:=情况;
DEFINE Bad=100;
DEFINE RealBad=200;
DEFINE UnisonPenalty:=bad1;
DEFINE DirectToFifthPenalty:=BadBed;
DEFINE DirectToOctavePenalty:=RealBad;
DEFINE ParallelFifthPenalty:=Infinity;
DEFINE ParallelUnisonPenalty:=Infinity;
DEFINE EndOfPerfectPenalty:=Infinity;
DEFINE LeadingTonePenalty:=Infinity;
DEFINE DissonancePenalty:=Infinity;
DEFINE OutOfRangePenalty:=realBad;
DEFINE OutOfScalePenalty:=Infinity;
DEFINE TwoConsecutivePenalty:=1;
DEFINE DirectionalPenalty:=1;
DEFINE PerfectConsonancePenalty:=2;
DEFINE CompoundPenalty:=1;
DEFINE TenthToOctavePenalty:=8;
DEFINE SkipToOctavePenalty:=8;
DEFINE SkipFromUnisonPenalty:=4;
DEFINE SkipPrecededBySameDirectionPenalty:=4;
DEFINE FifthPrecededBySameDirectionPenalty:=3;
DEFINE SixthPrecededBySameDirectionPenalty:=8;
DEFINE FifthFollowedBySameDirectionPenalty:=3;
DEFINE SixthFollowedBySameDirectionPenalty:=4;
DEFINE TwoSkipNotesInTriadPenalty:=3;
DEFINE BadTonicPenalty:=Infinity;
DEFINE LydianConsonancePenalty:=5;
DEFINE UpperNeighboringPenalty:=6;
DEFINE LowerNeighboringPenalty:=6;
DEFINE OverTenthPenalty:=Infinity;
DEFINE OverOctavePenalty:=bad1;
DEFINE SixthLeapPenalty:=2;
DEFINE OctaveLeapPenalty:=5;
DEFINE BasCadencePenalty:=Infinity;
DEFINE DirectPerfectOctobassPenalty:=Infinity;
DEFINE RepeatedOctobassPenalty:=bad1;
DEFINE DissonanceNotFittingThirdPenalty:=Infinity;
DEFINE BassOctobassPenalty:=5;
DEFINE TwoRepeatedNotesPenalty:=2;
DEFINE ThreeRepeatedNotesPenalty:=4;
DEFINE FourRepeatedNotesPenalty:=7;
```
<table>
<thead>
<tr>
<th>Penalty</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>LeapCadencePenalty</td>
<td>+13</td>
</tr>
<tr>
<td>NoteCombinationPenalty</td>
<td>infinity</td>
</tr>
<tr>
<td>NoteSeparationPenalty</td>
<td>8</td>
</tr>
<tr>
<td>UnionSeparationPenalty</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>NotePersistencePenalty</td>
<td>infinity</td>
</tr>
<tr>
<td>NoteToneRhythmPenalty</td>
<td>infinity</td>
</tr>
<tr>
<td>EighthJumpPenalty</td>
<td>m16</td>
</tr>
<tr>
<td>HalftonePenalty</td>
<td>+13</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>UnionGroupPenalty</td>
<td>21</td>
</tr>
<tr>
<td>MelodicRhythmPenalty</td>
<td>+2</td>
</tr>
<tr>
<td>SkipToDownbeatPenalty</td>
<td>+1</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>ThreeStepPenalty</td>
<td>+2</td>
</tr>
<tr>
<td>DownbeatUnionPenalty</td>
<td>m16</td>
</tr>
<tr>
<td>VerticalTritonePenalty</td>
<td>+2</td>
</tr>
<tr>
<td>MelodicTritonePenalty</td>
<td>4</td>
</tr>
<tr>
<td>AscendingSixthPenalty</td>
<td>+1</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>RepeatedPitchPenalty</td>
<td>+1</td>
</tr>
<tr>
<td>NoteContraryToOtherPenalty</td>
<td>+1</td>
</tr>
<tr>
<td>NoteTriadPenalty</td>
<td>34</td>
</tr>
<tr>
<td>InnerVoiceInDirectPerfect</td>
<td>+21</td>
</tr>
<tr>
<td>InnerVoiceInDirectTritone</td>
<td>+13</td>
</tr>
<tr>
<td>SexFiveChordPenalty</td>
<td>infinity</td>
</tr>
</tbody>
</table>

These penalty values have come partly from Fux's comments (the more important rules get a higher penalty), and partly from experience running the program. Obviously not all the penalties apply to every species of counterpoint. The penalties are all positive (that is, there are no rewards), because we want to be able to abandon a line as soon as its penalty gets too high. This implies that the penalty function should never descend. Even the smaller penalties have a profound effect on the outcome of the counterpoint search. The solver is consistently "bent" in the direction determined by the penalties causing the music to reflect the slightest changes in that direction.

We need a procedure which examines the current note of the current voice and returns a measure of its goodness (the lower the penalty, the better the note).

```
FORWARD INTEGER PROCEDURE SpecialSpeciesCheck(
    INTEGER Cn, Cv, v, Other[0], Other1, Other2, 
    NumParts, Species, MelInt, Interval, LastInt, LastIntClass, Pitch, LastMelInt); 
```

