Athanasius Kircher's Arca musarithmica (1650) as a Computational System

Andrew A. Cashner *

March 28, 2024

Abstract

In Book Eight of Athanasius Kircher's *Musurgia universalis* (Rome, 1650), this Jesuit polymath describes a computing device for automated music composition, called the Arca musarithmica. A new software implementation of Kircher's device in Haskell, a pure-functional programming language, demonstrates that the Arca can be made into a completely automatic computational system. Moreover, the project also demonstrates that the Arca in its original form already constituted a computational system that almost completely automatic though designed to be operated by a human user: as Kircher advertised, a completely "amusical" user could generate music simply by using the device according to his rules. The device itself served as a microcosm of Kircher's goal in the *Musurgia* to encapsulate and codify all musical knowledge, and demonstrate that music manifested the underlying mathematical order of the Creation and its Creator. This article analyses the concepts and methods of computation in Kircher's original system, in dialogue with the interpretation of his system in software. The Arca musarithmica, now available on the web, makes it possible actually to hear how well Kircher was able to reduce seventeenth-century music to algorithmic rules. The successes of the system are inseparable from many paradoxical elements that raises broader questions about how Kircher and his contemporaries understood the links between composition and computation, mathematics and rhetoric, traditional harmonic theory and emerging tonal practice, and concepts of "invention" and authorship.

Keywords: algorithmic composition, history of music theory, history of science, Athanasius Kircher, music encoding

1 An "Ark" of Musical Invention in Infinite Permutations

In 1650 Athanasius Kircher published a detailed specification for a device called the "Arca musarithmica", which appears to be an analogue computing device for automatic music composition (Kircher, 1650, II: 1–199). Within the ten books of the *Musurgia universalis*, this German-born Jesuit scholar of ancient languages and mathematics (1602–1680) attempted to compile everything that was known about music in a single Latin treatise (Findlen, 2004a; Fletcher, 2011; Godwin, 2009). The device he describes in Book Eight is more than a demonstration of musical combinatorics: Kircher epitomises his whole concept of music in one small box (Bohnert, 2010; Chierotti, 1994; Chierotti, 1992; Klotz, 1999; Murata, 1999; Pangrazi, 2009, 157–167; Wald, 2006, 134–143).

In the eighth book Kircher announces an "ars nova musarithmica recenter inventa", "a new musical-numerical method, recently discovered, by means of which even an untrained musician can achieve perfect composition in a short time" (Kircher, 1650, II: 1). Kircher's "new method" is a system for

*Assistant professor of music, University of Rochester, andrew.cashner@rochester.edu. This work was completed on the ancestral land of the Onöndowa'ga:' (Seneca) Nation, one of the Six Nations of the Haudenosaunee (Iroquois) Confederacy. Thanks are due to Christopher Brown, Devin Burke, Paulo Cereda, Margaret Murata, David Temperley, and the anonymous reviewers of this journal for reading drafts and providing helpful feedback. All translations are my own. This article is dedicated to Dr. Donald Knuth, from an admirer and fellow organist, in gratitude for the way his work inspired me to learn computer programming and seek connections between computer science and music.

