Preliminary report on the design of a constraint-based musical planner

Other

AUCS / TR9113

Preliminary report on the design of a constraint-based musical planner.

© *Simon Holland*

Friday, 24 May, 1991

Department of Computing Science
University of Aberdeen
King's College
Old Aberdeen,
AB9 2UB, Scotland, UK.

Abstract

This work described in this paper forms part of a wider research project, described in Holland (1989), to find ways of using artificial intelligence methods to encourage and facilitate music composition by musical novices. This paper focusses on the key component of a knowledge-based tutoring system under development to help novices learn to compose and analyse musically 'sensible' chord sequences. This key component is a constraint-based musical planner dubbed 'PLANC'. The musical planner (together with its set of musical 'plans') can be used to construct and analyse chord sequences in terms of musical strategies that can be understood and made use of by complete musical novices. PLANC can generate a class of musically 'interesting' chord sequences that include the chord sequences of many well known existing pieces of music, as well as generating a large space of new 'interesting' sequences. The design of the planner draws on a characterisation of creativity due to Johnson-Laird (1988). The planner is psychologically plausible , though not intended as a detailed cognitive model. An overview of the structure of PLANC is presented, and its suitability for use in a tutoring system is considered. The design of the planner is criticised. Each of the main components of PLANC is analysed: plan variables, constraints, value generators and methods. Much of the 'knowledge' used in PLANC consists of informal musical knowledge: three appendices analyse the different kinds of informal knowledge used. The applicability and value of similar constraint-based mechanisms in intelligent tutors in a wide range of other open-ended domains is considered.

Key Words

Education, Intelligent Tutoring Systems, Music Composition, Guided Discovery Learning, Constraints, Artificial Intelligence, Interactive Learning Environment.

Preliminary report on the design of a constraint-based musical planner.

Simon Holland

Friday, 24 May, 1991

Department of Computing Science
University of Aberdeen
King's College
Old Aberdeen,
AB9 2UB, Scotland, UK.

Abstract

This work described in this paper forms part of a wider research project, described in Holland (1989), to find ways of using artificial intelligence methods to encourage and facilitate music composition by musical novices. This paper focusses on the key component of a knowledge-based tutoring system under development to help novices learn to compose and analyse musically 'sensible' chord sequences. This key component is a constraint-based musical planner dubbed 'PLANC'. The musical planner (together with its set of musical 'plans') can be used to construct and analyse chord sequences in terms of musical strategies that can be understood and made use of by complete musical novices. PLANC can generate a class of musically 'interesting' chord sequences that include the chord sequences of many well known existing pieces of music, as well as generating a large space of new 'interesting' sequences. The design of the planner draws on a characterisation of creativity due to Johnson-Laird (1988). The planner is psychologically plausible , though not intended as a detailed cognitive model. An overview of the structure of PLANC is presented, and its suitability for use in a tutoring system is considered. The design of the planner is criticised. Each of the main components of PLANC is analysed: plan variables, constraints, value generators and methods. Much of the 'knowledge' used in PLANC consists of informal musical knowledge: three appendices analyse the different kinds of informal knowledge used. The applicability and value of similar constraint-based mechanisms in intelligent tutors in a wide range of other open-ended domains is considered.

CONTENTS

1 Introduction

This work described in this paper forms part of a research project to find ways of using artificial intelligence to encourage and facilitate music composition by musical novices. This paper describes the key component of a knowledge-based tutoring system called 'MC' (Holland, 1989) to help novices learn to characterise and compose musically 'sensible' chord sequences. This key component, dubbed 'PLANC' (which may be pronounced as "Plan C" or to rhyme with 'Clancy' according to taste) is a constraint-based musical planner. The musical planner (together with its set of musical 'plans') can be used to construct and analyse chord sequences in terms of musical strategies that can be understood and made use of by complete musical novices (Holland, 1989) with very little training. PLANC can generate the chord sequences of many well known existing pieces of music as well as generating a large space of new but musically 'interesting' sequences. The design of the planner draws on a characterisation of creativity due to Johnson-Laird (1988). The use of constraint-based planners in intelligent tutoring systems for open-ended domains (Holland, 1989) other than music is discussed.

This paper is one of a set giving full technical details on the constraint-based musical planner PLANC. Parts of the detailed description of the planner (especially section 5) are much easier to read if the reader is already familiar with the musical planner in general terms. Such familiarity is assumed in a few places. Overviews of the planner can be found in Holland (1989) and Holland and Elsom-Cook (1990) .

Background

In Holland (1989) and Holland and Elsom-Cook (1990) a framework is presented for a knowledge-based tutoring system (MC) to help novices learn to compose music. One goal of the tutor is to help students learn how to compose new, musically coherent chord sequences. A core problem in developing such a tutor is to devise a representation for 'strategic' aspects of chord sequences. To be pedagogically useful, the strategies must meet the following requirements. They need to account for a wide range of existing sequences. They must be usable by beginners to generate new, interesting, musically coherent sequences. It must be potentially easy to communicate to novices how and why the strategies 'work'. A plan-like representation, a family of musical 'plans', and a plan interpreter that fit these requirements has been implemented and is now described.

Where to find out more

The present paper describes and analyses in detail the constraint-based representation, the design and implementation of the prototype of the planner, and one sample musical plan. Other technical aspects of the planner are covered elsewhere in Holland (1991b, 1991c and 1991d). These three papers deal respectively with the following aspects of the planner ;

- illustrations of the kinds of musical transformations that the planner can support, and an analysis of their educational usefulness,
- details of several contrasting implemented musical plans,
- details of the representation used by the planner to describe chord sequences in a concise, perceptually meaningful fashion (Harmony Talk, formerly called Harmony Logo).

Structure of the paper
The present paper is organised as follows. Firstly, by way of motivation, we will give an overview of the characteristics of PLANC, and their appropriateness for the task under consideration. Next we will discuss the basis of the form of computation of the planner, namely constraint satisfaction. We will then discuss in detail each of the main components of PLANC: the plan variables, the constraints, the value generators and the methods. We will also discuss control of search in PLANC. Much of the 'knowledge' used in PLANC is to a greater or lesser degree informal musical knowledge: there are three appendices discussing the nature of the different kinds of knowledge. Finally we discuss the limitations of the musical planner, draw conclusions, and point out possibilities for further work.

A note on terminology
The word 'planning' is used quite diversely in the AI literature. We have adopted the definition shared by Stefik (1980), and Cohen and Feigenbaum (1982) that planning is deciding on a course of action before acting. From this viewpoint, planning is a special case of problem solving, which may be defined as developing a sequence of actions to achieve a goal.

2 Characteristics of the planner
The architecture of PLANC has various characteristics advantageous for use in an intelligent tutor for music composition. We will list these features now, saying why they are desirable and then show in the remainder of this paper how they are realised. The educationally useful characteristics are as follows.

Firstly, PLANC permits the student to work bottom up or top down. That is to say, it is possible to use PLANC to experiment with the *details of a plan* without having to specify the *strategic aspects of the plan.* Example details could include the use of the Locrian mode, or a certain 'home establishment method', or the degree on which the modulation takes place in the 'moving goal-post plan' (see

Holland 1989 for details). Conversely, it is also possible with PLANC to experiment with broad strategic aspects of a plan without needing to specify (or understand) the details. PLANC behaves as though it can work backwards to guess the likely or plausible strategic implications of a local detail, and then work forwards to co-ordinate numerous details to fit with the global choices. There are two specific reasons for considering this flexible style of computation educationally advantageous. Firstly, it offers an important opportunity to improve a user's motivation, since general ideas may be more eagerly absorbed when their relevance to some particular detail of current interest can be capitalised on. Secondly, the combination of top down and bottom up approaches to design of a complex artifact reflects the varied way that composers (and other designers) seem to prefer to work (Sloboda 1985). Composers seem to work in any of the following ways at different times. Sometimes an encompassing idea is fleshed out top down, while at other times a small fragment is elaborated upwards into an idea that conforms to the fragment: often a combination of these two approaches is used. For these reasons, PLANC is designed to treat its variables in the "multiway" fashion of logic programming or constraint satisfaction, as opposed to the more conventional functional fashion. That is to say, instead of all plan variables being treated as input to the specification, any subset of variables may be treated as input, and any subset as output (though with some exceptions, noted later). Such a scheme is not without problems. We can briefly characterise the main problem, and its solution as follows. In general, not all functions have well-defined **inverses**. For example, if the modulo base 10 of some number is 4, the unknown number could be 4, 14, 24, or 34, or any of an infinite number of other possible values. If a planner made an arbitrary choice for such an inverse, there would always be the possibility that it might have to be retracted later to fit with some other constraint. There would remain an infinite number of possible values to search. Consequently, there would be no guarantee in general that the process would halt. To avoid this problem, the solution adopted in PLANC is to restrict variables to finite sets of possible values in all cases.

The second educationally desirable feature of PLANC is concerned with the manner of availability of default values. To avoid the user having to deal with questions that he may not yet understand or care to answer, PLANC must have defaults values always available. Yet it would be quite against the spirit of the task if " hard-wired" defaults were used. PLANC is organised so that where appropriate, choices of values for variables may operate as heuristics influencing the order in which default values are produced for variables in other parts of the plan. For example, we will see that the choice of 'hometype' in the 'return home plan' does not prevent the 'establish home' constraint from taking any value, but it does affect the default order in which such values are produced.

A third characteristic is that if more than one song fits a particular specification, PLANC is able to produce alternative versions. To deal with cases in which there could be very many solutions, PLANC makes use of a simple mechanism that tends to present musically "preferred" solutions first (in a sense to be explained later).

Fourthly, PLANC allows musical plans to be used in hierarchical or recursive fashion, i.e to call each other or themselves. It is possible to nest suitably constrained musical plans as subsystems or methods of achieving effects in other musical plans. This is not described in this paper but is explained briefly in Holland (1991c).

Fifthly, one useful general feature of PLANC is that the predominantly declarative statement of the plans appear to form a good foundation for a future version of PLANC that could explain its assumptions and reasoning to a student.