```
FORWARD INTEGER PROCEDURE OtherVoiceCheck( 
    INTEGER Cn, Cv, v, Species); 
```

We define the special attributes of each species later. These are checked by the SpecialSpeciesCheck procedure (declared here for convenience, but defined later to make the presentation more clear).
The rules are applied here in an order which hopes to reduce redundant checks of various kinds. At the start we apply rules that apply to every pitch in the melodies.

1. melody must stay in range;
   IF OutOfRange(Cp,BasePitch) THEN ValVal=OutOfRangePenalty;

2. extremes of range are also bad to be avoided;
   IF ExtremeRange(Cp,BasePitch) THEN ValVal=ExtremeRangePenalty;

3. Chromatically altered notes are accepted only at cadences. Other alterations (such as flato) will be handled later;
   IF NOT NextToLastNote(Cn,v) THEN
       BEGIN
       IF (Species=2) THEN ValVal=OutOfRangePenalty
       ELSE
       IF (vTotNotes(v-2) OR (Mode=4) OR (Cp=Other8) OR (Interval=1) OR (Interval=5)) THEN
       IF NOT InNode(Pitch,Node) THEN ValVal=OutOfRangePenalty
       ELSE
       ValVal=OutOfRangePenalty;
   END

4. The leading tone cannot be doubled, nor should the raised leading tone be preceded by its unraised form. The next to last chord of the counterpoint must have a leading tone somewhere.
Species Counterpoint

IF UnrealLength>>Tone THEN
  IF DoublePitch,Cn,v) = leading tone doubled THEN Val++ValDoubleLeadingTonePenalty
  ELSE IF Pitch=19 THEN Val++ValBadCadencePenalty
  ELSE IF LastNode(Pitch,Node) = last note THEN Val++Val=10thNodePenalty
  ELSE IF NumParts = last voice THEN Val++Val=10thVoicePenalty
  THEN IF NotDouble(11,Cn,v) AND NotDouble(12,Cn,v) THEN Val++Val=10thLeadingTonePenalty

IF Val=Infinity THEN RETURN(val)

IF Cn=2 THEN
  BEGIN
    LastIntInt(-1,(Cn-1,v)=LastIntInt(-2,v)
    LastDir=(LastIntIntLastIntInt)
  END;

IF (NumLastInt) = LastIntInt(intClass,(Cn-1,v)=Other1) MOD 12 THEN
  dissonance must be handled correctly. The procedure
  Resonance returns true if a CP forms a dissonance that is
  not correctly prepared. Other rules later check that it is
  resolved correctly.
  IF Resonance(intClass,Cp,v,Species) THEN Val++Val=DissonancePenalty

IF Val=Infinity THEN RETURN(val)

Val++Val=SpeciesCheck(Cn,Cp,v,Other1,Other2,NumParts,Species,LastInt,Interval,intClass,LastIntInt,Species,LastIntInt)

IF Val=Infinity THEN RETURN(val)

This procedure calls at the current state from the point of view of the current
species. There are a number of rules unique to each species.

IF FirstNote(Cn,v) THEN RETURN(val)
  IF no further rules apply to first note THEN Val=Val=Infinity THEN RETURN(val)

The "fundamental rule" defines which intervallic relationships are acceptable between
two melodies. Fux presents this as four rules, but as Martini remarks, these four are
actually reducible to one -- voices cannot move by direct motion to a perfect consonance.
In multi-voice counterpoint, this rule can be broken, especially at cadences.

IF DirectionToPerfectConsonance(Cn,Cp,Other1,Other2) THEN
  IF we do have direct motion to perfect
  THEN IF intClass1 = reached an octave or unison THEN Val++Val=DirectionToOctavePenalty
  ELSE Val++Val=DirectionToFifthPenalty

The expression "NumParts=1" checks that we are indeed in a multi-voice situation. In
two part counterpoint we do not allow direct motion to a perfect consonance at all.
Basic Rules

; check for more blatant examples of the same error;
IF IntClass=fifth AND LastIntClass=fifth
THEN Val=Val+ParallelFifthPenalty;

IF IntClass=unison AND LastIntClass=unison
THEN Val=Val+ParallelUnisonPenalty;
   ; voices parallel in octave or fifths;
   ; which Fux seems to consider to be worse;
   ; I than just direct motion to these;
   ; IF Val=Val+Infinite THEN RETURN(val);

; certain melodic intervals are disallowed;
IF BadMelody(ValInt) THEN Val=Val+BadMelodyPenalty;
   ; IF Val=Val+Infinite THEN RETURN(val);

This rule (melodic interval check) is actually built into the search procedures
given below so its inclusion here is mostly for completeness.

; must end on unison or octave in two parts, fifth and major third
; allowed in 3 and 4 part writing;
IF LastNote(Cn,v) AND
   ; it is the last note;
   ; IntClass=unison
   ; must be octave or unison here (2 parts);
THEN
   IF NumParts=1 OR Interval=1 two part writing or bass;
   THEN Val=Val+EndOnPerfectPenalty
   ELSE
      ; NumParts=1 so we must have 3 or more parts writing;
      IF IntClass=fifth AND IntClass=MajorThird
         THEN Val=Val+EndOnPerfectPenalty
      END

If we end on a major third, this should of course be a major third above the cantus,
not below it (similarly for the fifth).

; penalize direct motion any kind (contrary motion is better);
IF MotionType=Jump(Cn-1,v),Cp,Other1,Other2+Direction
THEN
   BEGIN
      Val=Val+DirectionMotionPenalty;
      IF IntClass=Tritone
         THEN Val=Val+DirectToFifthPenalty
      END
   ; penalize compound intervals (else position is favored);
   IF ABS(Interval)>Octave
      ; I avoid compound intervals wherever possible;
      THEN Val=Val+CompoundPenalty
   END

Now we present a raft of rules involving skips. For example, it is considered bad to
have too many consecutive skips in the same directions.

; penalize consecutive skips in the same direction;
IF (Cn) AND ConsecutiveSkipsInSameDirection(Cn,Cn-1,v),Cp(Cn-1,v),Cp
THEN
   BEGIN
      INTEGER totalJump;
      Val=Val+TwoSkipsPenalty;
      totalJump=ABS(Cp-Int(Cn-1,v));
      IF totalJump>Octave
         THEN Val=Val+TwoSkipsOnIntervalPenalty
      END
   ; penalize a skip to an octave;
   IF ((IntClass=unison) OR (Skip(ValInt) OR Skip(Other1-Other2)))
      THEN Val=Val+SkipToOctavePenalty
   IF (SkipsFromInterval) AND (Skips(ValInt))
      THEN Val=Val+SkipFromIntervalPenalty

I do not skip from a unison (not a very important rule);
IF (Other1=Int(Cn-1,v)) AND Skips(ValInt)
THEN Val=Val+SkipFromUnisonPenalty
1. penalize skips followed or preceded by motion in same direction;
   IF Cn2 AND Rsip(MelInt) AND SameDir
     THEN 1 skip preceded by same direction;
   BEGIN

2. especially penalize fifths, sixths, and octaves of this sort;
   IF Rsip(MelInt) = Fifth
     THEN Val=Val+SkipPrecededBySameDirectionPenalty
   ELSE
     IF Rsip(MelInt) = Fifth OR Rsip(MelInt) = Octave
       THEN Val=Val+SkipFollowedBySameDirectionPenalty
     ELSE Val=Val+SixthFollowedBySameDirectionPenalty
   END

3. IF Cn2 AND Rsip(LeadInt) AND SameDir
   THEN
