Music Perception
Fall 1998, Vol. 16, No. 1, 151–166

Computational Modeling of Music Cognition: Problem or Solution?

PETER DESAIN, HENKJAN HONING, HUUB VANTHIENEN, & LUKE WINDSOR
Nijmegen University

The central purpose of this paper is to elaborate on the methods for computational modeling and to show how, although computational modeling should in principle be instrumental in the understanding of the structure of musical knowledge and the processes involved in music cognition, in practice it too often degenerates into a loose "if you want to understand the theory, here, look in my program" approach. We will show how many issues cannot be decided by inspecting computer programs as they are written nowadays, and we will indicate a possible solution, giving examples from the field of music cognition research.

MUSIC cognition research builds upon various disciplines: musicology, psychology, computer science, and the neurosciences. The field is hence interdisciplinary by its very nature. It not only depends on contributions from these disciplines, but can also link them within single research projects. Each discipline can be regarded as having made some contribution to explaining the phenomena of music cognition, although often in ways that cannot be easily applied or communicated outside that discipline's boundaries. In the margins where these approaches overlap, an explicitly interdisciplinary approach can have the virtue of building theories about mental processes that have proven so far to be too difficult to approach in a monodisciplinary way. Through computational modeling, these theories can be formalized such that they can be implemented as computer programs. As a result of this process, more insight can be gained about the nature of the theory, and most importantly, theoretical predictions are, in principle, much easier to develop and assess. Computational modeling thus has become a well-established method in many fields (e.g., Partridge & Wilks, 1990; Pylyshyn, 1984).

Address correspondence to Peter Desain, "Music, Mind, Machine" Group, NICI, Nijmegen University, P. O. Box 9104, NL-6500 HE Nijmegen, The Netherlands. (e-mail: desain@nici.kun.nl)

Two approaches might be taken to apply such methods to music. The first attempts to formalize musical *knowledge*: here, the theoretical constructs and operations used by music theorists are subjected to such formalization (e.g., Lewin, 1987). The second approach, that of modeling music *cognition*, has the aim of describing or explaining the mental processes that take place when music is perceived or produced (e.g., Jones & Boltz, 1989). Clearly, both of these approaches can be complementary, but cognitive models need not reflect current music-theoretical constructs, nor must models of musical knowledge have cognitive pretensions.

Although music theory, psychology, computer science, and the neurosciences use widely divergent approaches and theoretical constructs, the past decades have shown quite some progress in working toward an interdisciplinary enrichment of our understanding. Hence, there is a tradition on which to build. For example, the work of H. C. Longuet-Higgins (e.g., 1987) reflects a successful attempt to cross the boundaries between two of the disciplines involved, namely, music theory and computer science. The availability of implementations of his theories allows other researchers to experimentally test, refine, and extend the models in ways that are otherwise impossible. Since this pioneering work, many computational models of musical processes have been developed. Similar attempts have been made to form a bridge between psychological and music-theoretical work (e.g., Krumhansl & Kessler, 1982), and some attempts have been made to forge links between psychology and computational modeling (e.g., Povel & Essens, 1985). These three disciplines are combined less often, but with some notable success (e.g., Bharucha, 1987). In addition, strong bridges between the neurosciences and the psychology of music have been developed recently (e.g., Besson & Faïta, 1995; Janata, 1995). Underlying many of these attempts to cross between disciplines is the idea that the processing and mental representation of musical structures, whether in pitch or rhythm, can provide a common ground for research. Such structures can be stated formally or informally within music theory, their processing can be investigated by experimental psychology, both of these aspects can be modeled in computer programs, and both can be given an architectural basis by neuroscience.

The Challenge of Music for Computational Methods

Surprisingly, the apparent success of computational modeling has become one of its main problems: although a huge number of models have been proposed for many musical domains in the form of working computer programs, the psychological validation of these models is often far from satisfactory. The models' implications for theoretical constructs and the relationship between different models is often left obscure. For example, in the area of beat induction (beat tracking), an extremely wide variety of models are proposed (see Desain & Honing, in press). All the models are formalized in different computational paradigms, from rule-based production systems to coupled oscillators, and if the models are validated at all, then this is done on their own test set, making cross-model comparisons quite difficult. These problems obscure the central issue that they are all partial models and address different aspects of the beat induction process. Some models can track a given beat when tempo fluctuates but have to be supplied with an initial beat hypothesis; others cannot handle expressive timing and tempo fluctuations, but deal well with the establishment of the beat percept during the initial stage. Some predict the perceived beat for cyclic rhythms only. Thus, in itself, proposing a new model of beat induction can hardly be seen as a contribution to the field anymore. Recently a methodology has been emerging in which a working computational model is seen much more as the starting point of analysis and research, rather than as an end product (e.g., Desain & Honing, 1992). This approach needs further elaboration, because it promises a way out of the present stagnation. A computational model is thus no longer an aim unto itself, but a means to compare and communicate theories between different research communities.