Taken as a whole, this paper demonstrates a theoretical framework and series of practical techniques whereby musical plans, as presented informally in Holland (1989) can be realised in computational form with the educationally useful characteristics identified above. The framework uses the metaphor of constraint satisfaction, implemented in logic programming using finite generators of default values, and other techniques. To begin our exposition, we will identify a theoretical and practical basis for the form of computation of the planner.

3 Basis of the form of computation of the planner

We claim that we can view the satisfaction of a musical plan as being a **special case of the formal class of constraint satisfaction problems** (CSP). Hentenryck and Dinebas (1987) give an admirably concise formal definition of this class of problems.

> Assume the existence of a finite set I of variables $\{X_1, X_2,.....,X_n\}$ which take respectively their values from the finite domains $D_1, D_2,.....,D_n$ and a set of constraints. A constraint $c(X_{i1},.....X_{ik})$ between k variables from I is a subset of the cartesian product D_{i1} x .... x D_{ik} which specifies which values of the variables are compatible with each other. A solution to a CSP is an assignment of values to all variables which satisfy all the constraints and the task is to find one or all the solutions.

Hentenryck and Dinebas (1987) go on to give a clear statement of the simplest (though not the most efficient) way of solving constraint satisfaction problems using Logic programming languages. They point out that

> "Given a particular CSP, it is sufficient to associate a logic program with each kind of constraints (sic) and to provide a generator of values for the variables."

Given the above basis, the form of computation in the planner PLANC is quite simple, in principle. On the other hand, the formal versions of the musical plans we saw in Holland (1989) and Holland and Elsom Cook (1990), if encoded in terms of primitive musical constraints, such as those proposed by Levitt (1985), would form large, highly complex constraint nets. The problem of making such nets of constraints comprehensible to novice students would be a substantial problem.

These two problems (of actual and perceived complexity) are tackled by limiting the use of constraint satisfaction to co-ordinate only the most essential *strategic* aspects and the most essential low-level *details* of the plan. The high-level decisions taken by constraint satisfaction are expressed predominantly in terms of Harmony Talk (Holland 1989) concepts. More specifically, the high level decisions made by constraint satisfaction are harnessed to specify Harmony Talk programs. The programs produced by constraint satisfaction are translated into notes by the Harmony Talk interpreter. Thus, Harmony Talk concepts are used to keep the "grain size" of the constraints manageable. This has advantages and disadvantages. One advantage is that the net of constraints required becomes dramatically simplified. This has an unexpected educational bonus, because it reveals the suprisingly small amount of ordinary musical knowledge that is sufficient, when appropriately applied, to support musically intelligent behaviour over a wide range of case studies. The exploitation of concepts derived from Longuet-Higgins' (1962) and Balzano's (1980) theories means that the primitive concepts used by PLANC (e.g. harmonic trajectories) are potentially easy to explain to musically uneducated novices (through the medium of Harmony Space, as illustrated in Holland (1989)).

To summarise, the PLANC framework offers a simple way of combining the constraint satisfaction method of computation outlined above with notions derived from Harmony Space and Harmony Talk to produce an appropriate, workable and relatively understandable planner suited to our educational purposes. We illustrate how the simple approach to computation outlined above can be applied to create a manageable, relatively transparent and useful planner capable of the educationally useful behaviour outlined above.

4 The components of a musical plan: variables, constraints, finite generators, methods and ordered inference

We will now demonstrate how our informally stated musical plans can be translated into formal constraints. We will need to define at least three computational entities already mentioned for our plans: a finite set of variables, a set of constraints and a set of value generators. We will also need to define a set of *musical methods*, and ways of controlling inference by the planner, both explained below.

4.1 Methods

It appears that the constraint satisfaction paradigm could have been used to implement the planner in full from top level concepts down to note level. However, for reasons outlined above, this was not done.One reason is that this approach would have run the risk that the workings of the planner at the middle and lower levels would have involved long chains of inference that might be hard to explain to novices. A second reason was the sheer complexity of the encoding that would have been required. For these reasons, the values of variables representing the important strategic aspects of the chord sequence, and the most important low level details, are calculated by constraint satisfaction and input into low level procedures called *methods*. The job of the methods is to procedurally translate combinations of variables into Harmony Talk code using slot filling and Harmony Space simulations. This allows various features of harmonic sequences to be encoded and manipulated very concisely. Thus, the constraint satisfaction process in effect drives a code generator that produces a Harmony Talk program describing the fully instantiated plan. (As a slight complication, elements of code produced in this way may be used as an input to constraints elsewhere in the plan, which may casue backtracking, as we will see later). In principle, a Harmony Talk interpreter may be used in the normal way to translate this program into sequences of notes. (However, see the 'limitations' section at the end of this paper for discussion of a practical limitation on this process under the current implementation.) The use of "methods" in this way has advantages and disadvantages. One advantage is that in many cases the planner is able to exploit the concepts of Harmony Talk, and their underpinnings in Balzano's (1980) and Longuet-Higgins' (1962) theories, to deal with harmonic phenomena in a very concise way. One price payed for this simplicity and transparency is that the mapping from musical plans concepts into fragments of harmony code, while precise in some cases, is somewhat arbitrary in others, as we will see shortly.

4.2 Control of search

PLANC associates a logic program with each kind of constraint. The underlying inference mechanism used in PLANC is that of logic programming as partially embodied in Prolog. The built-in search mechanism provided in Prolog is not particularly efficient (depth-first search with backtracking). The use of methods allows the search space to be kept relatively manageable, but even so, the order in which constraints are to be satisified must be considered in order to keep the planning feasible and efficient. We will look at this problem in more detail shortly.

5 Detailed description of PLANC

We will deal in turn with the five components (variables, constraints, generators, methods and control) required to implement a musical plan, using a formal version of the "return home" plan as an example. (In Holland (1991b), we will present some other musical plans, all of which have been implemented using PLANC. One of these plans is presented in full technical detail.)

5.1 Defining plan variables for return home

Our first task is to define *plan variables* that represent tactical or strategic musical alternatives for elements in the plan. There is always more than one way to split an informal musical plan into plan variables. The list of plan variables and their possible values adopted for the "return home" plan is as follows;

Plan variables	Possible values
hometype	(tonal, modal)
mode	(major, minor, dorian, aeolian, mixolydian, lydian, phrygian, locrian)
establish-home	(simple,implicit, repeated)
establish-home-length	(0, 1,2,3,4)
maintain-home	(tonal,jazz, modal+pedal, modal, jazz-chromatic)
away-point	(I, II,III,IV,V,VI,VII)
trajectory-home-length	(0,1,2,3,4,5,6,7,8,9,10,12)
emphasise-final-length	(0,1,2,3,4)
code	strings of Harmony Talk code
song-length	(0,...., 16)

We will now discuss each of these variables in turn, explaining what they refer to and their possible values. It is important to note that in each case we have restricted the number of possible values to a finite set of possibilities. Otherwise, there would be a danger that the constraint satisfaction process might not halt. The discussion of each variable is unfortunately dull. However, the variables are the basis for all of the other components, so the discussion cannot safely be left out. The reader may care to compare the discussion given here with the more informal and intuitive description of the plan given in Holland (1989) and Holland and Elsom Cook (1990).

5.1.1 'Hometype', 'mode', 'away-point' and 'trajectory length'

The first four variables, 'hometype', 'mode', 'away-point' and 'trajectory-length' are unremarkable and easy to understand;

The *hometype* variable has possible values 'tonal' or 'modal'. This variable distinguishes chord sequences that satisfy the return home plan according to whether their hometype is tonal or modal. This concept corresponds with the commonplace musical judgement as to whether a piece is in a major or minor key (tonal hometype), or in some other mode (e.g. dorian, etc. - modal hometype). This variable is common to many plans and can be considered as an instance of a recurrent plan variable 'type', such as 'integer' or 'character' in other languages.

The *mode* variable distinguishes chord sequences that satisfy the return home plan according to their mode. This variable has possible values drawn from the following set: (major, minor, dorian, aeolian, mixolydian, lydian, phrygian, locrian). The ionian mode is not been included, since for all practical purposes it is indistinguishable from the major mode. On the other hand, the aeolian and harmonic minor modes are worth distinguishing due to the difference in quality of the V chord. Like the hometype variable, this variable can recur in many plans and can be considered as a recurring 'type'.

The variable *away-point* distinguishes chord sequences that satisfy the return home plan according to the degree of the scale from which they begin their 'return home'. This variable has possible values drawn from the degrees of the diatonic scale as follows: (I, II, III, IV, V, VI, VII). This variable can be considered as an instance of a diatonic scale 'type'.

The variable *trajectory-length* has possible values in the range 0, ..., 12, and distinguishes chord sequences that satisfy the return home plan according to the length of trajectory employed for their 'return home'. The range of possible values is more or less arbitrary. The top end limit, 12, is proposed on the arbitrary grounds that this is the longest possible length of a harmonic trajectory in a fixed key that does not repeat itself. Note that the value of this variable may or may not have a straightforward relationship with the value of the away-point variable, since the trajectory might be carried out in a context where the alphabet of chords available for the trajectory was restricted.

The variable *emphasise-final-length* has possible values 0, 1, 2, 3, 4 and distinguishes chord sequences that satisfy the return home plan simply according to whether the home chord is sounded once when the chord sequence 'returns home' at the end of the plan, or whether it is repeated (or held for a longer than normal duration) at this point. This variable reflects the commonplace observation that the final chord in a sequence is often repeated (or held longer) to emphasise a sense of finality or arrival. The concept to which this variable refers is perhaps more ad hoc and more arbitrarily defined than the previous variables, but it refers to a straightforward musical phenomenon in tonal music that can have clear consequences in planning the proportions of a chord sequence. (Leaving aside the end of the chord sequence, it is assumed that the harmonic sequences planned take place at a regular, even tempo.)