     BEGIN
     IF Rsip(LeadInt) = Fifth
       THEN Val=Val+SkipFollowedBySameDirectionPenalty
     ELSE
       IF Rsip(LeadInt) = Fifth OR Rsip(LeadInt) = Octave
         THEN Val=Val+SixthFollowedBySameDirectionPenalty
       ELSE Val=Val+SixthFollowedBySameDirectionPenalty
     END;

4. too many skips in a row — favor a mix of steps and skips;
   IF Cn4 AND Rsip(MelInt) AND Rsip(LeadInt) AND Rsip(Uv(Cn-2,v) = Uv(Cn-3,v))
     THEN Val=Val+MelodicBoredPenalty;

5. avoid tritones melodically;
   IF Cn4 AND
     OBSC(=Uv(Cn-2,v))=Tritone OR
     ASS(C=Uv(Cn-3,v))=Tritone OR
     ASS(C=Uv(Cn-2,v))=Tritone
     THEN Val=Val+MelodicTritonePenalty;

As can be seen from the solutions given at the end of the paper, tritone handling is far from perfect. Perhaps the MelodicTritone penalty should be higher, and more elaborate checks should be added for tritones over larger melodic distances.

6. I do not allow movement from a tenth to an octave by contrary motion;
   IF Species6 AND BndPartials
     THEN
     IF 10 = Int(Other1-Int(Uv(Cn-1,v)) AND Int(Octave(Interval))
       THEN Val=Val+TenthToOctavePenalty;

Fux admits this rule (disallowing motion from a tenth to an octave) is without obvious musical justification, but states that the rule merely follows the practice of the great composers.

7. I made range checks — did we go over an octave recently;
   IF Cn2 AND ASS(Uv(Cn-2,v)) = Octave THEN Val=Val+OctavePenalty;
   I same for a tenth;
   IF Cn3 AND TotalRange(Cn,Cn,v) = Octave=10 THEN Val=Val+TenthPenalty;
   IF Val=Val+TenthToOctavePenalty;

8. slightly penalize repeated notes;
   IF Cn2 AND (Uv(Uv=Cn-2,v)) AND Uv(Uv=Cn-1,v) = Uv(Cn-3,v)
     THEN Val=Val+TwoRepeatedNotesPenalty;
   IF Cn5 AND (Uv(Cn-2,v)) AND
     Uv(Uv=Cn-1,v) = Uv(Cn-4,v) AND Uv(Uv=Cn-2,v) = Uv(Cn-5,v)
     THEN Val=Val+ThreeRepeatedNotesPenalty;
   IF Cn6 AND (Uv(Cn-3,v)) AND
     Uv(Uv=Cn-1,v) = Uv(Cn-5,v) AND Uv(Uv=Cn-2,v) = Uv(Cn-6,v)
     THEN Val=Val+ThreeRepeatedNotesPenalty;
   IF Cn7 AND (Uv(Cn-4,v)) AND
     Uv(Uv=Cn-1,v) = Uv(Cn-6,v) AND Uv(Uv=Cn-2,v) = Uv(Cn-7,v)
     THEN Val=Val+ThreeRepeatedNotesPenalty;
   IF Cn8 AND (Uv(Cn-5,v)) AND
     Uv(Uv=Cn-1,v) = Uv(Cn-7,v) AND Uv(Uv=Cn-2,v) = Uv(Cn-8,v)
     THEN Val=Val+FourRepeatedNotesPenalty;

9. Val=Val+FiveRepeatedNotesPenalty;

10. Val=Val+SixRepeatedNotesPenalty;

11. Val=Val+SevenRepeatedNotesPenalty;

12. Val=Val+EightRepeatedNotesPenalty;
These rules do not penalize sequences, nor do they catch widely separated pattern repetitions. We may add such checks to a later version of the program.

IF LastNote(Cn,v)
THEN
BEGIN
INTEGER LastPitch;
LastPitch=60(Cn-1,v) AND 12;
IF (LastPitch=11) OR (LastPitch=10 AND (NodePhrygian)) AND (Pitch>0)
THEN Val=Val+UnresolvedLeadingTonePenalty;
END;

Without this rule forcing the leading tone to go to the tonic, in multi-voice situations the program sometimes skips from the leading tone to some other note than the tonic to avoid direct motion to an octave.

IF ValInfinity THEN RETURN(val);
IF AnOctave(MelInt) THEN Val=Val+OctaveLeapPenalty;

IF HalfInt<0.5 AND (MelInt-IntClause) THEN Val=Val+PerfectConsonancePenalty;

no unisons allowed within counterpoint unless more than 2 parts;
IF NumParts<2 AND (IntervalUnion) THEN Val=Val+UnionPenalty;

IF ValInfinity THEN RETURN(val);

I seek variety by avoiding pitch repetitions;
Val=Val+PitchRepeats(Cn,v);v/2;

Without a few rules governing melodic practices, the counterpoint writer is as happy to bounce around by octaves as by steps. This can lead to some rather funny looking melodies.

I penalize octave leaps a little;
IF AnOctave(MelInt) THEN Val=Val+OctaveLeapPenalty;

I similarly for minor sixth leaps;
IF HalfInt<0.5 AND (MelInt>IntClause) THEN Val=Val+SixthLeapPenalty;

I penalize upper neighbor notes slightly (also lower neighbors);
IF C_odd
d(MelInt) AND
C_up(MelInt) AND
C_down(MelInt) AND
C_down2(MelInt) AND
THEN Val=Val+UpperNeighrPenalty;

I penalize lower neighbor notes slightly (also upper neighbors);
IF C_odd
d(MelInt) AND
C_down(MelInt) AND
C_up(MelInt) AND
C_down2(MelInt) AND
THEN Val=Val+LowerNeighrPenalty;

I do not allow normal leading tone to precede raised leading tone;
I also check here for augmented fifths and diminished fourths;
IF NOT (IntClause(MelInt) AND
(MelInt<MinorSecond) OR
(MelInt<MajorThird) OR
THEN Val=Val+AugFifthPenalty;

I slightly from upon leap back in the opposite direction;
IF C_odd
d(MelInt) AND (MelInt)=DifferentDir
THEN
BEGIN
Val=Val=0 MXX (MelInt)=MinIntLastInt-3;
IF C_odd AND (MelInt)=C_odd-2 (MelInt)=C_odd-3
THEN Val=Val+ThreeStepsPenalty;
END;

This rule only penalizes a leap of more than a major third followed immediately by a leap in the opposite direction. Fux has an example that leaps up an octave, then immediately down an octave, so this is not a terrible flaw. More "common sense" checks of this sort are given below.
Species Counterpoint

I try to approach cadential passages by step;
IF NumParts=1 AND ChnTotalNotes/4 AND (@$($MelInt)>4
THEN Val=Val+LeapCadentialPenalty;

I check for entangled voices;
Cross=0;
IF NumParts=1 THEN
FOR k=4 STEP 1 UNTIL Chn DO
IF (%(Chn,k)+-Cantus(x,-v))OR(%(Chn,k)-Cantus(x-1,v))<8
THEN Cross=Cross+1;
IF Cross=0 THEN Val=Val+0 MRK (Cross=2)x3;

This formula "(Cross-2)x3" does not penalize the melody until there are three crossings.

I don’t repeat note on upbeat;
IF (Upbeat(Chn,v) AND @MelInt(Union))
THEN Val=Val+RepetitionOnUpbeatPenalty;

I avoid tritones near Lydian cadence;
IF (MelInt4(Chn,v) AND @ChnTotalNotes/4 AND (Pitch))
THEN Val=Val+LydianCadentialTritonePenalty;

I avoid miscellaneous checks. More elaborate dissonance resolution and
cadential formula checks will be given under "Species definition";
IF Species=1 AND Downbeat(Chn,v)
THEN
BEGIN
IF Species=1 THEN
BEGIN
IF @MelInt(Union) AND 1 don’t repeat upbeat note on downbeat;
NOT LastNote(Chn,v)
THEN Val=Val+UnisonDownbeatPenalty;

I check for dissonance that doesn’t fill a third as a passing tone;
IF (Dissonance(Chn,LastNote(v)) AND
1 NOT Rise(Stairline)) OR (NOT SameDir)
THEN Val=Val+DissonanceNotFillingThirdPenalty;
END;
I check for Direct 8va or 8vb where the intervening interval is less than a fourth;
IF DirectNoteToPerfectIntonance(Up(Chn-2,v),Chn,Other2,Other9) AND
(@$($MelInt)<4)
THEN Val=Val+DirectPerfectOnDownbeatPenalty;
END;
I check for tritone with cantus or bass;
IF InitTritone THEN Val=Val+VerticalTritonePenalty;

I check for melodic interval variety;
