Volume 13 • THE COMPUTER MUSIC AND DIGITAL AUDIO SERIES

KNOWLEDGE-BASED PROGRAMMING FOR MUSIC RESEARCH

John William Schaffer

Deron McGee

A-R Editions, Inc.

Madison, Wisconsin

Contents

Preface vii

▨ Chapter 1 Introduction 1

Background 3

Conclusion 23

▨ Chapter 2 A Prolog Overview 25

Background 25

Basic Overview of the Prolog Structure 28

The Prolog Language 44

▨ Chapter 3 Introduction to Logic Programming 71

Introduction 71

Basic Problem-Solving Strategies 71

Search Strategies 76

▨ Chapter 4 Programming Knowledge-Based Systems 111

Introduction 111

Main Components of a Knowledge-Based System 112

A Sample Expert System: Nonharmonic Tones 114

Semantic Networks 136

Frames 138

Conclusion 141

▩ Chapter 5 A Generalized Model for Encoding Musical Data 143

Introduction 143

Formal Overview of the Data Model 145

▩ Chapter 6 A Knowledge-Based Intelligent Tutoring Engine 187

Introduction 187

Harmony Coach: A Knowledge-Based ITS for Diatonic Harmony 190

Conclusion 226

▩ Chapter 7 KIRS: A Knowledge-Based Simulation 227

Introduction 227

Part 1: Foundations 229

Part 2: Constructing the Model 239

Part 3: Implementing the Model 282

Part 4: Evaluating the Theory 303

Part 5: Conclusion 308

▩ Appendices

▩ Appendix 1 Tonal Harmonic Model: Middle-Tiered Control Heuristics 317

▩ Appendix 2 Tonal Harmonic Model: The THM Program Code 323

▩ Appendix 3 Implication-Realization English Rule Base: After Narmour 363

▩ Appendix 4 KIRS Program Code 371

▩ Bibliography 387

▩ Index 397

ONE

Introduction

Musicians dramatically expanded the application of computer technology for research and instruction with the advent of the first commercially available microcomputers in the late 1970s. Early applications were largely dictated by individuals' access to computer hardware: the capabilities of the machines in terms of processing speed, internal memory, and external storage and the availability of appropriate programming languages. Although this newfound accessibility alleviated many of the problems associated with large mainframe and minicomputer systems, many constraints remained, such as slow processing speeds and memory limitations. For example, only a few general purpose programming languages, such as BASIC and Pascal, were available on early microcomputer platforms, and, because these languages rely primarily on numerical representational schemes, many conceptual abstractions had to be overcome to facilitate meaningful musical applications. Some musical projects were highly amenable to such representation, such as generating analyses using set theoretic operations, whereas others, such as creating analyses using implication-realization models, were not.

Programmers using languages such as BASIC, C, or Pascal frequently rely on the sequentially based "divide and conquer" approach to programming, in which a problem is repeatedly divided into smaller problems until an explicit step-by-step method, or *algorithm*, for attaining a solution is found. Alexander Brinkman (1990) defines algorithm as "a detailed, unambiguous set of instructions for accomplishing a particular task" (p. 917). Algorithms might be defined to count occurrences of surface-level phenomena, such as chords or chord progressions, or to compute statistics on the basis of the results of such analyses. Designing algorithms to account for more intangible aspects of musical styles, such as experience, perception, and higher-level organization, proves considerably more difficult. Fortunately, there are models to assist in dealing with these sorts of problems.

Developments in the field of AI provide such tools, specifically ones designed for investigating how people acquire, store, and employ knowledge. One line of development facilitates the concept of *knowledge-based programming*: creating programs that attempt to apply human knowledge and problem-solving heuristics for dealing with various nonlinear sorts of problems. In such systems, knowledge is usually stored in the form of *rules* and *relationships* used by the computer program to deduce solutions to a problem logically, thereby modeling, or at least mimicking, human reasoning. Knowledge-based programs are not new, but a lack of sophisticated AI programming languages designed to work in microcomputer implementations limited their use in music research until recently.

The programming language Prolog is built around a *declarative* model, as opposed to more traditional *procedural* languages (such as Pascal and BASIC) that require programmers to state explicitly each detail of a program and the specific order in which the instructions are to be executed, thereby forcing the programmer to concentrate on *how* the program solves problems. Prolog programs are primarily concerned with describing and defining relationships between objects. In other words, they are concerned more with *what* is the nature of a particular problem and *what* are the various relationships of the relevant components than with the procedural details of how that output is obtained. Prolog uses the relationships defined by the programmer to search for solutions to questions posed to the system while working out many of the procedural details on its own. In addition, Prolog primarily processes and manipulates *symbols*, a significant step beyond the numerical methods used by most traditional general purpose languages. Symbols may be anything from characters, words, or sentences to representations of graphic images, an asset that enables the programmer to define properties and relationships among various symbols and to construct significant inferences between them. This ability makes Prolog a valuable tool for much AI research, particularly projects involving natural language processing, the development of expert systems, intelligent tutorial systems, intelligent user interfaces, and other types of knowledge-based systems.

BACKGROUND

Before we can attempt to learn how to write our own declarative programs or even grasp the nuances of AI languages such as Prolog, we need a clear understanding of the nature of AI research and, more specifically, knowledge-based systems. To attempt a foray into the field without an awareness of its history—both its successes and its failures—would prove no more fruitful than trying to undertake a musical analysis without understanding any music theory—possible, perhaps, at a very rudimentary level but certainly a painful and unproductive way to proceed! The remainder of this chapter, then, is devoted to such an examination. We look briefly at the history of the field, highlighting some of the more significant events, and end by taking a peek at some of the more promising music-related knowledge-based applications.