Music, as language, is an excellent domain for experimenting with this methodology: knowledge about the domain is available at many levels (psychoacoustic, psychological, neurobiological, music-theoretical, historical), and explicit formalization is not as alien to musical practice as commonly thought: consider, for example, the extensive symbolic music notation systems that have evolved. In human culture, music is as widespread a phenomenon as language. Moreover, although the variety of different musical practices within even a single culture, and across different cultures and periods, appears at first to refute the possibility of any global formalization of music cognition, there are reasons to believe that basic mechanisms underlie human perception and performance that facilitate and constrain the genesis of a music, much as they may do in language (Pinker, 1995). On the other hand, music is not an easy domain for developing computational models. One problem is the difficulty of dealing with mutually incompatible and interacting structural descriptions, for example, grouping and meter (see Lerdahl & Jackendoff, 1983). Another is the combination of discrete and continuous types of information such as that of discrete pitch classes and continuous frequency modulations. Moreover, as has been shown for language (e.g., Neely, 1991), the ongoing nature of music perception or performance often relies on a combination of both bottom-up and top-down mechanisms: one cannot easily pretend that our musical knowledge is static, that it does not influence what we hear and how we play, or that it cannot be revised by our continuing exposure to musical stimuli. These problems, though, make music a fruitful domain of research, because they may require complex solutions, the nature of which may be beneficial to the parent disciplines with which it is studied.

However, to analyze the limitations of current attempts to use computational modeling techniques in music cognition research, and to propose a methodology to go beyond these limitations, we first need to clarify and expand upon the different existing relationships between computational models and music cognition research in a more general fashion.

Computational Modeling in Music Cognition Research

As in cognitive psychology in general, the use of computational models has become prevalent in music cognition research (e.g., Bharucha, 1991; Gjerdingen, 1994). This work arose out of a relatively long tradition of research in experimental music psychology (Deutsch, 1982; Dowling & Harwood, 1986; Fraisse, 1982; Krumhansl, 1990; Sloboda, 1985) in which computational models were less of an issue. From a psychological perspective it is important that the method of computational modeling functions well in this context of empirical validation. But unfortunately, the gap between the cognitive work, which brings forward these models, and the experimental approach, which should validate them, is huge. Besides finding better ways to validate new computational models empirically, the need to compare existing models is obvious. Progress in the understanding of psychological processes is, at the moment, we would argue, hampered by the multitude of models proposed and their contrasting formalisms. The large variety of incompatible models within music cognition, such as those that attempt to model beat and/or meter induction (Desain & Honing, 1994), and the unclear link between them are manifestations of a real problem in cognitive science itself. In the end, cognitive science needs to make and justify claims about human cognitive processes, not only to develop programs implemented in one or the other computational paradigm. These problems are found in many other domains of cognitive science, for example, visual word recognition (Jacobs & Grainger, 1994). Current proposals for models based on continuous (subsymbolic/numerical) and discrete (symbolic/structural) processing and knowledge representations are almost always studied in isolation, in their respective research communities. This factionalism is exemplified by the separation between beat induction research that uses complex nonlinear dynamics (Large & Kolen, 1994) and that which uses rule-based and other symbolic methods (Longuet-Higgins, 1987). In such a situation, it will remain unclear what a particular computational paradigm of formalism contributes to the ease of expressing the model, whether it is essential to the model's description, or whether it can be considered merely an implementation choice.

However, gradual progress is being made toward the use of descriptions of behavior that exhibit more details than pure input/output specification, which we believe will help to solve the problems just mentioned. We have found that the development of an abstract formal representation of the behavior of systems that can be applied across paradigms yields direct insight into the differences (often smaller than expected) and similarities between the systems (Desain, 1993). Expectancy is one such notion that, apart from its more or less traditional interpretation as a psychological factor, can function as a theoretical construct that allows the study of computational models across different formalisms and paradigms. It defines the internal state of a system (arising from processing previous input) in terms of its effect on processing new material. As such, this notion can be applied to systems with widely different state representations (registers, activation levels, oscillator phases, etc.) but will still yield expectancy constructs that can in principle be compared. The link from expectancy to selective attention, to sensitivity changes over time, as developed by Large and Jones (in press), is an approach that is abstract enough to be applied to the validation of beat induction models in different computational paradigms. In our view, such attempts are worthwhile because they may help prevent fragmentation and needless duplication within the study of cognition.