IF ChnId AND TouchInterval(Chn,Chn,v)
THEN Val=MelodicSedimentPenalty;

RETURN(Val);
END;

We have not yet provided rules guiding the program toward good overall melodic shapes. It may be useful in this regard to treat a melody as a waveform and look
for its low frequency components. We then presumably want the various melodies to
have different such components or at least different initial phases.
Species Definition

Species Definition

There are differences between the species that can most easily be handled by providing a special set of rules for each species. This procedure is called in Check (given above). The two major special areas are cadential formulas and dissonance handling.

\[
\text{INTEGER PROCSPECIESCHECK}
\begin{align*}
\text{INTEGER } & \text{Cn, Cn}, \text{v, Other1, Other2, NumParts,} \\
& \text{Species, IntVal, Interval, LastIntVal, LastIntClass, Pitch, LastMelInt;}
\end{align*}
\]

BEGIN "SPEC"

\[
\text{INTEGER Val;}
\]

\[
\text{IF Species=1 THEN RETURN(0);}
\]

\[
\text{Val:=0;}
\]

\[
\text{IF species rules for last species;}
\]

\[
\text{accumulated penalty;}
\]

In first species all vertical intervals must be consonances (this is handled by the procedure \textit{Adissonance} given above), the first and last intervals must be perfect consonances, imperfect consonance are better than perfect, and the next to last interval must be a major sixth if the cantus firmus is below or a minor third if the cantus firmus is above. All these rules are handled by Check, so we have no further rules to apply in first species.

In second species the interval at the downbeat must be consonant, but the upbeat can be dissonant if it fills in a third (a "passing tone"). The next to last measure has a fifth to a major sixth if the counterpoint is above, and a fifth to a minor third if it is below. The phrygian cadence is also special. Direct motion to a perfect consonance between successive down beats is accepted if the intervening interval is larger than a major third. \textit{Adissonance} handles the passing tone check, so we need only define the cadences here.

\[
\begin{align*}
& \text{IF Species=2} \\
& \text{BEGIN "2"} \\
& \text{IF NextToLastNote(Cn,v) AND (Pitch=11 OR Pitch=10) AND} \\
& \text{we have the leading tone;} \\
& \text{BEGIN (Mode=Phrygian) OR Interval=I2) IN} \\
& \text{BEGIN} \\
& \text{LastIntClass=6th} \\
& \text{THEN Val=BadCadencePenalty;} \\
& \text{ELSE} \\
& \text{LastIntClass=Minor Sixth} \\
& \text{THEN Val=BadCadencePenalty;} \\
& \text{END} \\
& \text{ELSE "2"}
\end{align*}
\]

Fourth species introduces suspensions and accented dissonances. The dissonance must be resolved by a step downward. Since there are more similarities between third and fifth species and second and fourth, we can save some code by combining third and fifth below.

\[
\begin{align*}
& \text{BEGIN "5 & 6"} \\
& \text{3rd, 4th, and 5th need more info;} \\
& \text{INTEGER k;} \\
& \text{IF Species=4} \\
& \text{THEN} \\
& \text{BEGIN "4"} \\
& \text{IF Downbeat(Cn,v) AND LastIntClass=1} \\
& \text{strongly encourage ligatures;} \\
& \text{THEN Val=BadLigaturePenalty;} \\
& \text{Fux says we should try to use a ligature wherever possible, so we penalize anything else.}
\end{align*}
\]
Species Counterpoint

IF Upbeat(Cn,v) AND Dissonance(1stInterval)
THEN BEGIN
  IF (LastInterval-minorSecond OR LastInterval-majorSecond)
  THEN Val=Val+UnresolvedLigaturePenalty;
END;

If the last interval was a downbeat and a dissonance, the dissonance must be resolved on this beat by a step downward.

IF ActInt=Unison OR (Interval=8 OR (Unis(V(Cn-2,v)-Other2) MOD 12)=Unison)
THEN Val=Val+NoTimeForUnisLigaturePenalty;

The vertical interval cannot resolve to a unison or octave if the preceding downbeat was also a unison or octave (the ligature does not make direct motion to a unison or octave acceptable, but does enable one to move from one fifth to another). It is also bad to resolve to an octave if the cantus firmus is above the melody.

IF ActInt=fifth OR ActInt=Tritone
THEN Val=Val+NoTimeForUnisLigaturePenalty;

And, finally, the resolution must be to a consonance (not a tritone for example).

END "4"

ELSE

In third species in addition to the passing tone dissonance (which can occur on any beat except the first), we must also accept cambiatas. The result is that any dissonance must be approached by step. If the dissonance occurs on the second beat and is approached from above and is left by a third down, then it must be left by two steps up. In the latter case every interval must be consonant except the second (and the fourth if it is a passing tone). In addition, this particular dissonance (the cambiata) can only be a seventh if the cantus firmus is below, or a fourth if it is above (in two parts).

BEGIN "3.5"
INTEGER Above, Cross;
Above=Interval(28);

I added check to stop optimizer from changing 4th beat passing tones into repeated notes/]
IF (Best8(1stInterval(Cn, v)>8 OR Best8(1stInterval(Cn, v)>7) AND Cross){Cn-1, v)
THEN Val=Val+UnisOnBeat4Penalty;


Species Definition

I skipped down beat seems not so great:

IF Best8Onset(Cn,v) = 0 THEN
BEGIN
IF Rskip(MelInt) THEN Val = Val + SkipToDownbeatPenalty;
IF Cn=2 AND MelIntUnion OR MelIntFourth THEN
BEGIN
I look for parallel 5ve or 5 on downbeat;
END;
ELSE I = Cn=2;
IF ABS(Ui,v) - Base(i,v) AND 12 = MelInt THEN
Val = Val + DownbeatUnisonPenalty;
END;
END;
I check for cadence not resolved correctly on 4th beat:
IF Best8Onset(Cn,v) = 0 AND
(Rthird(MelInt) OR
Dissonance(MelInt) OR
(Other2 AND 12) AND
(OR(MelInt, MajorSecond) OR
RBS(MelInt, MinorSecond))) THEN
Val = Val + CadencePenalty;
IF Species = 3 AND
Cn=1 AND
Dissonance(LastIntClass) THEN
BEGIN
I cadence or passing tones?
BEGIN
CASE Best8Onset(Cn,v) OF
BEGIN
(0) IF NOT Rsnap(MelInt) OR
NOT Rsnap(LastInt) OR
(MelInt = LastInt) THEN
Val = Val + DissonancePenalty;
(1) IF NOT Rsnap(MelInt) OR
NOT Rsnap(LastInt) OR
(MelInt = LastInt) THEN
I can't happen (I can't be dissonant);
I cadence or passing tone are alike in this:
I step if passing, third if cadence;
I neither being a unison;
I both continue same direction;
I so less;
I now check more cases;
ELSE
IF NOT Rsnap(MelInt) THEN
I passing tone or (see ELSE clause on);
THEN
BEGIN
IF Above THEN
BEGIN
IF NOT RS7th(LastIntClass) THEN Val = Val + DissonancePenalty;
END;
ELSE
IF LastInt = Fourth THEN
Val = Val + DissonancePenalty;
END;
END;
Species Counterpoint