5.1.2 The 'establish home', 'establish home-length' and 'maintain-home' variables

Two out of the next group of three related variables describe quite complex concepts; but their possible values correspond to a limited number of arbitrary methods of realising them. We will describe the possible values informally here, giving more detail of the way these values are translated into actions in the section on methods. In the discussion of the return home plan in the chapter 8 of the Holland (1989) and Holland and Elsom-Cook (1990), we discussed the need for one chord degree to be distinguishable by the listener as a 'home' location throughout the piece. We also discussed the requirement for the home area to be established at the beginning of the piece, either by explicitly stating the corresponding chord, or by some implicit means such as the use of a modal pedal. As a first approximation, we can represent a limited choice of means to achieve these goals using the three variables 'establish-home', 'establish-home-length' and 'maintain-home', as follows. (Each of these variables is applicable to several other plans apart from "return home")

The *establish-home* variable has three possible values; 'simple', 'implicit' and 'repeated'. This variable distinguishes chord sequences that satisfy the return home plan according to three possible cases;

- simple: the initial establishment of the home area is carried out by a simple explicit statement of the home chord,
- repeated: the initial establishment of the home area is carried out by a *repeated* explicit statement of the home chord,
- implicit: there is no initial statement of the home chord: the establishment of the home area is carried out by some other means.

There are clearly many other ways by which a piece of music could "establish a home area" in a listener's mind. For example, the home area could be initially established by recursive use of a short "return home" sequence, or by the use of some other musical plan that starts and finishes in the home area in an obvious way. The implemented planner has been used successfully for this kind of hierarchical use of plans, as mentioned briefly in Holland (1991c), but it would complicate this example unduly to allow it here. Hence, for simplicity, we will limit the value of the establish-home-method variable to the possible values listed above. The last possible value ("implicit') is simply a place holder to indicate that no explicit method is being used to establish the home area. This was included because many existing chord sequences omit to establish the home area explicitly by simple harmonic means, perhaps relying on markers outside our scope such as melody, embedding in a longer piece with a clear tonal centre, or emphasis in performance to make the home area clear. If the home is not marked in some way, harmonic or otherwise, the plan may be hard to perceive in a way that corresponds to the plan. The 'implicit' value could be used to warn the novice composer to take non-harmonic measures to strengthen perception of the home area.

The **establish-home-length** variable has possible values 0,1,2,3,4. This variable is closely related to the establish-home variable. A value of 0 for this variable corresponds to a value of 'implicit' for the establish home variable and a value of 1 to the value 'explicit'. Values of 2,3, and 4 give the exact number of repetitions in cases of the value 'repeated' . The limitation to a maximum of 4 repetitions is arbitrary.

The last variable in this group, **maintain-home,** has possible values (tonal,jazz, modal+pedal, modal, jazz-chromatic). This variable distinguishes chord sequences according to different heuristic methods by which they *maintain* the perception of the home area on the part of the listener during the relevant part of a musical plan. As in the case of the 'establish-home; variable there are very many ways in which the musical task associated with the variable could conceivably be carried out, but for simplicity we consider only a limited number of crude methods. Each of the possible values of the variable corresponds to a package of measures (to be discussed in the section on methods) that work more or less successfully to maintain a sense of home area. Informally, the measures for each possible value are as follows. All of the values except one (jazz-chromatic) include measures to keep the chord sequence diatonic (i.e. to avoid chromatic roots) and to keep the chord qualities diatonic (i.e. to avoid chromatic notes). Leaving aside these common measures, the various values take individual measures as follows;

• tonal: an attempt to maintain the perception of the home area on the part of the listener is carried out by the adoption of a tonal home area, to be communicated by keeping within key constraints (as explained in the common measures above) and by the use of trajectories *down* the fifths axis.

• jazz: the same as the 'tonal' measure, but with a sense of the underlying diatonic scale heightened by the use of a sevenths chord arity. (This value is distinguished from the 'tonal' value since scale-tone diatonic sevenths tend to define a diatonic scale less ambiguously than the default scale tone triads).

• modal+pedal: an attempt to maintain perception of the home area on the part of the listener is made by the co-ordinated use of a modal home area, and the use of trajectories along the scalar axis. The perception of the modal home area is facilitated by the use of a modal pedal.

• modal: the same as the 'modal+pedal' value, but with the perception of the modal home area is aided by some unspecified method other than the use of a modal pedal. This value is a place holder in a sense similar to the value 'implicit' for the 'establish home' variable.

• jazz-chromatic: the perception of the home area on the part of the listener is maintained by the combined use of a tonal home area, the use of chromatic trajectories down the scalic axis and the use of a sevenths chord arity to heighten the sense of position in the diatonic scale. (This value reflects the common use of chromatic chord sequences in sevenths, particularly in jazz, apparently without upsetting the sense of home area. This is an insecure way of maintaining a perception of the home area and was included for experimental purposes.).

When the maintain home variable has value modal or modal+pedal, there is one extra heuristic measure taken to strengthen the perception of 'home' by the listener. Namely, a default direction is provided for the scalar trajectories that is a function of the value of the mode variable. This direction is set as *down* for the dorian and phrygian modes, and *up* for the aeolian, mixolydian, lydian and locrian modes. The rationale for these choices is that for each mode, a direction is chosen so that trajectories *towards* the home area in this direction tend to avoid the harsh-sounding scale tone diminished chord. So for example, in the dorian mode, we want to avoid the diminished VI chord, so we prefer downward trajectories to the home area such as (III II I) (as opposed to upward trajectories to the home area such as (VI VII I)). Similarly, in the aeolian mode, we want to avoid the diminished II chord, so we prefer upward trajectories to the home area such as (VI VII I) (as opposed to upward trajectories to the home area such as (I II III)). In modes such as the locrian mode, where this criterion does not really apply, we take an intuitive or arbitrary choice about which direction tends to sound better. In fact, it could be argued that the role of this measure is not to strengthen the perception of 'home', but simply to make chord sequences sound 'better' by avoiding a harsh quality chord, other things being equal. Note that trajectory direction can easily be made into a plan variable accessible to the user, or left under sole control of this heuristic.

5.1.3 The code variable

We now discuss the plan variable **code**. This variable takes values which are concatenated strings of Harmony Talk code assembled in various stages by the planner. (In reality, there is a new code variable for each stage of the process, but this can be a useful image to bear in mind.) This variable can take as initial value any arbitrary Harmony Talk code before the planning starts. Such initial values can be used to preface the song to be planned with an introduction or prelude, or more typically, to set initial Harmony space settings as a context for a plan.Settings may be overidden, as we will discuss later. When the code variable has no initial value, the vanilla setting will be taken by default to hold at the start of the song. This variable is different from the others in that it covers an infinite domain, since, for example, the length of the string could be indefinite. It would be possible in principle to arbitrarily restrict the length of the string and to restrict the possible values of all Harmony Talk variables to suitable finite domains, but any restrictions that would not seriously undermine the expressivity of the planner would tend in practice to lead to unacceptably large search spaces. The solution adopted, as we will see in the sections on constraints and control is to restrict the roles of this variable in connection with constraints to **input only**. Given this restriction, the variable is allowed to take unrestricted values. This solution is similar to the treatment of arithmetic constraints in Prolog. More complex versions of PLANC might allow symbolic markers to be included amongst Harmony Talk strings to permit mid-list insertion or other 'complex' manipulation of the code by constraints. For simplicity, as we will see in the section on control, this has not been allowed, and the only operations permitted on instances of the 'code' variable are evaluation by a Harmony Talk interpreter, concatenation of code, and 'unravelling' of code during backtracking.

5.1.4 The song-length variable

The **song-length** variable has possible values 0,..,16. This variable governs the total length of the song. There is a simplifying assumption for the purposes of the return home plan that everything takes place at an even tempo. That is to say, each harmonic event is assumed to take the same period of time, with the possible exception of the final event. The limitation to a maximum length of 20 events is arbitrary.

5.2 Defining constraints for the return home plan

We now have a set of plan variables for the "return home" plan and their possible values. The next task is to define a set of constraints to use with these variables, and to indicate how to interconect them to define the "return home " plan. We will list the necessary constraints and then define each one in turn. Constraints given in the list below with no indented sub entries are considered to be 'primitive' constraints; constraints with indented subentries are compound constraints built up from the constraints given in the relevant indented list. The distinction between primitive and compound constraints simply refers to whether it has been found necessary or useful to represent a constraint as a net of more primitive constraints: this does not necessarily bear a direct relationship to the strategic importance or conceptual complexity of the corresponding concept. For example, it has been found to be useful to represent the hometype constraint as a primitive constraint, although it plays a strategic role in several plans and is conceptually complex. The complex nature of the hometype constraint is reflected not by any internal complexity but by the influence it has on other constraints through its interconnections with them. Here is the list of constraints required for the "return home" plan.

- *hometype*
- *maintain home*
- *establish home1*
    - *establish home*
    - *establish home length*
- *trajectory home*
    - *trajectory plot*
    - *not equal*
- *co-ordinate lengths*
    - *plus*

Note that the same names are used in some cases both for variables and constraints. In fact the same names are used again in some cases for generators and methods. This does not seem to be a problem in practice. It should always be clear from context or by explicit mention which kind of entity is being referred to.

5. 2.1 Recap of the nature of constraints : constraints are multipurpose

It is useful at this point to recap the main features of constraints as computational tools, together with some useful imagery. In line with the formal definition presented earlier, a constraint can be thought of in mathematical terms as an arbitrary relationship between arbitrary entities. The relationship can be of any arity (i.e have any number of arguments). We can get a useful mental picture of constraint satisfaction by imagining (following Steele (80)) constraints as physical devices, with the values for variables as coming in and out on metal pins, much as in electronic components. While the operation of a Prolog interpreter can be difficult to explain to novices, this homely metaphor can help make the operation of a constraint satisfier intuitively easier to understand to some novices. In line with this mental picture, we will use terms such as 'pin', 'terminal' and 'argument' interchangeably.

In PLANC, (unlike in Steele's (1980) work), input and output values can in general be arbitrarily complex symbolic structures.