Sometimes computational models are based on constructs outside the realm of cognitive processes, like physical masses and forces (e.g., Todd, 1993). Although these models form a practical alternative to a wholly mentalistic approach, and they may provide reasonable predictions on an input-output level, there are currently no good ways of evaluating whether such physical variables are causal in a psychological sense, whether they are causal in a more physical sense, or whether they are just convenient post-hoc descriptions of some other process that underlies a particular behavior.

An example of this is the final ritard, the slowing down of tempo at the end of a piece, observed in musical performances (see Desain & Honing, 1996). It is difficult to judge whether such a ritard merely happens to mimic the motion of physical bodies under deceleration or whether there is some more direct causal relationship between musical motion and real movement. Hence, there is a need for methods of model evaluation that can distinguish not only between the success or failure of a model's ability to represent a particular type of behavior, but also to determine whether such a representation can be interpreted as more than an abstract computing device and used to make claims about the particular physiological mechanisms involved.

Testing the Validity of Computational Models

Before describing our proposals for elaboration of the methodology itself, we will sketch the conceptual framework of computational modeling and the constructs involved. Imagine that we are interested in the behavior of a human subject (e.g., a listener, performer, or composer) involved in some task. The subject can be seen to exhibit some kind of behavior in response to some stimulus material (see Figure 1a, top). This behavior can be studied (as in experimental psychology) to test the relationships between the factors involved. Although this approach tells us which factors in the stimulus are relevant for the response, and it can test in which way they influence behavior, it cannot provide a description of the mechanism of the cognitive process.

In the computational modeling approach, an algorithm is constructed that exhibits similar behavior in order to find out more about the cognitive process that achieves the task. This turns out to be a difficult task that often consumes most of the research effort. But, in principle, after the construction of a computational model, its behavior for given stimuli can be tested against that of the human subject (see Figure 1a), at least after an interpretation problem has been solved: the input and output representations of the program have to be related to the stimulus material and behavior of the human subject, and reductions and assumptions have to be made because the behavior of the subject is not formalized in itself (Ringle, 1983). For example, in investigating beat induction, an abstract notion of a repeating time interval brought forward by the program as the induced beat has to be related to a beat percept in the listener as measured by an overt response, like foot tapping. Moreover, even assuming that such a reasonable mapping exists between input and output of the model and output of the human subject, and assuming that the model builder has succeeded in creating an algorithm that exhibits behavior that agrees with the empirical data, we still cannot say much about the architecture of human cognition. The fact that the behavior of the two systems agrees does not, in itself, validate the algorithm as a model of the mental processes responsible for the behavior of the human subject. For example, the model builder might have implemented a large lookup table and listed all the stimulus patterns and the appropriate responses. This null-architecture does not explain anything, but it produces the desired behavior.

Therefore, we have to "open up" the program to check what is inside, that is, to look for the congruence between the model and the human subject at a finer scale—a comparison based on global input/output behavior (i.e., functional equivalence) is too coarse to make substantial claims about the psychological validity of the model. If we want to make statements about the architecture of human cognition, we must somehow relate the