```plaintext
IF Species=5
THEN
BEGIN "5"
INTEGER LastDisInt;
IF Cn=1 AND Beat(Donest(Cn,v),v)=8 AND Coord((Cn-1,v)) AND (Dur(Cn,v)\$Dur(Cn-1,v))
THEN Val=Val+DissonanceLigationPenalty(3);
IF Cn=3 AND
Dur(Cn,v)=halfnote AND
Beat(Donest(Cn,v),v)=4 AND
(Dur(Cn-1,v)=quarternote AND
Dur(Cn-2,v)=quarternote)
THEN Val=Val+DissonancePenalty;
IF Dur(Cn,v)=EighthNote AND (Coord(Cn,v)) AND Dissonance(NotInt)
THEN Val=Val+DissonancePenalty;
IF Cn=1 THEN LastDisInt+ABS(Gc(Cn-1,v)\$Other1) MOD 12;
IF Cn=1 AND
Dissonance(LastDisInt)
THEN
BEGIN "Dis"
CASE Beat(Donest(Cn-1,v)) OF
BEGIN
(0) (4)
IF LastDisInt=Fourth AND
(NotInt)INUnion AND
(Terd8-Other)\$union AND
Beat(Donest(Cn,v),v)=8
THEN
ELSE
IF (NOT ASstep(NotInt)) OR
(NotASstep(NotLastInt)) OR
((NotLastInt)\$NotInt) OR
(Dur(Cn-1,v)=eighthnote OR
(Dur(Cn-2,v)=quarternote AND Dur(Cn-3,v)=halfNote)
THEN Val=Val+DissonancePenalty;
(1) (3) (5) (7)
IF (NOT ASstep(NotInt)) OR
(NotASstep(NotLastInt)) OR
((NotLastInt)\$NotInt)
THEN Val=Val+DissonancePenalty;
(8)
BEGIN
IF (Dur(Cn-2,v)=eighthnote OR
(Dur(Cn-2,v)=halfNote))
THEN Val=Val+DissonancePenalty;
IF (NotInt)\$minorSecond AND NotInt\$majorSecond)
THEN Val=Val+DissonanceLigationPenalty;
IF NotInt\$Fourth OR ActInt\$Fretone
THEN Val=Val+DissonancePenalty;
IF NotInt\$Fifth AND Int\$octave
THEN Val=Val+DissonanceLigationPenalty;
IF ActInt\$AND
Coord(Cn-2,v)\$Other2) MOD 12)=8
THEN Val=Val+DissonancePenalty;
IF LastInt\$Union THEN Val=Val+DissonancePenalty;
END;
```
It is perhaps interesting that we do not need to add any special encouragement for cambiata - they occur as a side effect of the rules (once the dissonance handling is defined as acceptable). The passing tone (second note of the cambiata) is more likely to jump to an imperfect consonance (the leap down by a third making a sixth with the cantus than to a perfect consonance (the latter is slightly penalized), so we get cambiata simply as a result of the preference for imperfect consonances.

Multi-Part Counterpoint

Each added voice must obey all the normal rules with regard to the bass voice, but need not be quite so particular in relation to other voices. We assume here that the bass voice is either the cantus firmus or the first of the voices calculated by the Search mechanism. The extra rules for multi-voice counterpoint deal mainly with the makeup of chords.