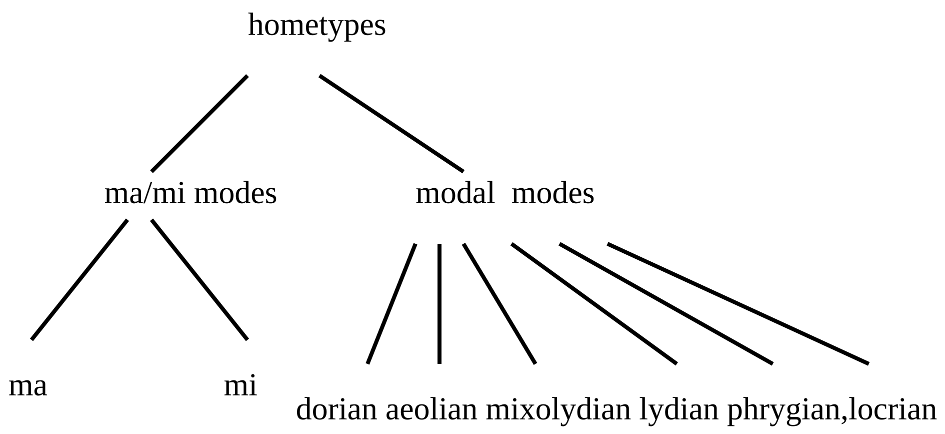
In general, a given 'pin' may be at different times an input or an output. Hence, for example, the constraint 'plus' can be used to sum two numbers, or 'backwards' to find a number that added to a given number will produce a specified sum, and so forth. Where a pin does not have a value, and there is not enough information on the other pins for the constraint to infer or compute a value, a value generator (described later) may be used to supply a value. On the other hand, if values are present on both on the pins of a device and they are inconsistent in terms of the constraint, (i.e. the relationship is overconstrained) the associated planner should effectively retract a value if possible, and cause other constraints or generators that ultimately supplied it to look for alternative values.

In summary, unlike functions in functional and procedural programming, constraints have the following multiple purposes: they can be 'run' in any 'direction', they can be used for consistency checking with supplied values, they can be used to calculate new values from partial specifications, and they may be able to supply values in the absence of any external values by use of value generators.

In keeping with the 'device' metaphor, diagrams 1 and 2 (see end of the paper) show the return home plan in diagrammatic form as a net of constraints. At this point in the paper not all matters dealt with in these diagrams have been discussed, but the diagrams may nonetheless help the reader to visualise matters at this stage in the exposition. Constraints are shown as circles with text in them, giving their name. In diagram 1, all primitive constraints are shown. In diagram 2, for convenience, to give a higher-level view of the plan, some groups of primitive constraints have been clumped into compound constraints. The pattern by which this clumping has been done corresponds to the textual hierarchy of constraints given above. Variables are shown as shaded boxes, connected by bidirectional lines to constraints. The lines correspond to links between variables and constraints along which values can 'flow' in either direction. Deliberate interconnection of lines is indicated graphically by the use of little circular connectors. Where variables or constraints are connected in this way, values must ultimately coincide. Where a line crosses another line in the diagram for purposes of layout but there is no logical connection, a little croquet hoop "jump over" sign is used. Square boxes and lines with arrows are to do with 'methods' and will be dealt with later. Any constraints in the diagrams not identified in the above tree diagram (e.g. 'small num') are actually value generators and will be dealt with later. We will now inspect each constraint in turn.

The Hometype constraint

The hometype constraint has two arguments, mode and hometype. The hometype constraint mutually constrains the possible values of these arguments as follows. If the mode variable has value major or minor, there is no choice about the hometype - it must be tonal. Similarly, if the mode is one of dorian, aeolian, mixolydian, lydian phrygian or locrian, the hometype must be modal. Conversely, if the hometype variable has value tonal , the mode variable may have value either major or minor. Similarly, if the hometype variable has value modal, the mode variable may have value either dorian, aeolian, mixolydian, lydian phrygian or locrian. Note that if the
hometype variable has a value, but the mode variable does not, there must be some way of making a choice between the possible values of the mode variable. We will deal with this in the section on value generators. Note that this constraint, like the others must be able to function under all possible patterns of availability of values: values may be supplied for both variables, either variables or neither variable. All of the above can be simply summarised simply by saying that this particular constraint requires that the relationship between hometype and mode must be in a parent-child relationship, with the mode as a leaf in the following tree;



This constraint is trivial to represent in Prolog, for example as follows (using the standard definition of member);

```
hometype(Htype,Mode):-
    htypeCatalog(Htype,Modes),
    member(Mode, Modes).

htypeCatalog(tonal,[major,minor]).
htypeCatalog(modal,[dorian,aeolian,mixolydian,lydian,phrygian,locrian]).
```

(The variables, constants and predicates in the coding correspond with the relevant plan variables, values and constraints in the obvious way as indicated by their names.) This is how the hometype constraint is implemented in PLANC.

The 'maintain home' constraint
The maintain home constraint is computationally very similar to the hometype constraint to the extent that they both enforce simple tree-relationships. The maintain home constraint has two arguments, 'hometype' and 'maintain home'. Some values of the maintain home variable are associated only with a modal hometype, and others only with a tonal hometype. In particular, the tonal maintain home values are "tonal" and "jazz" , and the modal values are "modal+pedal" and "modal" . We can summarise the maintain home constraint as follows: the relationship between hometype and home maintenance method must be in parent-child relationship in the following tree, with the maintain home method as a leaf. As with any constraint with two arguments, either both or neither values may be known initially.

Trivially, this constraint at the level of description we are currently considering can be represented in Prolog along the same lines as the previous constraint.

The 'establish home' constraint
The establish home constraint is a constraint between the hometype and establish home variables. The interrelationships between possible values are as follows. The home type may be of type 'tonal' or 'modal'. The establish home method variable may take one of three values, 'simple', 'implicit' or 'repeated'. Any of these methods can be used with either hometype, but the ordering of default establish home variable values is different for the different hometypes. We will deal with this point when we come to deal with the value generators. For the time being we will merely note that the constraint can be summarised as follows in a tree where the ordering of siblings is significant. Note that the tree is **not** a type tree, as in the previous two constraints, since the two main branches have the same leaves, but in a different order.

repeated statement

Trivially, this can constraint be represented in Prolog as follows;

establishHome(Htype,EhomeMethod):-

```
    ehomeCatalog(Htype,EhomeMethod),
    member(EhomeMethod, EhomeMethods).


ehomeCatalog( ma_mi, [simple_statement, implicit,repeated_statement])
ehomeCatalog( modal, [implicit, simple_statement,repeated_statement])
```

The 'establish home length' constraint
A very simple constraint is needed to co-ordinate the *number of events* in the 'establish home' part of the chord sequence with the value of the 'establish home' variable. We have already noted the existence of a plan variable called 'establish home length' whose value may be an integer from 0 to 4. By convention, the value of the establish home variable and the value of the establish home length variable constrain each other according to the following table, which can be used in either direction;

establish home	establish home length
implicit	0
simple_statement	1
repeated_statement	2,3,4

The corresponding constraint can be represented in Prolog very easily as follows;

```
ehLength(Ehm, Ehlen):-
    ehlenCatalog(Ehm, Ehlen),
    member(Ehlen, Ehlens).


ehlenCatalog( simple_statement,[1]).
ehlenCatalog( implicit, [0]).
     ehlenCatalog( repeated_statement,[2,3,4]).
```

The 'establish home1' constraint
The task of the compound constraint 'establishhome1' is to simply to co-ordinate the primitive constraints 'establish home' and 'establish home length'. The sole function of the ' establish home1' is to constrain the values of the 'establish home' variable in its two constituent primitive constraints to be equal. This could be represented in Prolog as follows.

```
establishhome1(Htype, Ehm,Ehlen):-
    establishHome(Htype,Ehm),
    ehLength(Ehm,Ehrep).
```

The trajectory plot constraint
The next constraint we will deal with is the primitive constraint 'trajectory plot'. The 'trajectory plot' constraint is a relation with arguments 'start degree', 'finish degree', 'trajectory length' and 'code'. When a given instance of the trajectory plot constraint in a plan is required to calculate a trajectory, the relevant direction, underlying alphabet, filter settings and so forth must be known. Hence the trajectory plot constraint needs access to the Harmony Talk code that instantiates the piece up to the point at which the trajectory applies. The trajectory plot constraint can then simulate the execution of the assembled Harmony Talk program to discover the 'state' of Harmony Space (i.e. the value of the 'trajectory affecting' Harmony Space variables) at the point at which the trajectory is to be calculated. Thus the value of the 'code' variable in a trajectory plot constraint constitutes the Harmony Talk code for the song up to the point at which the trajectory plot applies. It should be clear that the trajectory plot relation is strongly related to the trajectory plotting function used at the heart of Harmony Talk that plots Harmony Space positions in response to sequences of Harmony Talk instructions. The main difference is that in PLANC it takes the form of a (nearly) multiway constraint, as we shall now explain. Given a set of initial Harmony Talk settings (which can be calculated from the 'code' variable) , the purpose of the trajectory plot constraint is to make the trajectory start, finish and length variables consistent. (Note that trajectory plot is designed to plot a trajectory with externally specified starting point, independent of the last position of the Harmony Talk turtle.)