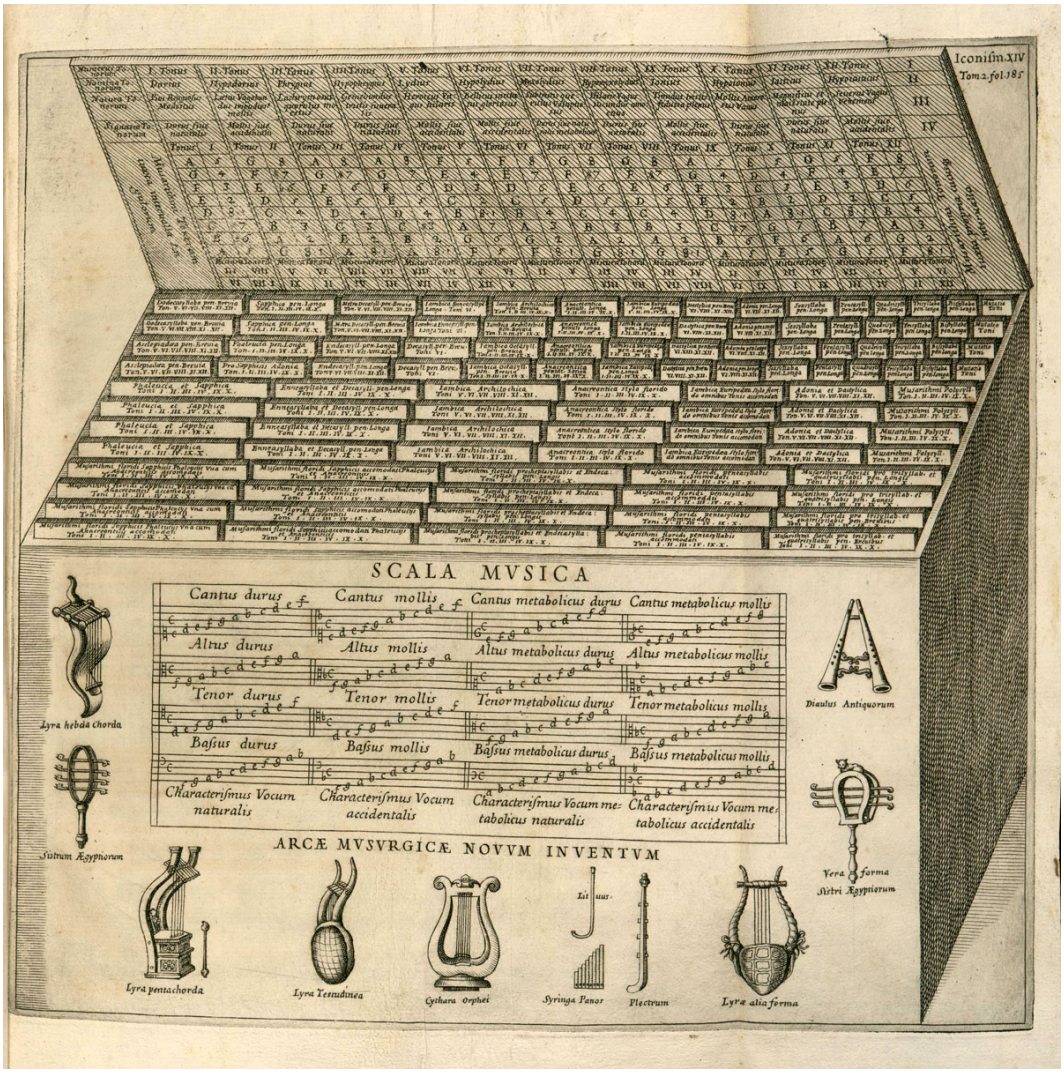


Figure 1: Kircher's Arca musarithmica, *Musurgia universalis* (1650) II, facing p. 185.

automatic music composition, embodied in the physical device whose name means something like "box for musical calculation" (Fig. 1). By means of this device, he says, a "Tyro" (novice) or even an "amusical person" can compose nearly every type of modern music from simple four-part homophony to florid, imitative counterpoint, and even polychoral and recitative styles. Moreover, an *amusicus* (someone without musical knowledge) can compose music expressive of any mood, and can even set words to music from any world language – all simply by operating the device according to Kircher's detailed rules. Kircher advertises that, if used properly, the ark can produce music that a professional musician would recognise as correct, appropriate to the text, and rich in variety. He includes a full sample composition which he says was generated by an untrained musician using the ark and heard with acclaim in elite Roman circles (Kircher, 1650, II: 166).