```plaintext
INTEGER ARRAY IntervalWithBase(100); 1 = octave, 2 = step, 3 = third, 4 = fourth, 5 = fifth, 6 = sixth, 7 = seventh
```

We need to look at the chord currently formed by the counterpoint and decide...
whether the pitch doublings are acceptable. In general, the leading tone should not be doubled, the fifth is rarely doubled, the third and sixth (figuring from the bass) can be doubled, but rarely tripled, and the octave can occur as often as necessary. A full chord (octave, third, and fifth for example), is better than a partial one (octave, octave and third for example). In any case, the chord must contain at least one imperfect consonance. A root position triad is better than a first inversion triad (octave, third, and fifth is better than octave, third and sixth). Dissonances between upper voices are acceptable as long as the voices are individually consonant with the bass, but the 6-5 chord is a special case. In this chord the upper voices contain both the fifth and sixth above the bass, and according to some sources, the fifth should be treated as a form of suspension. To simplify matters we allow the 6-5 chord only in fifth species.

\[ \text{PROCEDURE AddInterval(INTEGER n);} \]
\[ \text{BEGIN} \]
\[ \text{INTEGER ActInt;} \]
\[ \text{ActInt=-(3\times\text{ActInt}(\mod 12)) \times (4,2,3,4,5,6,7,8,9,10,11,12);} \]
\[ \text{IntervalsWithBase(ActInt)-IntervalsWithBase(ActInt)+1;} \]
\[ \text{END;} \]

After each voice has checked that its proposed pitch agrees with the basic rules, it must also check for agreement with the other voices currently active. The following procedure handles this:

\[ \text{INTEGER PROCEDURE OtherVoiceCheck(INTEGER Cn,CP,\nu,\text{NumParts},\text{Species});} \]
\[ \text{BEGIN} \]
\[ \text{INTEGER Val,ActCas,ActInt,Origin,Int,ActPitch,BaseLastCP;} \]
\[ \text{BOOLEAN AllSkip;} \]
\[ \text{IF Val=0 \text{ THEN RETURN(0)};} \]
\[ \text{two part or bass voice, so nothing to check;} \]
\[ \text{BEGIN Return(Val+1);} \]
\[ \text{Val=0;} \]
\[ \text{ActCas=BaseLastCP(Cn,\nu);} \]
\[ \text{END;} \]

Since we assume that either the first voice or the cantus firmus is the bass, we cannot allow other voices to cross below the current bass and thereby render invalid all the previous calculations of voice leading and chord type.

\[ \text{IF Cn<ActCas} \]
\[ \text{THEN Val=Val+CrossBelowBasePenalty;} \]

We must also ensure that the raised leading tone in the bass (if present) does not confuse the consonance checker into thinking a diminished fourth is a major third.

\[ \text{Intervals(ActCas(\mod 12));} \]
\[ \text{IF IntervalsMajorThird \text{ AND NOT InMode(ActCas,Mode)} \text{ THEN Val=Val+AugmentedIntervalPenalty;} \]
\[ \text{ActPitch=ActCas(\mod 12);} \]
\[ \text{IF Val=Infinity \text{ OR (\text{NumParts} AND Dissonances(ActCas)) \text{ THEN RETURN(Val);} \}
\[ \text{logic here is that only the last part can be non-lit species} \]
\[ \text{and any therefore have various dissonances that don't want to be} \]
\[ \text{calculated as chord tones;} \]
\[ \text{AllSkip=Skip(ActCas-Cn(\mod 12));} \]
\[ \text{AddInterval(ActCas);} \]
\[ \text{LastCn=ActCas(\mod 12);} \]
\[ \text{FOR i=0 \text{ STEP 1 \text{ UNTIL } i<\text{Val}}} \]
\[ \text{BEGIN} \]
\[ \text{check our pitch against each other voice;} \]
\[ \text{OtherCn=Other(Cn,\nu,k);} \]
\[ \text{OtherCn=Other(Cn,-1,\nu,k);} \]
\[ \text{IF NOT AllSkip(\text{OtherCn-Other}) \text{ THEN AllSkip=FALSE;} \]
\[ \text{AddInterval(OtherCn-Cas);} \]

\[ \text{END;} \]
Multi-Part Counterpoint

I avoid unison with other voices;
IF NOT LastNote(Cn,v) AND Other6=cp THEN Val=Val+UnisonPenalty;

I keep upper voices closer together than lower;
IF Other6=Cn4Bass AND ABS(Cp-Other6)2OctaveAndFifth
THEN Val=Val+UpperVoiceLumpFifthPenalty;

I check for direct motion to perfect consonance between two voices;
Int8-ABS(Int8-Other6) MOD 12;
Int8-ABS(Int8-LastCp) MOD 12;
IF Int8=Int8
THEN
IF Int8Unison THEN Val=Val+ParallelUnisonPenalty ELSE
IF Int8Semitone THEN Val=Val+ParallelSemitonePenalty;

IF Cn2 AND Int8Unison AND (ABS(Int8-Other6) MOD 12)+Unison
THEN Val=Val+ParallelUnisonPenalty;

IF ValInfinity THEN RETURN(Val);

I penalize tritones between voices;
IF Int8=Tritone THEN Val=Val+TritonePenalty;

The following block of code attempts to ensure that a 6-5 chord is properly prepared
and resolved. If there is a dissonance between these two voices, it is either a fourth
or tritone (6-3 or 5-3 chord), a second (6-5 with 6th above), or a seventh (6-5 with
fifth above). The dissonance in the 6-3 chord is considered unobjectionable. In the
implementation here, the 6-5 chord is prepared by holding the fifth while the sixth is
added (a sort of suspension below), then "resolve" the fifth by moving downward by
step.

IF Species=3
THEN BEGIN
IF Dissonance(Int8) AND Int8Unfourth
THEN BEGIN
INTEGER ourLastInt;
ourLastInt=(LastCp-Bass(Cn-1),v) MOD 12;
IF ourLastUnison
THEN BEGIN
IF ourLastUnififth
THEN
IF (ASkip(Cp-LastCp) OR Cp=LastCp)
THEN Val=Val+UnresolvedSixFivePenalty
ELSE
IF (ASkip(Other6-Other6) OR Other6=Other6)
THEN Val=Val+UnresolvedSixFivePenalty
END;
IF Dissonance(Int8) AND Int8Unfourth AND Int8Semitone
THEN
IF (Int8Semitone AND Cp=LastCp)
THEN Val=Val+UnpreparedSemitonePenalty
END;
I penalize direct motion to perfect consonance except at the cadence;
IF NOT LastNote(Cn,v) AND
DirectionToPerfectConsonance(LastCp,Cp,Other6,Other8)
THEN Val=Val+InnerVoiceDirectToPerfectConsonancePenalty;

I if we have an unraised leading tone it is possible that some other;
I voice has the raised form thereof (since the voices can move at very
I different paces, one voice's next to last note may be long before
I another's);
IF LastPitch=18 AND
THEN Val=Val+DoubledLeadingTonePenalty;

I similarly for motion to a tritone;
IF NotionType(LastCp,Cp,Other6,Other8)=DirectMotion AND Int8=Tritone
THEN Val=Val+InnerVoiceDirectToTritonePenalty;
Species Counterpoint

I look for a common diminished fourth when a raised leading tone is in
the bass, a "major third" above it is actually a diminished fourth.
Similarly, an augmented fifth can be formed in other cases;
IF AcctPitch = 3 AND (OtherPitch MOD 12) = 5
    THEN Val = Val + AugmentedIntervalPenalty;

I try to encourage voices not to move in parallel too much;
IF NotionType = (LastCh, C, D, Other), OtherCh = ContraryNotion
    THEN Val = Val + NotContraryToOtherPenalty;
END;

Now we must check the current contents of the chord being formed:

IF check for doubled third;
    IF interval[base(123)] < 1 THEN Val = Val + ThirdDoublePenalty;

IF check for doubled sixth;
    IF interval[base(123)] > 0 AND interval[base(6)] < 1 THEN Val = Val + SixthDoublePenalty;
    IF check for too many voices at octaves;
    IF interval[base(12)] > 0 THEN Val = Val + TripletBasePenalty;

IF check for doubled fifth;
    IF interval[base(5)] < 1 THEN Val = Val + FifthDoublePenalty;
    IF check that chord contains at least one third or sixth;
    IF nNumParts AND ORT LastNote(Ch,v) AND interval[base(12)] < 0 AND interval[base(6)] = 0
        THEN Val = Val + NotTripletPenalty;

I discourage all voices from skipping at once;
IF nNumParts AND AllSkip
    THEN Val = Val + VoiceSkipPenalty;
    I except in 5th species, disallow 3-5 chords altogether;
    IF interval[base(5)] > 0 AND interval[base(12)] > 0 AND Species = 5
    THEN Val = Val + FifiveChordPenalty;
RETURN (Val);
END;

Searching Methods

There are many ways to search for acceptable solutions to a counterpoint problem.
The main constraint is compute time. If we make an exhaustive search of every
possible branch of a short (10 note) two voice first species problem, we have 1610
possible solutions. Even if we could check each in a nanosecond, an exhaustive
search in this extremely simple case would take 1000 CPU seconds (about 20 minutes)
on the F4. Because we hope to handle problems far more complex than this simple
one and hope to do it reasonably fast, we must find a smarter search method.

The underlying method is a recursive search. It starts from its current pitch and
tries in succession all possible melodic intervals from that pitch (if necessary), looking
for any such interval whose associated cumulative penalty is less than the current
best overall penalty. The cumulative penalty is the sum of all the penalties associated
with each of the notes in the counterpoint melody. The overall best penalty is the
lowest cumulative penalty of any complete counterpoint solution found to that point in
the counterpoint. At the beginning of time this overall best penalty is infinity (no
solutions have been found), but once any solution at all is found, the overall penalty
is reset to that new number. If a position is reached that makes further progress
impossible, the searcher backs up one note and tries some other interval. The
intervals are chosen in an order that maximizes the chance of finding a good
interval quickly. If any solution at all exists, we are guaranteed to find it. Given enough time, we are also guaranteed to find the "best" solution according to the rules. By checking only those branches whose penalty is less than the current minimum, we can drastically reduce the number of branches that must be checked, but total execution times can still be high. For a two part counterpoint with a short cantus firmus it is not unreasonable to carry out such a search, but more complex cases drag to a halt. Since our initial goal was to write five to eight part mixed species counterpoint, we obviously need more intelligence guiding the search.

The next method tried was inspired by an article in Science about computer circuit design by simulated annealing. Like the other methods discussed below, this method gains much of its efficiency by accepting less than optimal solutions — it is not guaranteed to find the best solution, and may in some cases not find any solution at all. In this algorithm, our initial counterpoint is just any random collection of notes. Our annealing "temperature" is the number of semitones each of these notes can move. Time is represented by successive passes over the counterpoint applying the same rules as were applied in the recursive search case, but here we look for the local minimum penalty (whereas in the recursive case we grabbed the first acceptable branch and started down it). Each note independently moves to its local minimum and the next pass is started. This method is extremely fast, and works well in first species counterpoint. It does not always converge on a very good solution, but we originally thought that we could run it across several random collections, and thereby increase our chance of getting something reasonable. In practice, however, these successive runs do not improve much. But more important, beyond first species the annealing process sometimes cannot find any acceptable solution. This problem is most easily observed in second species where a note can be dissonant if it is a passing tone. As the rules are structured, we do not penalize a step to a dissonance because one more step (if possible) will resolve the dissonance correctly. Nor do we ever look ahead to see if such a resolution is possible. If there is no possible resolution after all (if a passing tone is impossible at that point), the annealer has no good way to back out. After many fruitless attempts to get around this problem we finally jettisoned the entire notion.

Bernard Mont-Reynaud suggested changing the search to be a best-first search. In this version we compute the penalty associated with each possible melodic interval from the current pitch, then continue recursively using the best of these results first. If forced to abandon a branch, we back up and try the next best interval until a complete solution is found. The first such solution may not be a very good solution, however, because a melody can be lead down a primrose path into a quagmire (by accepting the smallest local penalty we risk falling into a bad overall pathway). If the program is told to search every branch (as in the earlier method), we once again get bogged down in long computations. So a new twist is added. Once we have a solution we drop back to the very beginning and try a different beginning interval. This rather odd looking practice grew from experience watching many hundreds of runs — generally most of the wasted effort (branches checked that led nowhere) seemed to be attributable to the fact that these new branches made no real difference in the global appearance of the given melody — much time was being spent optimizing something that had already given all it had to give. By trying a new starting interval, we maximize the chance of finding a truly different solution. Once again we abandon any branch if its accumulated penalty is above that of the best complete solution found so far. Although first through fourth species are quickly solved, fifth species is still a problem. The number of choices increases in this species not only because it generally has more notes in the counterpoint melody, but also
because we have a number of possible rhythmic values to assign each note. Several optimizations were added to reduce this problem: we search only for those solutions that are markedly better than the current one (as determined by the variable PenaltyRatio), and once a solution has been found we don’t spend too much time on any other single interval (controlled by the variables MaxBranch and Branches). Compute times for multi-voice fifth species can be high even with all this machinery.

To illustrate how the rules help the program decide whether one melody is better than another, take the following three acceptable counterpoints to the bottom line (the cantus firmus). The number under each note is the penalty associated with that note. Despite the fact that the first (top) solution starts out with a higher penalty, it ends up with a lower overall penalty. This case illustrates that a simple best first search is not entirely adequate in all cases.
We can point out in detail how each note in these lines gets its penalty:

In the first attempt at multi-part counterpoint we solved one voice at a time and built up the entire ensemble by layering. This worked well for three voices because the first voice added was always pretty good, and the second added voice still had enough degrees of freedom to find an acceptable solution. As more voices were added however, the later layers became less and less acceptable. It became clear that the entire ensemble has to be calculated together, that all the voices must be examined to decide the current overall best configuration. The search routines that follow implement this form of search.
Species Counterpoint

First we need a place to save the last complete solution while we search for a better one:

```plaintext
INTEGER ARRAY BestFit[1:HostNotes,1:HostVoices];
```

and variables to hold the current maximum penalty (MaxPenalty), the current actual best (lowest) penalty (BestFitPenalty), the ratio that determines how much better a new solution has to be to be worth pursuing (PenaltyRatio), a flag to tell the searcher when to quit (AllDone), the current branch counter (Branches), and the maximum number of branches we are willing to search before giving up (MaxBranch).

```plaintext
INTEGER BestFitPenalty,MaxPenalty,Banches,MaxBranch;
BOOLEAN AllDone;
REAL PenaltyRatio;
```

We will examine only the best continuations from any given point, making no effort to save every possible continuation. The macro NumFields determines how many continuations we save at each branch.

```plaintext
DEFINE NumFields=18;
```

Each continuation consists of its associated penalty and the melodic intervals of each voice that make up the next note of the continuation. At each branch we search for the NumFields best continuations and save them in an array. The following code implements this mechanism.

```plaintext
DEFINE Field="(HostVoices)";
DEFINE End="(FieldNumFields)";
PROCEDURE ClearSpace(INTEGER ARRAY Sp);
ARRAY Sp[infinity];
INTEGER PROCEDURE SaveIndex(INTEGER index, INTEGER ARRAY sp);
BEGIN
  BEGIN
    IF index < current NUMFIELD-th worst,
      THEN find its position in SP, insert space for its data, and return a pointer to the block. The blocks are stored "backwards" for陣延;
    END;
  END;

INTEGER i;
FOR i:=End STEP -Field UNTIL 0 DO
  IF Sp[i] index THEN DONE;
  IF i=0
    THEN 8 is the end of the list. If i=8 then we insert INDEX;
    THEN
      BEGIN
        NUMFIELD(Sp[i],Sp[Field],i);
        Sp[i] index;
        BEGIN
          SP[i] penalty for block starting at i. SP[i-1]=index into 1 melodic interval array for voices 1, SP[i-2] for voice 2 and so on. The searcher starts at SP[EndP] and works backwards 1 through the stored continuations as it searches for a satisfactory overall solution;
        END;
      END;
      RETURN;
END;
```

The next procedure takes a newly completed solution and saves it. At this time we also check for problems associated with raised leading tones. If the musica ficta rules were at all clear, we could also add them to the solution. The raised leading tone code can run into problems – a future version may try to be smarter about cadential passages.
PROCEDURE SaveResults(INTEGER CurrentPenalty, Penalty, v, i, Species);
BEGIN
        INTEGER i, v, LastPitch, v, Cn, k;
    FOR v = 1 STEP 1 UNTIL v = 1 DO
        BEGIN
            Cn = TotalNotes[i];
            LastPitch = (Cn - 1, v) MOD 12;
            IF NOT InNote(LastPitch, Notes[i]) THEN
                BEGIN
                    i = i + 1;
                    BEGIN
                        Result = Result + Penalty;
                        FOR k = 1 STEP 1 UNTIL k = Cn DO
                            BEGIN
                                IF k = (Cn - 1) THEN DONE;
                                Pitch = (Cn - k, v) MOD 12;
                                IF (Pitch = v AND PitchMod) OR
                                    (v(v = v AND PitchMod)) THEN
                                    BEGIN
                                        Step = Step - Penalty;
                                        THEN Done;
                                    END;
                                    Pitch = (v(v = v AND PitchMod)) MOD 12;
                                    IF pitch = Fourth OR Pitch = Fifth OR Pitch = Minor Third OR
                                        Pitch = Octave THEN DONE;
                                    ELSE
                                        FOR k = 1 STEP 1 UNTIL k = Cn DO
                                            BEGIN
                                                IF i = v AND (Other(Cn - k, v) MOD 12) = 11 THEN DONE "outer";
                                                IF (v(v = v AND PitchMod) AND minor third) OR
                                                    (v(v = v AND PitchMod)) THEN
                                                        Step = Step - Penalty;
                                                        THEN Done;
                                                END; "outer"; END;
        END;
    END;

Musica ficta are not entirely trivial to add to the program. If we were doing so, this
would be the place to do it. The ficta-finder would run through all the voices here
looking for tritones. If one is found a decision has to be made whether to alter
(fatten or sharpen) the current note, or leave it alone. In a simple version tried
during the development of this program, we used the following sequence:

Step through all voices
Step through all notes of each voice
If current pitch is B (II) and it is approached and left by either
a unison, a rising major second, a rising major second, a
descending minor second, or a descending minor third (to
avoid creating illegal augmented or diminished intervals
melodically), and it forms a tritone with some other voice,
Then
If melodic interval is not a unison
Then flatten the B
Else
While pitch is B, look to see if the flattened form will
create tritones (E in other voices), and check to see that
the interval on either side of the repeated B’s is the
correct form of interval. If all this is true, flatten all
the B’s.

This algorithm is moderately conservative (it makes no attempt to sharpen F’s for
example), but still makes a mess of certain passages. For example, take the simple
passage:
Here our algorithm flattens the second C but not the first (correctly avoiding a tritone melodically), but perhaps the correct fit is an F-sharp. The algorithm also ignores the possible presence of the unaltered pitch in other voices as in:

In any case, since fitc are not a primary concern of the author, the entire problem was shunted aside and left as an exercise for the interested reader.

We have now updated the value of the best fit penalty and the maximum penalty that a new solution should have, and have signalled the searcher to back up to the start again. Next we save the current state of the CtrPit array (which holds the solution transposed to C) in the BestFit array, adding back the original transposition:

Now we print out the results for debugging and what not.

We need an array containing the legal melodic intervals in the order in which we want them to be checked. Whether this order is of great importance or not depends on how the searching is carried out. In the current scheme, the order doesn’t matter much.

PRELONG WITH
   1,-1,2,-2,3,-3,4,-4,5,7,-5,8,12,-7,-12;

INTEGER ARRAY Index(1:161)
We need a procedure that examines the possible continuations and puts the best such continuations in order in a local array (on the stack). Each time the searcher lurches forward a note, a new recursive call is made allocating this and other arrays of local data. We can back up without losing previous state merely by returning from a call.

**Recursive Procedure Look**

```
RECURSIVE PROCEDURE Look(
    INTEGER CurPen, CurVoice, NumParts, Species;
    ARRAY integer Pan, Is, CurNotes;
BEGIN
    BEGIN penalty, Pit, I;
    FOR I = CurVoice - 1 STEP 1 UNTIL 10 DO
        pit=Index(i, CurNotes) - Index(i, CurNotes[CurVoice - 1], CurVoice);
    END;

    Pit is the proposed new pitch for the current voice (CurVoice). For each such pitch we check how it affects the global penalty of the rest of the active voices.

    penalty = CurPen + Check(CurNotes[CurVoice], Pit, CurVoice, NumParts, 
        IF CurVoice = NumParts THEN Species ELSE 1); 
    Settle(CurNotes[CurVoice], Pit, CurVoice);
    IF penalty <= Mass 
        THEN 
            IF CurVoice = NumParts
                THEN 
                    BEGIN

We have a possible continuation for CurVoice. Now we need to find whether other voices can continue also given this pitch. Since the voices may not be moving together, we check new pitches only for those voices that are getting a new note at the current time.

```

FOR I = CurVoice + 1 STEP 1 UNTIL NumParts DO
    IF CurNotes[I] = 0 THEN DONE;
    IF NumParts 
        THEN Look(Penalty, I, NumParts, Species, Lim, Pan, Is, CurNotes);
    ELSE
        BEGIN
```

If the current voice is the last voice, then we save the current proposed overall continuation in the Pens array, and continue looking for more. When we finish this loop we will have all the best solutions in order in the Pens array.

```
BEGIN
    BEGIN x, j;
        xySaveIndex(Penalty, Pan);
        IF x > 0
            THEN 
                FOR I = 1 STEP 1 UNTIL NumParts DO 
                    Pan[x] = I + Is(I);
                ELSE
                    Lim: Lim MIN Penalty;
    END;
END;
```

Next we need a procedure that calls `Look` (given above), and coordinates all the voices as the ensemble grinds toward the cadence.