Trajectory plot differs from constraints looked at so far in that the fourth argument, "code " is treated strictly as an input. The reason for this is as follows. Let us consider what would happen if the code variable in the trajectory plot constraint were to be treated as a 'multiway' variable.   Assume that the start degree, finish degree and trajectory length variables already had values, but the 'code' variable did not. The job of a truly multiway constraint in such circumstances would be to calculate a string of Harmony Talk code that would put a Harmony Talk interpreter into a state that would render a trajectory with the given start degree, finish degree and trajectory length consistent. But the search space of possible strings of Harmony Talk code that could satisfy this condition is clearly infinite. The search space could be made finite by taking the measures discussed earlier: that is to say the domain of each Harmony Talk variable could be arbitrarily restricted to a finite range, and the permissable length of Harmony Talk code strings could be arbitrarily restricted. However, the value of the 'code' variable in any musical plan typically also has to satisfy the constraint that it is built up of concatenated fragments of Harmony Talk code. Each constituent fragment has a value depending on the value of other plan variables. If the 'code' variable in the trajectory plot constraint were allowed to behave as an output, it would be necessary to arrange its set of possible values to span the space of all possible concatenated code strings that could be produced at that point by the action of other plan variables. But from the point of view of the trajectory plot constraint running in reverse, this would define an impractically large search space. In other words, depending on the severity of the restrictions, the effect would be either to impose crippling restrictions on the possible songs which the planner could produce, or a combinatorial explosion. For this reason , the code variable in the trajectory plot constraint is treated an 'input only' argument. This allows the set of possible values of the 'code' variable to span an infinite space in principle without compromising the efficiency of the search. The way in which this variable is arranged to be input only will be discussed in the section on control.

The above state of affairs absolves us of the need to provide a way of calculating code from start degree, finish degree, and length. However, the multiway nature of the planner still requires that if any two plan variables out of start degree, finish degree and trajectory length are instantiated, the constraint should be able to calculate the third. This is a simple calculation given the value of the 'code' variable and given a Harmony Talk interpreter.

Trajectory home

The 'trajectory home ' constraint is a very simple compound constraint based on the trajectory plot constraint. In brief, the task of the compound constraint 'trajectory home' is simply to constrain the finish degree of the trajectory plot constraint to equal the degree of the scale I**,** and constrain the start degree to equal some degree not equal to I, using the primitive constraint 'not equal'. Thus, this compound constraint is a specialisation of the primitive trajectory plot constraint designed to deal with a harmonic trajectory from some non-home area to the home area.

The "co-ordinate lengths" constraint

The compound constraint 'co-ordinate lengths' takes as inputs the variables 'establish home length', 'trajectory length', 'emphasise final length' and 'song length'. The job of this compound constraint is to ensure that the total length of the song is the same as the sum of its parts. More precisely, two primitive 'plus' constraints are used to ensure that the total length of the piece is the sum of the number of events in the establish home section, the trajectory section and the emphasise arrival sections of the piece. Recall that for all parts of the song (except possibly the emphasise final part, as explained later), an even harmonic tempo is assumed to hold. This is the last of the constraints we need to look at for the 'return home plan'.

5.3 Generators

Our next task is to define finite generators of values for each variable. Generators can be considered as a special type of constraint: namely constraints over exactly one variable, in which the value of the variable is constrained to be a member of some set of possible values. As regards the implementation of generators, it turns out that the same piece of code can often act both as the definition of a constraint and as a generator for one or more variables involved. However, it is worth treating generators separately for two reasons. Firstly, generators play a role conceptually distinct from that of constraints and variables. Secondly, in the domain under consideration, the order in which a generator produces values is significant. We will begin by discussing the reasons for this.

5.3.1 Preferred vs correct solutions

In many constraint satisfaction and logic programming applications, we either want one solution, or all the solutions. This is not usually the case in musical planning. To see why this is so, consider the following. As already explained, for pedagogical reasons, one of the design goals of PLANC is to support the use of *sparse* plan specifications. (The term 'plan specification' refers to a specification for a piece in terms of the combination of a plan and a set of values for some of the variables.   A *sparse* plan specification is one where very few variables, or perhaps one, is given a pre-specified value.) Clearly, a sparse plan specification is typically highly underconstrained, and there will be a vast choice of resultant songs that satisfy the specification. The quality of the resultant songs may vary considerably. Since the list of all "correct" solutions is likely to be impractically long, we want some way of bringing the 'best' ones to the top. If we can find some way of knowing in advance in a more or less principled way where good variants are likely to be found, this is not only practically useful but could in itself constitute a useful kind of musical knowledge. If such knowledge could be *explicitly* characterised in some form potentially easy for novices to understand, this might be useful from an educational point of view.

5.3.2 How to find preferred solutions

It turns out that there is a very simple heuristic that seems to help find "preferred solutions". The heuristic makes use of commonplace, ordinary musical knowledge. No strong claims are made for this heuristic, but it does seem to help PLANC to behave competently. Both the heuristic, and the commonplace kinds of knowledge on which it is based are very easy to understand. The heuristic is simply to order the possible values in each of the generators, where possible, in terms of values that are generally thought to be most typical or compositionally useful in a general context. Note that this does not make sense for all of the variables, as we will note in detail later . In other cases, it is not to difficult to get a measure of agreement on a reasonable ordering. Musicans tend to agree that the ordering for **modes** in order of compositional usefulness or typicality probably begins 'major, minor, dorian,...' and ends '...locrian'. There might be disagreement over the ordering of modes in middle of the list. The possible justifications for such orderings and the possible means of communication of such justifications to novices will be considered separately in a later section.

5.3.3 Why might the suggested heuristic help?

Why might such a heuristic be useful? In Levitt's (1985) melodic improvisation program, 'new' material was balanced with a mass of default material in an effort to produce something that a listener would find interesting but 'filled out' in familiar ways so as to be capable of easy assimilation. With PLANC and its domain of strategically planned chord sequences, default behaviour may perform a roughly similar function in some circumstances, in that it could help balance a selection of unusual elements or strange combinations pre-selected by the user. On the other hand, where a user makes a stereotyped preselection of elements in the first place, the same heuristic could lead to stereotyped, 'archetypal' sounding chord sequences. Hence, the ordering heuristic we have suggested may err on the side of conservative, easily assimilable results. In the domain of chord sequences, this is probably a reasonable bias. As mentioned earlier, no strong claims are made for this heuristic for picking out "preferred " solutions , but it does seem to be useful. There are several examples of its use given in Holland (1991c). At any rate, the heuristic is simple to explain to novices and might help to draw their attention to the relative advantages and disadvantages of various musical materials.

5.4 Generators for the 'return home' plan variables

We will now briefly discuss a generator for each variable used in the return home plan. In some cases, the list of possible values given previously effectively defines the generator. In other cases, more sophisticated arrangements are required, as we will see. The generators for the first three variables, 'mode', 'hometype' and 'establish home' are all similar. Remember that *justifications* for orderings will be considered separately in an appendix.

The generator for the **mode** variable is based the list [major, minor, dorian, aeolian, mixolydian, lydian, phrygian, locrian]. Trivially, this generator could be coded directly in Prolog as as follows;

```
modeGenerator(Mode):-
    modeCatalog(Modes),
    member(Mode, Modes).

    modeCatalog( [major, minor, dorian,
        aeolian, mixolydian, lydian, phrygian, locrian]
```

In practice, to code it this way would needlessly duplicate code used to define the hometype constraint. Practically speaking, in plans that we are interested in, the mode variable is always used in conjunction with the hometype constraint. It turns out to be both efficient and perspicacious to embody the hometype constraint and generators for the mode variable in a single set of relations. Hence, the generator for the mode variable is actually implemented as two context sensitive generators, as shown below. These generators form part of the hometype constraint, seen earlier. Clearly the order of elements in the lists matters. The generators are context sensitive in the sense that the order in which values are produced is affected by the value of the hometype variable. This reflects the way in which the hometype constraint and the generator for the mode variable interact.

```
htypeCatalog(ma_mi,[major,minor]).
htypeCatalog(modal,[dorian,aeolian,mixolydian,lydian,phrygian,locrian]).
```

Note that whether the coding for the generator for the mode variable is kept separate or integrated with the coding for the hometype relation makes no difference to the solutions that PLANC will find, but it can make the search more efficient.

The generator for the **hometype** variable is formally very similar to the generator for the mode variable, but it is encoded in slightly different (though related) way for efficiency and clarity. The generator for the **hometype** variable is based on the list (tonal, modal). Clearly, this generator could be coded directly as a single list, as we suggested for the generator for the mode variable. But as in the previous case, in practice the hometype variable is always used in conjunction with the hometype constraint. Hence, it is more efficient to represent the generator for the hometype variable as part of the same two context sensitive generators already used as generators for the mode, and hence as part of the hometype constraint. The ordering of default values for the hometype variable is actually represented implicitly in the ordering of two clauses that serve as generators for the mode and hometype variables alike. As in the case of the generator for the mode variable, the alternative encodings make no difference to the set of results produced and affect only the efficiency with which they are produced.

The generator for the **maintain home** variable is based on the list (tonal, jazz, modal+pedal, modal, jazz-chromatic). In fact, for reasons very similar to those given in the case of the mode variable, the generator for the maintain home variable is actually implemented in PLANC as two context sensitive generators that also function as part of the maintain home constraint. They are context sensitive in the sense that the constraining effect of the the value of the hometype variable on the value of the maintain home variable is used to cut down fruitless search by the generator. The encoding is as follows;

```
mhome(Htype,Mhome):-
    mhomeCatalog(Htype,Mhomes),
    member(Mhome, Mhomes).

mhomeCatalog(ma_mi,[tonal, jazz, jazz-chromatic]).
mhomeCatalog(modal,[modal+pedal, modal]).
```

There are two generators for the **establish home** variable, but unlike multiple generators we have seen so far, they are required to be distinct for *conceptual* reasons. It turns out to be useful to have the *ordering* of values supplied by the establish home generator (as opposed to the set of possible choices provided by the generator) dependent on the value of the hometype variable. This simply reflects the musical judgement that judgements about the typicality or preferredness of ways of establishing home seem to be affected by whether the context is modal or tonal. Making the *ordering* of a generator sensitive to the value of some variable is a useful step to take to help 'float' preferred solutions to the top. This measure is easily implemented, as follows; Practically speaking, in plans that we are interested in, the establish home variable is always used in conjunction with the hometype constraint. This gives us the opportunity, if we wish, to make the generator for the establish home variable context sensitive to the value of the hometype variable. We simply build two context sensitive generators into the establish home constraint, like so;

```
establishHome(Htype,EhomeMethod):-
    ehomeCatalog(Htype,EhomeMethod),
    member(EhomeMethod, EhomeMethods).


ehomeCatalog( ma_mi, [simple_statement, implicit,repeated_statement])
ehomeCatalog( modal, [implicit, simple_statement,repeated_statement])
```

Note that the establish home 'constraint' does not actually constrain the values of the hometype and the establish home variables at all: it is simply a mechanism for mediating the ordering effect we have been discussing. However, for terminological and expositional convenience, we will continue to refer to it as a constraint.