While skepticism might be warranted for some of Kircher's claims, there is evidence that people around the globe did build and operate the Arca musarithmica. Kircher's Jesuit brothers circulated copies of the *Musurgia* through their global networks as far as Manila and Mexico City, and evidence suggests a version of the Arca was brought into the German lands (Findlen, 2004b; Irving, 2010, 48–50; Annibaldi, 1995). Three surviving physical implementations of the Arca musarithmica were known until I discovered a fourth in Puebla, Mexico (Bohnert, 2010, 127–156; Boni, 2020). The version in New Spain was probably made around 1690 by a university mathematics professor who was part of a circle of Kircher aficionados in New Spain that included the poet Sor Juana Inés de la

Cruz (Cashner, 2022b; Osorio Romero, 1993; Trabulse, 1984; Beuchot Puente, 1995). I also found a pamphlet describing the Arca from late eighteenth-century Madrid. While a physical implementation of the Arca might have value purely as a curiosity, it is hard to believe that no one ever tried to use them as intended. In fact, the version in Puebla includes only a partial selection of Kircher's data tables, which seem to have been selected for their practical value.

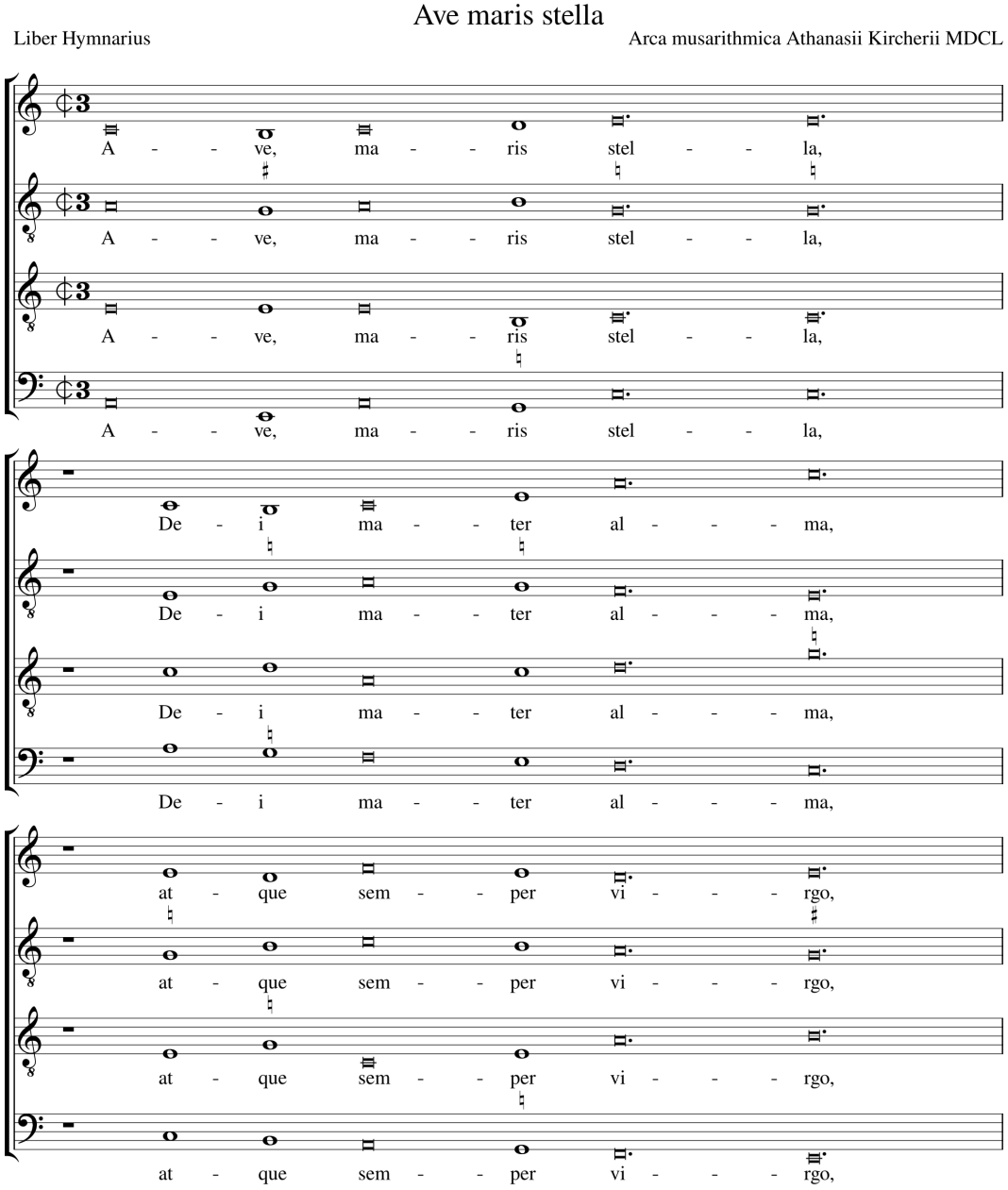
None of the surviving implementations are automatic machines, though: the Wolfenbüttel and Cambridge versions are wooden boxes with paper slats inside holding Kircher's tables of numbers and notes, and the Puebla version consists of paper copies of the tables, originally looseleaf. Carlo Mario Chierotti argues that despite Kircher's use of terms like *mechanicus* and *artificiosus* to describe the Arca, his invention is neither mechanical nor automatic; and in fact requires significant discretion from a human operator to produce workable results (Chierotti, 1994, p. 390). Jim Bumgardner calls a related device by Kircher "nothing more than a glorified recipe box" (Bumgardner, 2009, p. 3).