Artificial Intelligence: Historical Views

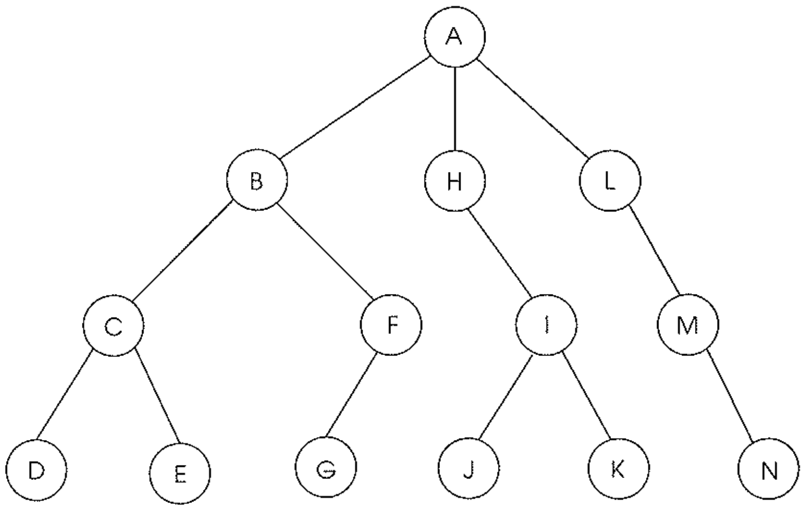
Relatively few people are aware of the precise nature and potential use of knowledge-based systems, in part because AI scientists themselves are still attempting to understand and define just what constitutes intelligence. Only when this task is accomplished can we attempt an adequate definition. For the present, perhaps it is best for us to examine several definitions. One of the more interesting viewpoints on AI is presented by John Haugeland (1985):

> Artificial Intelligence [is] the exciting new effort to make computers think. The fundamental goal of this research is not merely to mimic intelligence or produce some fake. Not at all. "AI" wants only the genuine article: machines with minds, in the full and literal sense. This is not science fiction, but real science, based on a theoretical conception as deep as it is daring: namely, we are, at root, computers ourselves. (p. 2)

Although Haugeland's concept is certainly colorful, it represents only one aspect of the more multifaceted views taken by most scientists regarding the actual role of AI research. Yoshiaki Shirai and Jun-ichi Tsujii (1984) present us with a more traditional perspective:

> Broadly considered, artificial intelligence can be viewed from two standpoints. The first is the scientific standpoint aiming at understanding the mechanisms of human intelligence, the computer being used to provide simulation to verify theories about intelligence. The second standpoint is the engineering one, whose object is to endow a computer with the intellectual capabilities of people. Most researchers adapt the second standpoint, aim-

Figure 3.5 a) A sample tree structure.



diatonic progressions using a diagrammatic representation quite similar—in fact, identical—to a state space. Figure 3.4, drawn from their book *Tonal Harmony,* shows this representation. To use the model, we need simply to start at any given point in the diagram and follow the arrows until reaching a desired state. (Dotted lines represent a motion to any other point in the diagram.)

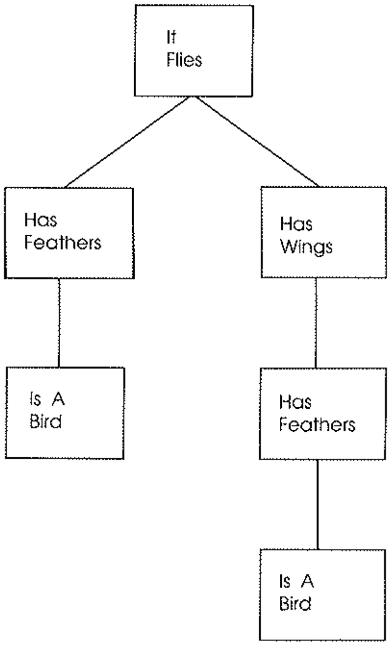
SEARCH STRATEGIES

We can observe from our previous discussion that the process of determining an answer, or deducing the validity of a piece of information, is carried out by searching a path through a state space. For this reason most general knowledge-based heuristics are referred to as *search* strategies. The remainder of this chapter deals with four such heuristic models: *depth-first, breadth-first, iterative deepening,* and *best-first* search strategies.

Depth-First Search Strategies

In dealing with any given state space, we must first seek to organize our node structure in such a way as to clarify the process of seeking

Figure 3.5 b) A tree structure after figure 3.1.



an answer. In a more complex state space, such as one we might model after those in figures 3.1 and 3.2, we can quickly get lost in the process of working through the various combinations of pathways. In other words, the exponential nature of the search quickly proves a significant hurdle for us when working in such an unfocused way. To facilitate our quest, we can redesign our state space into a series of downward-branching *trees,* where we first assign descending entry paths into the top of each relevant node and then assign any number of possible descending branches leading out of that node and into additional lower nodes. Several examples of this can be seen in figures 3.5. The top of the tree (*root*) represents the one possible start state for that domain, and any bottom node represents a possible goal state. Because any given query may have more than one goal, or because the same start state may be used to trace answers to different queries, our tree can have more than one bottom branch. Since these conceptualized trees are structures that enable us to seek answers by employing consistent "downward" paths from top (query) to bottom (goal), we refer to such searches as *depth first,* meaning we find a successful path by simply moving down to the bottom.