The generator for the **establish home length** variable is technically similar to the first three generators. It is based on the list (0,1,2,3,4). Conceptually, the generator for this variable can be represented in Prolog as follows;

```
ehrepGenerator(Ehrep):-
    smallnum(Smallnums),
    member(Ehrep, Smallnums).

    smallnum([0,1,2,3,4]).
```

However, much as in the first three examples. the coding for the generator for establish-home repeats is combined with the coding of the generator for the establish home constraint. This gives rise to three context-dependent generators as follows;

```
ehrepCatalog( simple_statement,[1]).
ehrepCatalog( implicit, [0]).
     ehrepCatalog( repeated_statement,[2,4,3]).
```

The next four variables all have very simple generators. The **away-point** variable uses the following generator;

```
diatonic_degreeCatalog( [i,ii,iii,iv,v,vi,vii]).
```

This ordering is arbitrary, since it does not make immediate sense to refer to degrees of the scale as being more or less useful or typical as chord roots, although various orderings could no doubt be put forward.

Conceptually, **trajectory-length** variable could use the following generator;

```
middlenum( [0,1,2,3,4,5,6,7,8,9,10,11,12]).
```

But in fact no generator is needed for this variable in the return home plan.
The order in which constraints are satisfied in the return home plan is such that the first constraint to be satisfied in which this variable is involved is always the trajectory plot constraint. The trajectory plot constraint is so arranged that if it is underconstrained, generators for the other variables (start degree and finish degree) are always consulted first, so that the value for the trajectory length is always calculated from the other variables if it does not already have a value. This is done simply to cut down unneccessary search.

The **emphasise-final-length** variable uses the following generator;

```
smallnum([0,1,2,3,4]).
```

The ordering is more or less arbitrary.

The **song length** variable in the plan return home could, fairly arbitrarily, use the following generator;

```
    bignum([0,1,2,3,4,5,6,7,8,9,10,11,12,12,14,15,16]).
```

However, as in the case of the 'trajectory length' variable, no generator is actually used for this variable in the return home plan. As we mentioned earlier, the 'co-ordinate lengths' relation in the return home plan constrains the total length of the song to be the same as the sum of its parts (see diagram 1). To this end, the song-length variable is connected to the "equals" terminal of a 'plus' constraint. Now, a feature of standard Prolog is that the "plus" constraint is not truly multiway, and that values for the two addends must be provided. But since Prolog allows us to order the firing of the constraints, we can simply arrange things so that the length of the 'establish home' part of the song and the 'trajectory home' length are always known by the time this constraint is considered. Hence the only possible unknowns are the total length of the song and the length of the' emphasise arrival' part of the song. The total length of the song could be quite variable, whereas the 'emphasise arrival' variable has a fairly small search space. Hence to save needless search, if no value has been provided by the user for the variable 'song length', its value is calculated from the 'co-ordinate lengths' constraint..

Of course, in cases where generators are not actually used, they could nonetheless be easily supplied to make the plan read more declaratively, and to make the program work (albiet inefficiently) if the search order were changed.

Finally, the generator for the **code** variable is, in effect, the vanilla setting. This value is provided not explicitly, but procedurally (although it could easily be provided explicitly as well). This procedural provision happens without the need for any specific measures to ensure it, as follows. Constraints that use the 'code' variable normally use it to calculate a harmony space state. To do this, a Harmony Talk interpreter is required. But the Harmony Talk interpreter implicitly prefaces all Harmony Talk strings with the Harmony Talk code that produces the vanilla setting. This completes our examination of the generators required for the 'return home' plan.

5.4 Defining methods for the return home plan

We have now defined plan variables, constraints and generators for the "return home" plan. Our task in this section is to define the 'methods' needed for the return home plan. Recall that methods are low level procedures which take values of plan variables as input and which produce strings of Harmony Talk code as output. Note that unlike constraints, methods always have distinguishable inputs and outputs. The outputs of the methods are assembled to produce a Harmony Talk program describing the fully instantiated plan. The *order* in which the outputs of methods are calculated matters, but discussion of this point will be deferred until the next section, on control of the planner. For now we will just assume that the outputs of methods are calculated in the order that they contribute to the final piece. To understand each method, we need to indicate its possible inputs, its possible outputs, and its mapping rules.

In some cases, the code contributed by a method represents a considerably simplified or partial account of the concepts represented by the values of corresponding input variables. Below we give a list of methods used in the return home plan (in bold), together with (indented underneath each method) a list of the variables required as input to each method. Justifications for the mappings from plan variable values to Harmony Talk code is given in an appendix.

hometype method
    mode
maintain home method
    mode
    maintain home
establish home1 method
    establish home
    establish home length
trajectory plot method
    start degree
    trajectory length
    code
emphasise arrival method
    emphasise arrival length
    song length
    code

We will now look at each method in turn. It may help to refer back to diagram 1.

The hometype method
The hometype method is associated with the hometype constraint. It has a single input, the mode variable. The hometype method maps the value of the mode variable (with possible values major, minor, dorian, aeolian, mixolydian, lydian phrygian,locrian) onto a fragment of Harmony Talk code that sets the harmonic

centre and window shape accordingly. The mapping from mode name to harmonic centre is done using a simple lookup table. The table is as follows;

```
modename([[major,i],[minor,vi],[dorian,ii],[phrygian, iii],
[lydian,iv],[mixolydian,v],[aeolian,vi],[locrian,vii]]).
```

So for example,where the mode variable has value dorian, the code fragment produced by the hometype method is as follows;

```
set harmonic-centre  ii
```

The other part of the job of this method is to deal with the mapping from mode name to window shape. This requires no explicit code in most cases, since for most modes the default diatonic window shape is required, and this is provided automatically by the vanilla setting. The only mode calling for explicit setting of the window shape is the minor mode. A simple test on the value of the mode is used to control the insertion of the appropriate code fragment. So for example, where the mode variable has value minor, the hometype method produces the following code;

```
    set window-shape  harmonic_minor
set harmonic-centre    vi
```

Maintain home method
The 'maintain home' method is associated with the 'maintain home' constraint and has two inputs, the variables 'maintain home' and 'mode'. The code contributed by this method represents a simplified, partial account of a limited number of ways of maintaining a user's perception of the home area. For an informal description of the packages of measures corresponding to each distinct possible value of the maintain home variable, see the earlier section on plan variables. For a justification of the mapping, see the appendix on that topic. The possible values of the maintain home variable are as follows: (tonal, jazz, modal+pedal, modal, jazz-chromatic). As we noted in the section on the variables, all of the packages except one (jazz-chromatic) call for measures to keep the chord sequence diatonic (i.e. to avoid chromatic roots) and to keep the chord qualities diatonic (i.e. to avoid chromatic notes in the chord). To map this idea into code, the following piece of Harmony Talk code is contributed irrespective of the value of the maintain home plan variable;

```
set quality-lock         off
set chromatic-filter     on
```

This code when interpreted by the Harmony Talk interpreter corresponds to a set of measures to make sure that trajectories will avoid chromatic root notes and that chords are not locked into a rigid shape that violates key boundaries. (However, these measures are easily overturned by later methods, as we shall discuss later.)

The remainder of the code contributed by this method is based on a table look up on the value of the maintain home variable. The result is slightly qualified in some cases by the value of the mode variable. For an informal outline account of each mapping, see the descriptions given in the section on plan variables. For a justification of the mapping, see the appendix on the justification of the methods. Here are the fragments of code associated with each possible value of the maintain home method.

tonal method

```
set trajectory-axis fifths
set trajectory-sense    down
```

jazz method

```
set trajectory-axis fifths
set trajectory-sense    down
set arity               sevenths   (alternative value 'ninths' also available)
```

modal method

```
set trajectory-axis scalar
set trajectory-sense    Direction    (depends on value of 'mode')
set pedal               on
```

modal+pedal method

```
set pedal               on
set trajectory-axis     scalar
set trajectory-sense    Direction    (depends on value of 'mode')
```

jazz chromatic method

```
set chromatic-filter     off
set trajectory-axis     scalar
set trajectory-sense    down
set arity               sevenths
```

The two minor ways in which the table look ups are inflected require brief clarification, as follows. Firstly, in the 'jazz' case, two possible values are given for the chord arity. Either of these values are satisfactory in the case where the maintain home variable has value 'jazz', so the method is arranged to ensure that they are offered in turn by the normal operation of backtracking. Note that the method ignores whatever chord arity is currently set, although it could easily be arranged to take it into account. Secondly, in the case where the maintain home variable has values modal or modal+pedal , the Harmony Talk value used for the Harmony Talk variable 'trajectory sense' in the code fragment depends on the value of the plan variable 'mode', as described in the section on variables. The relationship can be represented in Prolog using the following table;

```
preferredTrajectoryDirection([up,[aeolian,mixolydian,lydian,locrian]).
preferredTrajectoryDirection([down,[dorian,phrygian]).
```

Before concluding the discussion of the maintain home method, this is a useful moment to broach the topic of 'subgoal interaction'. At the beginning of the discussion of the method for maintain home, we showed the following fragment of code, contributed for all values of the maintain home variable except 'jazz-chromatic'.

```
set quality-lock        off
set chromatic-filter    on
```

As the prototype version of PLANC happens to be coded, it does not bother to explicitly supress the above code even when the value of the maintain home variable is 'jazz chromatic', which requires different settings. The code contributed in addition to the above fragment in the 'jazz-chromatic'case is as follows:

```
set chromatic-filter     off
set trajectory-axis     scalar
set trajectory-sense    down
set arity               sevenths
```

The net effect of the concatenated fragments is that the chromatic filter is set to 'off', overiding the initial setting to 'on'. The performance of an action in a plan only to allow it to be immediately undone by an other part of the plan is inelegant. This behaviour could be easily remedied in the present case by a simple test for the value 'jazz chromatic' before the contribution of the first code fragment, but it is a useful illustration of the absence of provision for methods to protect the results of their actions (e.g. resultant Harmony Space states) against later actions by other methods. We will come back to this topic later.