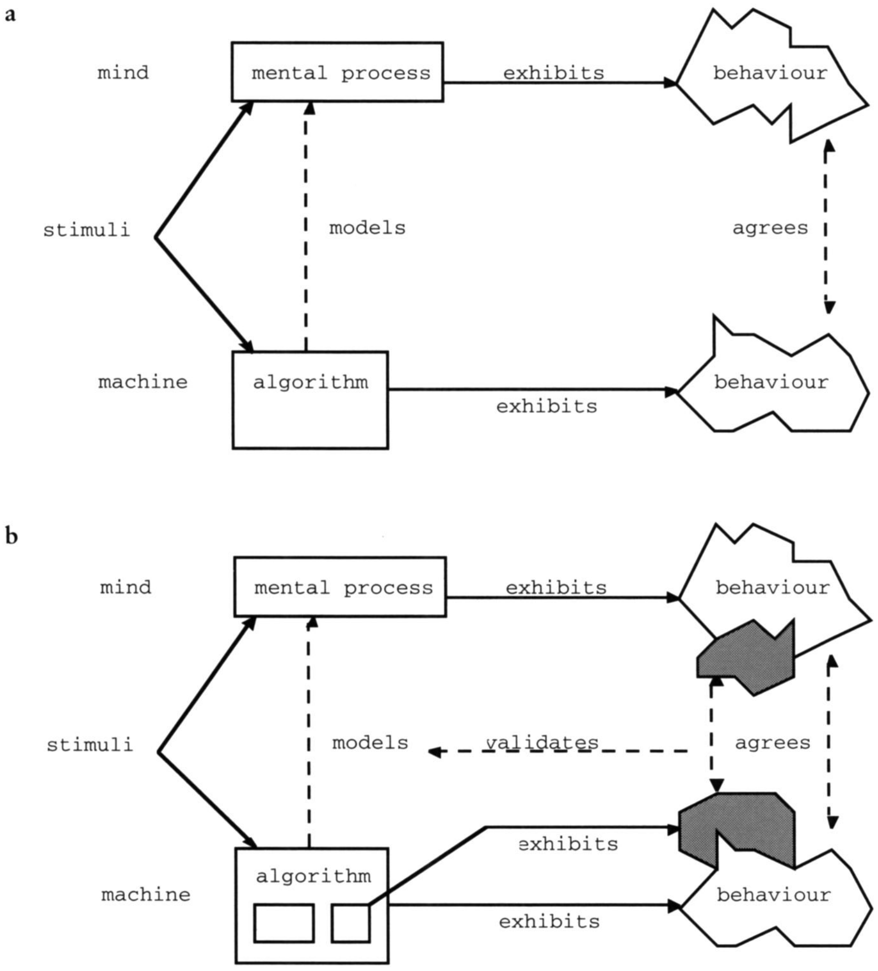

Fig. 1. Psychological validation of a computational model.

architecture of the program to that of the human subject. Take for example an algorithm that decomposes a task in two subprocesses (see Figure 1b)—say, initial establishment of a beat and subsequent tracking of it during tempo changes. If the program is claimed to be a cognitive model, this architecture is taken to reflect a similar subdivision in the mental processes of the subject. Since it was shown, in principle, that such an architecture is a viable one (because the program performs well), we now have to show that it is psychologically valid. In most cases, we cannot assess this directly (apart from case studies of brain-injured subjects and from brain-scan techniques), nor should we aim to prove that the neurobiological processes have a similar modularity. To show this strong equivalence (Pylyshyn, 1984) is, in most cases, not a realistic objective. Because the descriptions are at such a high abstraction level, we can claim only that our model is a psychologically plausible decomposition of the cognitive processes to some degree, and we have to show this in an indirect way.

To show such weaker abstract architectural equivalence, a number of additional steps must be taken. First, we must extend the definition of behavior for the model (see Figure 1b, bottom gray area) beyond mere input-output specification. For example, we could define the range of timing disturbances that do not throw our beat induction model off track and specify how this changes when the second processing stage takes over. Then, we would try to define a similar extension of the behavior of the subject (see Figure 1b, top gray area), measuring differential sensitivity to the same timing disturbances during the establishment of a beat percept. Defining such extensions to definitions of behavior, such as the amount of input required before an output can be given, the errors that are most likely to occur, the points in time until different subprocesses can be disturbed, and so on, is an art in itself, which is still much more developed in psycholinguistics than it is in music cognition research. Nonetheless, once these extensions were made, it would be possible to check the agreement between these new behavioral factors at an appropriate level of analysis. Clearly, it would not be appropriate to base one's interpretations of the congruence between model and human behavior on precisely the same metric. For example, we would not necessarily seek to show that a human subject's sensitivity to timing perturbations in the stimulus was exactly the same as that of the model, but that it varied in the same manner. We might even look for correlations between different measures, which imply similar underlying causality: greater computational load for an algorithm might correlate with human reaction times, giving some indication that similar changes in the stimulus required greater "cognitive" resources for both subject and computational model. If the model predicts this extended behavior well, this can be taken as evidence of the adequateness of the model as a description of mental processes. This is especially so if the predicted behavior is critically related to the proposed decomposition, an issue that can be better understood when more models with a conflicting modularity are available for study. In this indirect way, evidence for the psychological validity of the design of a computational model at a certain level of modularity can be collected.

When the modularity of an algorithm is shown to be consistent with the extended behavior, we can proceed to break down the next level of architecture and see if there are ways to verify those levels empirically. However, a computer program can be broken down into extremely fine building blocks, levels of descriptions of behavior that were not intended by the modeler to reflect anything in human cognition. We propose an elaboration of the computational modeling methodology by defining extensions of programming languages in which these concerns can be expressed explicitly by the model builder.