Did the Arca musarithmica really work as advertised? And should we consider it to be a form of computer? What kind of computation did Kircher think was necessary to compose music, and how did he connect concepts of calculation and composition? How did Kircher encode musical knowledge in the Arca, and did his invention actually output music that would be judged coherent and intelligible?

To answer these questions, I created a working, completely automatic, software implementation of Kircher's Arca musarithmica (Cashner, 2021) in the pure-functional programming language Haskell (Haskell.org Committee, 2024; Wikibooks contributors, 2018). Anyone may generate music using the web application version (Cashner, 2023), which also includes the complete source code and documentation written in a literate programming style (Knuth, 1992). Users can choose among prepared input texts and select the style, metre and mood of the music; or they can even type in their own text. Building the ark in software is more than just a programming project; it is an act of interpretation balancing textual analysis with creative synthesis. The implementation serves two main purposes: first, it tests the hypothesis that Kircher actually did describe an effective system for musical composition in the form of an algorithm that only lacked the technological means to make it fully automated. Second, it makes it possible to test the output of the ark to a degree Kircher could not have imagined, and thereby to evaluate the kind of musical knowledge encoded in the ark and how much musical knowledge is actually required to generate acceptable music with the system.

I argue that Kircher did establish a computational system for composition, including data structures and algorithms, in which he formulated specifically musical conceptions of computation. The program demonstrates that it is possible to conceive of the entire ark as a single mathematical function: it takes one series of symbols, representing an input text and the choice of musical parameters, and transforms those into another set of symbols that represents musical notation. Kircher explicitly aims to provide a mathematically sound method of composition, and though his specifications fall short of the standards of modern computer science, his algorithm works. The musical results generated by this implementation do sound like mid-seventeenth-century Italianate music in a conservative, church-oriented style. There are some mistakes in the data, and portions of the instructions are underspecified or omitted, but in the end only a few parts of the system actually required the addition of missing data or processes. In most of those cases Kircher was probably right to assume that a human user would supply these gaps intuitively. These places where Kircher skips a step or assumes knowledge that is not encoded in the ark actually reveal important aspects of his understanding of music, because the whole device should be understood as an attempt to systematise a compositional theory of music.