**Recursive Procedure BestFitFirst**

```
RECURSIVE PROCEDURE BestFitFirst(INTEGER CurTime, CurrentPenalty, NumParts, Species);
BEGIN
    BEGIN i, j, Lim, ChoiceIndex, NextTime, CurTime;
        INTEGER ARRAY Pens[I, Field, NumFields], Is[I, NumParts], CurNotes[I, Num Voices];
        IF aiDone OR CurrentPenalty > MaxPenalty THEN RETURN;
```

`AllDone` is true after we have found a solution. At that point we exit all the calls currently active and start afresh with a new maximum penalty.
Species Counterpoint

```
1
ChoiceIndex=End;
AllDone=FALSE;
RRRCl(1,Pane,Infinity);
RRRCl(1a);
RRRCl(1);
Branches=Branches+1;
IF Branches=MaxBranch THEN RETURN;
IF (Branches MOD 181) = 0
THEN
MaxPenalty=MaxPenalty+PenaltyRatio;
```

If we just let the searcher run until a solution is found, it happens quite frequently that the search gets completely bogged down in a blind alley, spending immense amounts of time beating up against an impossible cadence situation. The branch counter gives us a way to jump out of such a situation. In addition, as we spend more and more time on a given attempt, we gradually reduce the acceptable maximum penalty somewhat like a person getting more and more frustrated as more effort is poured into a fruitless search.

```
CurMin=Infinity;
Lim=Best Penalty-Current Penalty;
NextTime=Infinity;
FOR i=1 STEP 1 UNTIL NumParts DO
BEGIN
CurTime=NextTime[Index(CurTime,1),1];
IF CurTime=0
THEN NextTime=NextTime MIN CurTime;
END;
```

We now know what the next note is overall. We set up the CurNotes array marking all voices that have an onset at the minimum next time. Each of these voices will then take part in the search for the best overall continuation.

```
FOR i=1 STEP 1 UNTIL NumParts DO
IF Onset[i]=MaxTime[Index(CurTime,1),1]=NextTime
THEN CurNotes[i]=1;
FOR i=1 STEP 1 UNTIL NumParts DO
IF CurNotes[i]=0 THEN DONE;
Look(i,NumParts,Species,Lim,Pane,1s,CurNotes);
```

Now we have the NumFields best continuations saved in the Pens array along with the associated penalties. We go through these continuations one at a time, trying to find one that will get us all the way to a cadence.

```
CurMin=Pens[ChoiceIndex];
IF CurMin=Infinity THEN RETURN;
WHILE NOT AllDone DO
BEGIN
IF CurTime=TotalTime
THEN
IF (CurMin=Current Penalty)=MaxPenalty
THEN RETURN ELSE
ELSE
IF (CurMin=Current Penalty)=Best Penalty
THEN RETURN;
```

We are still below the global maximum penalties, so set up the next continuation and give it a whirl.

```
FOR i=1 STEP 1 UNTIL NumParts DO
IF CurNotes[i]=0
THEN Set(i,NumParts,Index(Pens[i]=Index(CurMin=i),1),1);
IF NextTime=TotalTime
THEN Best[i]=0 (NextTime,Current Penalty,CurMin,NumParts,Species)
ELSE
SaveResults(Current Penalty,CurMin,NumParts,Species)
```
```
The variable *ChoiceIndex* points to where we are in the continuations. By decrementing it by field size, we move on to the next best continuation.

```plaintext
ChoiceIndex = ChoiceIndex - Field;
IF ChoiceIndex = 0 THEN RETURN;
Current = Pen[ChoiceIndex];
IF Current = infinity THEN RETURN;
IF Current > THEN RanPenalty > BestFit Penalty Penalty Ratio;
END;
END;
```

The only thing left that is of any interest is the code that decides what rhythms to employ in fifth species. For simplicity's sake, we just load up an array with the legal rhythmic patterns and choose among them randomly. This approach obviously leaves much to be desired. Musical styles are differentiated more by rhythmic practices than melodic, and on a smaller level, it is the fluid handling of rhythm that makes a group of sounds a piece of music rather than a pedagogical exercise. However, this entire program ignores issues of phrasing or larger melodic and rhythmic structures. If we decide to carry this effort further in that direction, a much more complex decision and search mechanism will be required. Just as Fux's book was one step on the way toward Parnassus, this program, simple minded, even toy-like in many ways, may lead toward good music somewhere down the road. In any case, messing around with counterpoint is a pleasure in itself, so the author needed no further justification.

```plaintext
INTEGER Seed;  
random number sequence seed;
INTEGER ARRAY Rhythms[8][8][8], Rhythms[1][8];
    | array of legal rhythmic patterns;
SIMPLE PROCEDURE FillRhythms;
BEGIN
INTEGER i, j;
Rhythms[1][1][1] = allNote;  
    | for first species "rhythm";
Rhythms[1][1][2] =
FOR i = 1 STEP 1 UNTIL 2 DO Rhythms[i][1][1] CASE OF (0, halfnote, halfnote);
Rhythms[1][1][2] =
FOR i = 1 STEP 1 UNTIL 3 DO Rhythms[i][1][2] CASE OF (0, halfnote, quarternote, quarternote);
Rhythms[1][1][3] =
FOR i = 1 STEP 1 UNTIL 4 DO Rhythms[i][1][3] CASE OF (0, quarternote, quarternote, halfnote);
Rhythms[1][1][4] =
FOR i = 1 STEP 1 UNTIL 5 DO Rhythms[i][1][4] CASE OF (0, quarternote, quarternote, eighthnote, eighthnote, halfnote);
Rhythms[1][1][5] =
FOR i = 1 STEP 1 UNTIL 6 DO Rhythms[i][1][5] CASE OF (0, quarternote, quarternote, eighthnote, eighthnote, quarternote, quarternote);
Rhythms[1][1][6] =
FOR i = 1 STEP 1 UNTIL 7 DO Rhythms[i][1][6] CASE OF (0, eighthnote, eighthnote, eighthnote, eighthnote, eighthnote);
Rhythms[1][1][7] =
FOR i = 1 STEP 1 UNTIL 8 DO Rhythms[i][1][7] CASE OF (0, quarternote, quarternote, quarternote, eighthnote, eighthnote, eighthnote);
Rhythms[1][1][8] =
Rhythms[1][2][1] = allNote;  
    | for first species "rhythm";
Rhythms[1][2][2] =
END;
REQUIRE FillRhythms INITIALIZATION;
```

Now a few simple procedures to clear and access the arrays associated with the rhythm finder.
Species Counterpoint

PROCEDURE UsedRhyme(INTEGER n);
RhyPat(n,0)=RhyPat(n,0)+1

INTEGER PROCEDURE CurRhyme(INTEGER n);
RETURN(RhyPat(n,0));

PROCEDURE CleanRhyme
BEGIN
INTEGER i;
FOR i:=1 STEP 1 UNTIL 9 DO RhyPat(i,0)=0;
END;
INTEGER PROCEDURE GoodRhyme
BEGIN
INTEGER i;
if (RandNum(i)) THEN RETURN(1 MAX (1-1));
if (CurRhyme(i)=CurRhyme(i-1)) THEN RETURN(0 MIN (1-1));
RETURN(1);
END;

The rest of the program merely stuffs the cantus firmus into the CtrPt array and sets
the starting points of the various voices. We append several examples of
counterpoint written by the program. Under each counterpoint note is the associated
penalty. Currently the program has no provision for starting a melody with a rest,
nor does it reward invertible counterpoint and imitation. It tends to let voices get
entangled in each other, and makes no decisions about overall melodic shapes.
Certain implied vertical clashes are not noticed (especially those involving tritones).
Given these caveats the program does quite well in a reasonably short time.

Acknowledgments

I thank Jonathan Berger for his very valuable advice and criticisms.