From Programming Language to Cognitive Modeling Language

ARTIFICIAL INTELLIGENCE AND EXPERIMENTAL PSYCHOLOGY

Ideally, a computational model should form a bridge between research in artificial intelligence (AI) and experimental psychology. On the AI side, the program would be developed to perform a certain task, and within the field of psychology, the same program would be interpreted as a model of human cognition and its mechanisms would be tested for psychological validity. Both sides have different aims. For technical AI, a computer program should perform well, and even the use of an unintelligent tabular or exhaustive search method is not a priori excluded. For psychology, other objectives exist. It is desirable that the program have characteristics (other than input-output relations) that reflect the properties of human cognition as well. Because a formal model exists as a means of communication between the disciplines, and objective formal assertions can be made, one would think that the issues can be easily determined. However, this is not the case. The gap between AI and experimental psychology is huge, although some progress has been made in bridging it (Jacobs & Grainger, 1994; Kosslyn, 1994; Newell, 1990; but see also Brachman & Smith, 1980; Wilks, 1990). We think the main reason for this gap is the fact that computational models tend to be expressed as a program in a general programming language, and no further formal description of its state as a model is available. A programming language is not a modeling language, and the availability of an implementation of a theory does not necessarily make it clear what predictions about human performance follow.

Any program is in some sense structured, that is, it consists of a number of components and some way to connect them. An example of such a structure is the hierarchical top-down decomposition of a program in procedures or routines that in some way call each other. Although the higher levels of such a hierarchical structure might reflect a corresponding structure in the architecture of cognitive processing, the lower levels may have been designed to optimize the performance of the program on a particular computer and hence describe aspects of this machine rather than of the cognitive process the program models. It is generally not obvious which aspects of modularity were and which were not intended to reflect this architecture, and which aspects deal with a particular implementation only (the grain-problem; see Pylyshyn, 1979). So, a program alone does not specify which parts of its components are psychologically relevant and what that relevance may be. A component of a program may thus have several roles from the psychological viewpoint. For instance, it may indicate a functional module in the description of how the mind works, but it may also be interpreted as a module in the architecture of the brain itself. On the other hand, the component may be present only for reasons of implementation.

In this sense, a computer program is too *imprecise* as a model, because it leaves essential information implicit, namely, in what sense the components are psychologically relevant. On the other hand, a computer program can be considered too *precise*, because it provides detailed algorithmic information even where the theory does not want to claim that equivalent processes are somehow operative in the human brain.

A PROPOSAL FOR A COGNITIVE MODELING LANGUAGE

A solution to these problems might be found in specially designed languages. For example, in SOAR (Laird, Newell, & Rosenbloom, 1987; Newell, 1990), a cognitive modeling language that formalizes cognition as goal-directed problem solving, one solution to such problems is attempted. The design of SOAR supposedly reflects many characteristics of human cognition. ACT* (Adaptive Control of Thought), a production system based on the assumption that knowledge is initially propositional and procedures can be derived from these propositions, has a similar goal (Anderson, 1983). When compared with SOAR and ACT*, our proposal is more open. Although the aims are similar, we do not go so far as to postulate a fixed cognitive architecture but rather aim to develop formal constructs that can be added to existing programs to formalize their interpretation as a cognitive model. We thus propose to develop constructs that permit an algorithm to be described formally such that it is not solely a description of a computation, but also entails a specification of its status as a partial model of another (mental) process implementing the same computation. For example, such constructs would allow a formal expression of the fact that modeling per se often stops below a certain level in a program and that everything below that level is considered implementation details.

Sometimes the complete process can be described only by means of a constraint on the input-output behavior, and no detailed processing is brought forward. This is, for example, the case in the model of beat finding by Povel and Essens (1985), which predicts the metric mental framework that will be induced by a cyclic rhythmical pattern but is not intended to model the process of how that percept develops. In terms of formal specification languages (Partsch, 1990), only the specification of the program (in terms of input-output constraints) is brought forward as a model, not the implementation. Presently, no formal expression of this situation can be stated in our programs, yet they are critical to their interpretation as cognitive models.