The establish home method

The 'establish home' method is associated with the 'establish home' constraint and takes as inputs the variables 'establish home' and 'establish home length'. The possible values of the 'establish home' variable are (simple, implicit, repeat), and the possible values for the 'establish home length' variable are (0,1,2,3,4). The code contributed by this method represents a simplified account of a limited number of ways of *establishing* (as opposed to maintaining) the location of the home area to the ears of the listener. For informal details of the packages of measures corresponding to each distinct possible value of the establish home variable, see the previous section on plan variables. The value of the establish home variable together with the value of the establish home repeat variable simply select the appropriate fragment of Harmony Talk code and the appropriate numeric value to set up the appropriate number of clicks on the home area (see code fragments below). The fragments of Harmony Talk code are as follows, indexed by the relevant values of the maintain home variable.

simple statement

```
set turtle-position      I
hold
click                    1
```

repeated statement

```
set turtle-position      I
hold
click  establish-home-length
```

implicit
there is no code associated with this value of the establish home variable, and no action to take.

Like the maintain home method, the establish home method is crude but works adequately, as can be seen in the scenarios of Holland(1991c).

The trajectory plot method
The penultimate method is the trajectory plot method. This is associated with the 'trajectory plot' constraint and takes as inputs the variables 'away' and 'trajectory length'. The method simply slots these value for these variables into a Harmony Talk code fragment for generating the corresponding trajectory as follows;

```
set          turtle-position     away
click        trajectory length
```

(PLANC as currently implemented does this very slightly differently, since is uses a 'click until' construction. However, this makes no material difference.)

The 'emphasise arrival' method
The establish home method is associated with the 'co-ordinate lengths' constraint, and takes as input the plan variables 'song length' and 'emphasise final length'. It also takes as input the chord arity derivable from the Harmony Talk code up to the point in the piece at which this method contributes its code. The job of the emphasise arrival method is as follows. Where the value of the emphasise final length variable is zero, the method does nothing. Where the value of the emphasise final length variable is non zero, the method builds a duration list that will cause all notes in the chord sequence to be played with a unit duration, except for the final note which will be held for the requisite duration given by the value of 'emphasise final length'. The method also looks at the Harmony Talk code assembled for the song so far to inspect the chord arity. In the case of the use of sevenths as the basic harmonic material (chord arity sevenths), the method causes arrival at the home area can be restated with chord quality altered to the more stable sounding major sixth, as opposed to the default but tense sounding major seventh. This completes the description of the methods required for the return home plan.

5.4.1 Constraints affecting order of calculation
Our final task is to show how search is controlled in PLANC. The order in which variables, constraints and methods are considered must be controlled to make the planning process feasible and efficient. There are various constraints (in the everyday sense) affecting order of calculation, which we will now consider.

• The target Harmony Talk code must of course be lexically ordered to correspond with the ordering in time of the planned final musical product. We will refer to this lexical ordering as the "lexical ordering of the song". This is easily arranged: we simply arrange for the outputs of the various methods to be concatenated in the desired logical order as a list operation. Note that this does not in itself put a constraint on the order in which ouputs for the methods need to be calculated, but it does have an effect when considered in combination with other factors.

• For reasons discussed earlier, the code variable in the 'trajectory plot" constraint is treated as an 'input only' argument.

• For the value of the code variable to be accurately known at any point, the output of all methods that might contribute code fragments to the song up to that point must be known.

• It follows from the above two considerations that when a 'trajectory plot' constraint is considered, all of the methods that might contribute code fragments to the song up to the point at which the trajectory constraint applies must be considered first.

• Unlike constraints, methods have well-defined inputs and outputs, so before the output of a method is required, values for its inputs must have been calculated.

Putting these considerations together, some constraints (in particular, trajectory plot) require as input the state of the song at some relevant point in the song. We need to make sure that we never get into a position where we try to satisfy one of these constraints before all of the methods that could conceivably contribute trajectory affecting code at earlier points in the lexical ordering of the song have made their contribution. (The term 'trajectory affecting code' is defined in Holland (1991d).) One simple way to achieve this is to make sure that all of the methods are calculated more or less in the order that their contributions appear in the final piece. In order to achieve this, the constraints need to be considered in an order that will ensure the values required as inputs by the methods are found before the corresponding method is calculated. Note that it is trivially easy to control the order in which constraints are considered in Prolog by making use of its procedural semantics: in concrete terms by ordering the relations on the logic program. (Sterling and Shapiro,1986). But note that this ordering of considering the constraints does *not* mean that the Harmony Talk code is assembled without regard to events that may be due to happen 'later' in the piece: the automatic action of backtracking will make sure that the Harmony Talk code is unravelled and rebuilt as many times as necessary to fit with details occuring anywhere the piece. We will come back to this point in a later.

To satisfy these constraints on the order of calculation, the top level constraints and the methods are satisfied in the following order;

```
hometype_constraint
hometype_method

maintain_home_constraint
maintain_home_method

establish_home_constraint
establish_home_method

trajectory_home_constraint
trajectory_home_method

emphasise_arrival_constraint
emphasise_arrival_method
```

The more or less completely regular grouping of major constraints and methods is convenient in the 'return home' plan, but there is no necessity for such a rigid arrangement in general as long as the constraints on ordering outlined earlier are satisfied.

The restriction that the code terminal in the trajectory plot constraint should be 'input only' is carried out simply by the ordering of satisfaction of the constraints and by ordering the consideration of generators within the constraints.

6 Related work

The idea of describing a piece of music as a network of constraints was proposed by Minsky (1981). The earliest exploration of this idea in detail, (applied to representing musical styles) appears to be Levitt's (1985). Levitt's goals and methods differ from those developed here in number of respects. Firstly, Levitt worked on describing style rather than trying to characterise compositional ideas, (though no completely hard and fast line can be drawn). Secondly, Levitt was not trying to address the problem of making his characterisations comprehensible to novices. Thirdly, at the time of Levitt's work, there were technical difficulties in building practical constraint satisfaction interpreters, so it is unclear how much of the work was actually implemented using constraint satisfaction as opposed to functional programming. Levitt's work was an important inspiration for the work on the musical planner.

Kemal Ebcioglu (1986) has carried out highly impressive work on detailed representation of highly specific kinds of musical knowledge using his own logic programming language. His program harmonises chorale melodies in the style of Bach to a very high standard. However, this impressive work has very little relevance to our present concerns. The knowledge used would almost certainly be impenetrable to any novice, and Ebciouglu's program focuses firmly on the very specialised taks of harmonisation of chorales in the style of Bach. Consideration of this work would take us beyond the scope of the present paper.

Balaban (undated) is looking at ways of using logic programming to formally explicate common musical terminology. The long term goals are "to bring the study of music to the level of a formal field, where models can be compared and their properties proved" (Balaban, undated). In the long term ,this work may prove of great importance in educational applications, but it is hard to judge the relevance of the work in its current stages.

Vandenhede(1986) carried out some experiments on using Prolog II specifications to create pieces of music that deliberately created tonal ambiguity and induced or avoided the pereception of metre. The musical ideas used are probably sufficiently pithy and simple to be communicated easily to novices with knowledge of the relevant concepts, and are expressed in a more or less declarative form.

7 Limitations and criticisms

7.1 Conceptual limitations

PLANC is a way of finding and exploring the kind of knowledge needed for musical planning, and studying how this knowledge interacts, rather than an attempt to put this knowledge into a definitive form.

The mapping from the informal statements of the plans to plan variables and methods is rather crude and limited. Still, it permits a start to be made on examining how musical goals and musical means can interact.

Leaving aside the end of the chord sequence in most of the plans, it is assumed that the harmonic sequences take place at a regular, even tempo.

If the planner could be multiway in all respects, in principle it would be able to act as a recogniser, accepting notes as musical input and deciding whether or not this satisfies a musical plan constraint specification. More usefully, such a planner could in principle work simultaneously bottom up from fragments of notes and top down from a musical plan, meeting in the middle. For practical reasons already discussed, the planer is not multiway in all respects. The planner is capable of this kind of behaviour with small details only where they are mentioned explicitly as plan variables.

PLANC as it stands has *not* been designed to be useable as a stand-alone tutor. PLANC is a necessary prerequisite for such a tutor, since it has been necessary first to formulate the relevant knowledge and to find out how to represent and use it.

PLANC does not make provision for novices to construct or modify plans themselves, for example by adding or subtracting constraints. In the section on further work, we discuss the possibility of constructing a version of PLANC that could be programmed graphically by novices.

The issue of protecting subgoals from interference by other subgoals, except by the action of constraints to co-ordinate them at a high level, has not been explored.

We do not attempt to tackle this problem here because it is out of the line of our main concerns. There does not seem to be any reason why an extended version of PLANC might not mark particular actions as protected, or use any one of a variety of approaches. This issue has been explored in AI planning in general.

Although the maintain home method ensures the correct settings at the point in the piece at which it operates, it cannot protect these settings against being overridden in turn by methods operating later in the piece. PLANC gives no explicit means for a method to protect settings it has made against possible actions by other methods acting later in the final piece. This is a special case of a well known problem in the AI area of Planning referred to a mutual interference between subgoals. It appears that existing techniques could be adapted to solve this problem in PLANC, but this is beyond the scope of our purposes. In practice, mutual interference does not appear to cause major problems in PLANC, because the high level constraints tend to co-ordinate the various methods to work together constructively.

7.2 Limitations of implementation

PLANC was implemented as a rapid prototype and its coding is not particularly neat or elegant.