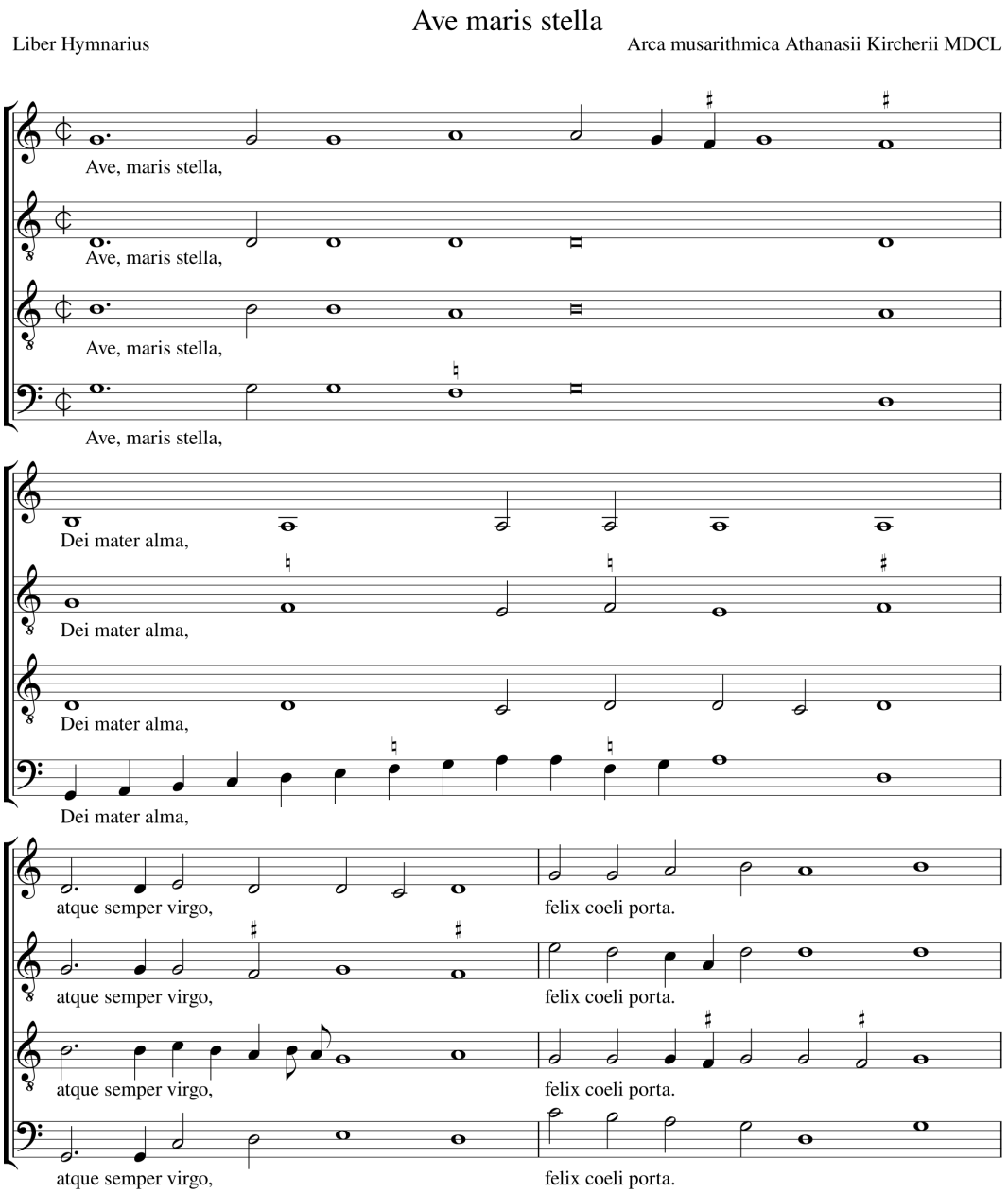
This is not the first attempt to implement the ark: in addition to the physical implementations already mentioned, Agnes Cäcilie Bohnert modeled the ark in a Java program, and Bumgardner built a related system in Perl. Bumgardner's program implemented the musical portion of the *Organum mathematicum* described by Kircher's student Gaspar Schott, which was much simpler musically but could also compute the date of Easter and solve other problems. Bohnert's dissertation provides a comprehensive treatment of Book Eight of the *Musurgia*, tracing its mathematical and philosophical foundations (such as the *Ars combinatoria* of Ramon Llull) and comparing the description of the Arca to the two physical implementations then known. I was unable to inspect Bohnert's actual software, but according to her description, the program implemented the two complete divisions of the Arca (Syntagmata I and II) in a way that followed closely the manual method described by Kircher (Bohnert, 2010, 123–126). Users would select musical permutations one at a time, adjust them, and add them to the musical composition, and then hear the result. The program is not described as fully



Example 1: Automatic setting of *Ave maris stella* by the Arca musarithmica Haskell program, in simple style (Arca MEI output rendered by Verovio)

automatic, however, and Bohnert says it can only create very short settings of a small number of texts. The implementation presented here is to my knowledge the first realisation of the Arca musarithmica to be fully automatic and to be capable of setting texts of any length and type. Exx. 1 and 2 show simple and florid settings of the hymn *Ave maris stella* automatically generated by the ark.