When we cannot distinguish between cases in which specification or implementation is assigned a model status, measurements of, and reasoning about, such variables as computational complexity and memory load as correlates of human behavior become useless. The elegant abstraction mechanisms developed in computing science (Abelson & Sussman, 1985) form the beginning of a solution, because they may form the places where this status is expressed. Data abstraction constructs (e.g., supporting definitions of the structure of a frame, record, or object) allow one to specify the way in which data may be accessed and updated, and which parts of the program may do so. Procedural abstractions (e.g., allowing one to define a function or procedure) group descriptions of change into units of action and specify how the flow of information (e.g., call by value) between those units of action is controlled. Control abstractions (e.g., supporting recursion, iteration, parallelism) support more or less abstract ways in which actions can be chained together.

These abstraction constructs can be annotated with assertions about their role in the interpretation of the program as a model. Consider, for instance, the naive ease with which statements in a program are ordered sequentially by the programmer, simply because the programming language requires the statements to be ordered. This means that information is lost about the nature and dependencies of stages in the processing. Statements may indeed critically rely on computations done in order, and a sequential ordering is explicitly meant. However, others do not need information computed by predecessors, being logically independent, and could be assumed to happen in parallel when interpreted as a cognitive model. A simple annotation, allowing the author to express these differences, may suffice here.

A language in which annotations can be given could be designed in two ways. In a first approach, an inventory of the needed constructs would be made and a semiformal language of annotations would be designed. Expressions in this language could then be added as comments to existing programs. In a second approach, the constructs would need to be integrated into the language itself (such that they can be checked for consistency and handled automatically). The Metaobject Protocol of the Common LISP Object System (Kiczales, des Rivières, & Bobrow, 1991) provides the means to do this easily. It would provide psychologists with an extended programming language in which modeling aspects can be expressed formally and added to the algorithm. In this way, it becomes a bit easier to reason about (as well as derive predictions of) task complexity, reaction times, memory load, error rates, and similar measures.

However, the programs presently proposed as computational models are often highly unstructured and lack the use of abstraction mechanisms. Hence, they are placed out of reach of the formal methods available for reasoning about algorithms such as equivalence proofs and formal semantics (see, e.g., Stoy, 1977). Automated reasoning about programs can provide help in proofs of their equivalence. Semantic equivalence is of course not only a practical issue: often the question of how different computational models really are can, in the end, be answered only by formal reasoning. Automated reasoning can also help with specification checks, which check whether the "what" (specification) conforms with the "how" (implementation), and semantically invariant transformations, which alter the implementation but are guaranteed to maintain the specification. Automated reasoning can also help in optimization, which alters the program architecture to achieve better computation and/or memory load, or even lead to automatic program synthesis (Smith, 1992), which derives an implementation directly from a specification (Partsch, 1990; Rich & Waters, 1986). After a reformalization of program code using meaning-preserving program transformations (Friedman, Wand, & Haynes, 1992), many important programs (or models) can be brought within reach of these methods of analysis and can profit from the advantages of computational modeling.

The form in which these tools from theoretical computer science (like program transformations) can be made available for psychologists remains to be determined. But when these tools are available in a programming environment, they can also help in extracting the essence of large published programs into a micro version, a technique we often use for understanding and comparing existing computational models and systems (e.g., Desain, 1991; Desain & Honing, in press; Honing, 1995). These tools are especially useful when code has to be lifted from an obsolete language or implemented in a new one, as was the case in the study of a complex program brought forward by Longuet-Higgins as a parser for musical rhythms (Desain, 1991). The formal tools help in keeping semantics (the input-output relation) invariant while disentangling control structure, making binding of variables to values explicit and defining their scoping regimes, clarifying in which part of the program a variable may be accessed. The aim in this work is to simplify the program and strip the code to its essential core, separating implementation aspects from central representational notions, and applying different types of abstraction that increase the modularity, transparency and parsimony. Of course, semantic-invariant program transformations are no cure for all programs, and the application of this formal approach is still difficult for large and unstructured programs. However, a formal method that cannot be applied in general to all programs in a certain programming language is often still applicable in the specific case of one program or part of a program that does not use the full power of the language.

But even when a neat formal specification of a model can be given, the problem may remain that the modularity that enables a neatly factored and modular description of complex behavior may not be a realistic representation of mental processing, nor of an optimal algorithm. One needs to be able to annotate parts of programs to state that the modularity at that level of design is not to be taken seriously in predictions about, for example, computation load, and that the intention is to freely allow any transformation (e.g., to merge computations from different modules) that leaves the program semantically invariant, that is, implements the same input-output relation.