In some trivial cases, features of two slightly different implementations of the "return home" plan have been discussed as though they were part of the same implementation, but the relevant features could trivially be combined in one implementation.

The implemented prototype Harmony Talk interpreter uses a dialect of Harmony Talk more primitive than the dialect used by PLANC, so the translation from Harmony Talk to note level cannot be mechanically checked using the implemented prototype Harmony Talk interpreter. The dialect used by PLANC is very similar to the dialect discussed in Holland (1991d), but with minor cosmetic differences due to later revision for purposes of exposition. However, the Harmony Talk programs produced by PLANC are so simple that it is very easy to be satisfied about their behaviour using pen and paper or hand simulation. This can be checked with the example programs assembled by PLANC as illustrated in Holland (1991c).

Calls to the 'trajectory plot' constraint are currently hand simulated, as discussed in the section on constraints.

8 Further work

PLANC does not make provision for novices to construct or modify plans themselves, for example by adding or subtracting constraints. One interesting possibility would be to construct a version of PLANC that could be programmed graphically. This might be arranged as follows. Individual constraints could be assembled into nets of constraints, of the sort seen in diagrams 1 and 2. Graphic constraints would correspond to fragments of Prolog code to be added to a Prolog program. Interconnecting constraints graphically would correspond to the common naming of variables in the various program fragments. One of the chief problems would be for the graphic planner to arrange the clauses into a suitable order that would keep the search space tractable. This problem might be tackled using suitable heuristics. For example, one heuristic might be to consider constraints in the order in which they contributed to the final piece. Graphic music programming has previously been considered in the case of functional programming (Desain and Honing, 1986) (though not implemented at the time of writing) and in the case of data flow (Sloane,Levitt,Travers, So and Cavero 1986).

An expanded version of PLANC might have the various kinds of knowledge used marked for different kinds of justification to the user. For example, some of the justifications could be illustrated by means of demonstrations or experiments in Harmony Space. On the other hand, justifications based on typicality arguments could be backed by an appeal to a library of examples.

It might be instructive to design a version of PLANC that did not use methods, but used constraints uniformly throughout. Interesting points of comparison would be to compare the expressivity and flexibility of the resulting system, balanced against its potential ability to communicate relevant knowledge to novices.

9 Conclusions

• Ways have been demonstrated of expressing in computational terms the plans informally presented in Holland (1989) and Holland and Elsom Cook (1990).

• A planner based on constraint satisfaction and Harmony Talk concepts has been designed and implemented that can use the musical plans expressed in computational terms to compose chord sequences that instantiate the plans.

• The planner has been implemented in such as way that default values are readily available for any element of any plan. This allows beginners to experiment with the elements of a plan in isolation or in any combination to find out their effects without being pressed to specify elements they may not yet understand or wish to focus on. However, the interaction of the constraints and the default value generators is such that defaults do not appear to be provided in a "hard-wired" manner; they respond flexibly to whatever constraints are in force at any point in the piece.

• The planner permits the student to work bottom up or top down: i.e. it is possible to specify low level matters of detail while postponing decisions about high level strategic choices as well as vice versa. This reflects the varied ways in which composers seem to like to work (Sloboda 1985).

• The planner is able to find alternative versions of songs that fit a given plan specification, where possible.

• There are at least three layers of knowledge in the planner that interact to produce musically 'intelligent' or knowledgeable behaviour: the net of constraints, the generator orderings and the code fragments in the methods. One virtue of the design of MC is that no layer is particularly complex or unmanageable: the interplay of the layers allows each layer to be comparatively simple.

• When composers are asked how they perform some open-ended creative task in composition, they usually give vague or highly fragmented, very partial replies. PLANC shows how three layers of very simple knowledge are adequate to go about such a task in a strategic but flexible manner.

• The design makes it very easy to use musical plans in a hierarchical or recursive fashion. This makes it possible to nest musical plans in order to achieve musical goals and model more complex musical behaviours.

• As a result of using the constraint-based programming paradigm, the traditional advantages of declarative programming accrue. Briefly, these are as follows. Knowledge is recorded relatively independently of any particular use. Declarative programming is "economical with knowledge. In the present case, it allows repeated use to be made in different contexts of simple, ordinary pieces of musical knowledge. For example, the heuristic that "scalic trajectories moving towards a harmonic centre, given a free choice of side to start from, will in general sound better if they choose it to avoid the harsh diminished chord", need only be recorded once, but can be usefully employed in different plans in different contexts.

• The declarative statement of the plans forms a good foundation for a musical planner that could explain the reasoning behind its inferences to a student. It makes the knowledge potentially very easy to manipulate and reason about.

• When represented formally, plans can record explicitly knowledge and heuristics about how to co-ordinate musical materials and constraints to satisfy commonplace musical goals or subgoals. For example, the net of constraints in the "return home" plan can be seen as a series of subgoals. Top level subgoals are 'establish home', 'maintain home', 'trajectory home' and 'emphasise arrival'. Depending on pre-specified strategic or low-level details, the planner can co-ordinate appropriate ways of satisfying each subgoal.

• PLANC makes it possible to explore the coherence and consistency of generalisations about how varied musical materials can be used to serve general musical goals. The formal statement of a musical plan or analysis can be a good way of exploring the original informal conception. It can be very hard to forsee the ramifications of an informal statement of constraints. Unexpected results from playing with a formal version may force refinement of the original conception. The simultaneously declarative yet generative nature of the formalism make the formalisation particularly informative. However, this kind of exploration requires manipulation of the programs representing the plans, not just their inputs. Hence it tend to be reserved for the programmer, rather than the naive user.

• The musical plans as implemented demonstrate how little musical knowledge is required for moderate competence in the domain, or to put it another way, how efficiently they use the little musical knowledge that they do have.

• The planner allows strategic notions about chord sequences to be expressed in forms that exploit the ease of communication of basic harmonic concepts to novices with access to harmony space .

• The planner in general is designed to allow novices to begin tackling interesting, motivating, "high level" musical tasks without requiring a lot of previous musical knowledge.

• The planner relates seemingly distantly musical materials to common musical ends. It makes use of knowledge of the uses, limitations and interrelations of musical materials

• Stating the plans in computational form allows the plans to be unfolded faithfully using materials of interest to individual students into sample ideas and fragments, avoiding the ambiguities of verbal formulations (while not devaluing the inspirational value of the verbal forms).

• For musical novices to be in a position to understand PLANC, the main prerequisite is a grasp of Harmony Space concepts. Novices familiar with Harmony Space concepts appear to be well placed to understand the basis for

PLANC's operation, at least in outline   Although no mechanisms have yet been designed for PLANC to act as an active tutor, most of the knowledge explicitly represented in PLANC appears potentially well suited to communication to novices.

• Note that PLANC exists only in the sense of a set of implemented reusable constraints, generators and methods, and a framework and set of principles for using them together - it has no separate existence.

References

Balaban, M. (undated) A formal basis for research in theories of Western Tonal Music. Technical Report 85-21, Department of Computer Science, State University of New York at Albany.

Balzano, G. J. (1980) The Group-theoretic Description of 12-fold and Microtonal Pitch systems. In *Computer Music Journal* Vol. 4 No. 4. Winter 1980.

Cohen and Feigenbaum (1982) The Handbook of Artificial Intelligence. Pitman, London.

Desain, Peter and Honing, Henkjan (1986) LOCO: Composition Microworlds in Logo. Proceedings of International Computer Music Conference.

Ebcioglu, Kemal (1986) An expert System for Harmonising Four-part Chorales. Proceedings of International Computer Music Conference.

Holland, S. (1989) Artificial Intelligence, Education and Music. Doctoral dissertation (available as CITE technical report no 88) Open University, Milton Keynes.

Holland, S (1991a) Preliminary report on the design of a constraint-based musical planner. Technical Report, Department of Compting Science, University of Aberdeen, Scotland. (This paper.)

Holland, S (1991b) A family of musical plans. Technical Report, Department of Compting Science, University of Aberdeen, Scotland.

Holland, S (1991c) Scenarios illustrating the educational uses of a musical planner. Technical Report, Department of Compting Science, University of Aberdeen, Scotland.

Holland, S (1991d) Harmony Talk - an object language for a musical planner. Technical Report, Department of Compting Science, University of Aberdeen, Scotland.

Holland, S. and Elsom-Cook, M. (1990) "Architecture of a knowledge-based tutor for music composition. In Guided Discovery Learning Systems", Ed. Elsom-Cook, M. Chapman and Hall, London.

Levitt, D.A. (1985) A Representation for Musical Dialects. Unpublished PhD Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology.

Longuet-Higgins, H.C. (1962) Letter to a Musical Friend. In Music Review, August 1962 pp 244-248.

Longuet-Higgins, H.C. (1962b) Second Letter to a Musical Friend. In Music Review, November 1962.

Minsky, Marvin (1981) Music, Mind and Meaning. In Computer Music Journal Vol.5 No.3 p.28 - 44

Sloane, B., Levitt, D. Travers, M. So, B. and Cavero, I (1986). Hookup 0.80. Prototype undistributed computer software for the Apple Macintosh. Media Laboratory, MIT, Cambridge, Mass.

Sloboda, J. (1985). *The Musical Mind: The Cognitive Psychology of Music*. Clarendon Press, Oxford.

Steele, G.L. (1980) The definition and Implementation of a computer programming language based on Constraints. AI Lab, MIT (Doctoral Dissertation).

Stefik, M.J. (1980) Planning with Constraints. Computer Science Dept. Stanford University. (Doctoral Dissertation).

Sterling, L and Shapiro, E. (1986) The Art of Prolog. MIT Press, Cambridge, Mass.

Vandenhede, Jan (1986) Musical experiments with Prolog II. Proceedings of the International Computer Music Conference, 1986. CMA, San Francisco.

Van Hentenryk, P. and Dincbas (1987) Forward checking in Logic Programming. In Proceedings of Fourth International Conference on Logic programming, pages 229-256 , MIT Press.