As an example, consider the design of a program that checks whether the surface, the notes of two different music-analytic trees, or the words in two syntactic structures are the same, regardless of the structure of the tree. The most simple and elegant program would modularize the task into a "flattening" of the trees and then compare the resulting lists. But this program is essentially inefficient: both trees are flattened completely before a difference in a first leaf may be detected. The program that avoids this inefficiency walks through both trees at the same time, maintaining an administration for both, and effectively merges the flattening and equality test programs. This program would predict processing times that are much more closely related to the essential complexity of the problem at hand and optimality can be proved. However, merged, nonmodular programs are notoriously difficult to write and maintain. A third solution requires more knowledge of parallel programming constructs. Instead of subjecting both trees to a flattening operation, a process is created for each tree that can deliver the next leaf when asked to do so. These processes are asked to produce a leaf each, and upon successful comparison, the process is iterated. All three implementations are semantically equivalent, that is, they will yield the same answer, but predictions about time and space complexity will differ. Moreover, the more complex algorithms can be derived almost automatically from the simplest one when a proper programming style is used. So, by programming in this style, the model can, on the one hand, be a neatly factored and modular description of complex behavior and, on the other hand, be subjected to automatic reasoning that extracts essential characteristics of this complexity in the form of a nonmodular, optimal algorithm.

Conclusions

By adopting some of the more advanced methods of computer science, the use of computational models in music cognition, and indeed, cognitive science in general, can be made much more fruitful. We would argue that by adopting such methods, comparisons between competing models, and between models and human behavior, become much more meaningful. Hence, such methods are more than just elegant programming concepts and have a fundamental role to play in progressing toward a more detailed and formal understanding of cognition. They are not, however, to be seen in isolation from the methods of experimental psychology or cognitive neurosciences. Indeed, without reliable data about human behavior and brain functions against which to compare a model's behavior, the psychological value of computational models, however well constructed, will always be limited. Moreover, neither should these models be seen as divorced from existing theoretical knowledge about music: musicological knowledge has an important role to play in providing semiformalized constructs that may be both modeled and tested to investigate their cognitive veracity. At present, however, such comparisons are extremely difficult to make, and we see the need for detailed attention to the way in which computational models are implemented and expressed. It is no longer enough to point to the program code and say, "look, here it is."[1]

References

Abelson, H., & Sussman, G. J. (1985). *Structure and interpretation of computer programs.* Cambridge, MA: MIT Press.

Anderson, J. R. (1983). *The architecture of cognition.* Cambridge, MA: Harvard University Press.

Besson, M., & Faïta, F. (1995). An event-related potential (ERP) study of musical expectancy: Comparisons of musicians with nonmusicians. *Journal of Experimental Psychology: Human Perception and Performance, 21*(6), 1278–1296.

Bharucha, J. J. (1987). Music cognition and perceptual facilitation: A connectionist framework. *Music Perception, 5*(1), 1–30.

Bharucha, J. J. (1991). Pitch, harmony, and neural nets: A psychological perspective. In P. M. Todd & D. G. Loy (Eds.), *Music and connectionism.* Cambridge, MA: MIT Press.

Brachman, R. J., & Smith, B. C. (1980). Special issues on knowledge representation. *SIGART Newsletter, 70,* 103–104.

Desain, P. (1991). Parsing the parser: A case study in programming style. *Computers in Music Research, 1*(2), 39–90.

Desain, P. (1993). A connectionist and a traditional AI quantizer, symbolic versus subsymbolic models of rhythm perception. *Contemporary Music Review, 9,* 239–254.

Desain, P., & Honing, H. (1992). *Music, mind and machine: Studies in computer music, music cognition and artificial intelligence.* Amsterdam: Thesis Publishers.

Desain, P., & Honing, H. (1994). Foot-tapping: A brief introduction to beat induction. In *Proceedings of the 1994 International Computer Music Conference* (pp. 78–79). San Francisco: International Computer Music Association.

Desain, P., & Honing, H. (1996). Physical motion as a metaphor for timing in music: The final ritard. In *Proceedings of the 1996 International Computer Music Conference* (pp. 458–460). San Francisco: International Computer Music Association.

Desain, P., & Honing, H. (in press). Computational models of beat induction: The rule-based approach. *Journal of New Music Research.*

Deutsch, D. (Ed.) (1982). *The psychology of music.* New York: Academic Press.

Dowling, W. J., & Harwood, D. (1986). *Music cognition.* New York: Academic Press.

1. This research has been made possible by the Netherlands Organization for Scientific Research (NWO).

Fraisse, P. (1982). Rhythm and tempo. In D. Deutsch (Ed.), *The psychology of music* (pp. 149–180). New York: Academic Press.

Friedman, D. P., Wand, M., & Haynes, C. T. (1992). *Essentials of programming languages.* Cambridge, MA: MIT Press.

Gjerdingen, R. O. (1994). Apparent motion in music? *Music Perception, 11*(4), 225–370.

Honing, H. (1995). The vibrato problem: Comparing two solutions. *Computer Music Journal, 19*(3), 32–49.

Jacobs, A. M., & Grainger, J. (1994). Models of visual word recognition: Sampling the state of the art. *Journal of Experimental Psychology: Human Perception and Performance, 20*(6), 1311–1334.

Janata, P. (1995). ERP measures assay the degree of expectancy violation of harmonic contexts in music. *Journal of Cognitive Neuroscience, 7*(2), 153–167.

Jones, M. R., & Boltz, M. (1989). Dynamic attending and responses to time. *Psychological Review, 96,* 459–491.

Kiczales, G., des Rivières, J., & Bobrow, D. G. (1991). *The art of the Metaobject Protocol.* Cambridge, MA: MIT Press.

Kosslyn, S. M. (1994). *Image and brain: The resolution of the imagery debate.* Cambridge, MA: MIT Press.

Krumhansl, C. L. (1990). *Cognitive foundations of musical pitch.* New York: Oxford University Press.

Krumhansl, C. L., & Kessler, E. J. (1982). Tracing the dynamic changes in perceived tonal organisation in a spatial representation of musical keys. *Psychological Review, 89*(4), 334–368.

Large, E. W., & Jones, M. R. (in press). The dynamics of attending: How we track time varying events. *Psychological Review.*

Large, E. W., & Kolen, J. F. (1994). Resonance and the perception of musical meter. *Connection Science, 6*(1), 177–208.

Laird, J., Newell, A., & Rosenbloom, P. (1987). SOAR: An architecture for general intelligence. *Artificial Intelligence, 33*,1–64.

Lerdahl, F., & Jackendoff, R. (1983). *A generative theory of tonal music.* Cambridge, MA: MIT Press.

Longuet-Higgins, H. C. (1987). *Mental processes.* Cambridge, MA: MIT Press.

Lewin, D. (1987). *Generalized musical intervals and transformations.* London: Yale University Press.

Neely, J. H. (1991). Semantic priming effects in visual word recognition: A selective review of current findings and theories. In D. Besner & G. W. Humphreys (Eds.), *Basic processes in reading: Visual word recognition* (pp. 264–336), Hillsdale, NJ: Erlbaum.

Newell, A. (1990). *Unified theories of cognition.* Cambridge, MA: Harvard University Press.

Partridge, T., & Wilks, Y. (Eds.) (1990). *The foundations of artificial intelligence, a sourcebook.* Cambridge, England: Cambridge University Press.

Partsch, H. A. (1990). *Specification and transformation of programs: A formal approach to software development.* Berlin: Springer.

Povel, D. J., & Essens, P. (1985). Perception of temporal patterns. *Music Perception, 2*(4), 411–440.

Pinker, S. (1995). *The language instinct.* London: Penguin Books.

Pylyshyn, Z. (1979). Complexity and the study of human artificial intelligence. In M. Ringle (Ed.), *Philosophical perspectives in artificial intelligence* (pp. 23–56). Atlantic Highlands, NJ: Humanities Press.

Pylyshyn, Z. W. (1984). *Computation and cognition: Toward a foundation for cognitive science.* Cambridge, MA: MIT Press.

Rich, C., & Waters, R. (1986). *Readings in artificial intelligence and software engineering.* Los Altos, CA: Morgan Kaufman.

Ringle, M. (1983). Psychological studies and artificial intelligence. *The AI Magazine, Winter/Spring,* 37–43.

Sloboda, J. (1985). *The musical mind: The cognitive psychology of music.* Oxford: Clarendon Press.

Smith, J. O. (1992). Physical modeling using digital waveguides. *Computer Music Journal, 16*(4), 74–91.

Stoy, J. E. (1977). *Denotational semantics: The Scott-Strachey approach to programming language theory.* Cambridge, MA: MIT Press.

Todd, N. P. M. (1993). Vestibular feedback in musical performance: Response to *Somatosensory Feedback in Musical Performance* (edited by Sundberg and Verrillo). *Music Perception, 10*(3), 379–382.

Wilks, Y. A. (1990). One small head: Models and theories. In D. Partridge & Y. A. Wilks (Eds.). *The foundations of artificial intelligence* (pp. 121–134). Cambridge, England: Cambridge University Press.