I focus my examination of the ark on the ways that it embodies musical concepts of computation. I consider the components of Kircher's original device as a human-operated system, discuss what was necessary to translate it for use by a digital automaton, and evaluate to what extent the software realises Kircher's computational system. I close with reflections on how Kircher's system reflects widespread views of the nature of music and of compositional craft, and how at the same time it demonstrates a gap between theory and practice.



Example 2: Automatic setting of *Ave maris stella* in florid style (Arca MEI output rendered by Verovio)

2 The Arca as a Human-Optimised Computational System

The original Arca should not be dismissed as a computing device just because it requires a human operator, or because its design may seem simple. A computational system does not require an electronic implementation; in fact, the fundamental modern concepts of computing, Turing machines and Church's lambda calculus, were defined before there existed any means of automating them (Lewis, 1996, 13–24, 35–46, 57–60; Barendregt & Barendsen, 2000, 5–6). If the human operator's role is purely mechanical and does not require intelligent decisions, then the system should still be considered automatic. With some exceptions to be discussed below, the user of the original Arca, like the human operator of a Turing machine, simply follows prescribed rules to transform symbols on paper. Kircher's system is optimised for a human user, not because it depends on human

discretion, but because it enables a human to access the data easily and manipulate it by rule. What may seem a simple structure of lookup tables is, first, more complex than it appears, and second, still a valid computational system. This section will focus on the original specification for the ark as a human-operated system, and will show how each component embodies a musical concept of computation.

As Kircher's illustration (Fig. 1) shows, the ark contains different kinds of musical information on the outside, on the lid, and stored within. The user is supposed to choose a text to set to music, in Latin by default, and prepare the text by dividing it into sections, phrases, words, and syllables. For each segment, the user must select an appropriate style and mood for the setting. Next the operator goes through the text phrase by phrase, and for each selects a *pinax* (rod or slat) from the appropriate *syntagma* (division), based on the intended musical style and the poetic metre of the text. The ark includes three *syntagmata*: the first is for simple, syllabic, homorhythmic counterpoint; while the second is for florid, melismatic counterpoint with imitation and even fugato. The third syntagma theoretically includes a huge range of styles and techniques but Kircher only published a small portion of it, claiming that he reserved the rest in secret for the eyes of "princes and worthy friends" only (Kircher, 1650, II: 184).

From the tables of numbers and rhythmic durations marked on the *pinakes* (plural of the borrowed Greek *pinax*) the user extracts prearranged permutations of numbers and rhythmic values. The user chooses a mood to suit the text from the table of tones (*toni ecclesiastici* or "church keys") on the lid, and then matches numbers to the pitch names and potential accidentals shown there. Finally, the table on the front specifies how to inscribe these notes on paper within acceptable ranges for the four voices. The ark's output is a notated musical composition for four voices (*cantus* or soprano, alto, tenor, and bass).

Kircher's Arca includes most of the elements that make up any modern computational system (Ifrah, 2001). The marked-up lyrical text forms the input; the slats with musical permutations and the lookup tables of pitches and accidentals function as part of a stored program; and the music paper serves as both a temporary processing area (like working memory) and output medium. The data on the ark's pinax tables is encoded in a form that allows for compact storage and quick retrieval. The ark's tables should be understood as data structures whose design implies the algorithms needed to access and manipulate them (Wirth, 1976, xii–xiii). Likewise, the tables map inputs to outputs according to a rule and are therefore one way of expressing functions. The human operator must do the actual processing of course, but the user must do this according to Kircher's prescriptions, equivalent to the program. And except in a few areas, Kircher does provide a complete list of procedures, specifying an algorithm that may not be quite precise enough to meet a rigorous mathematical definition but is nevertheless functional (Knuth, 1997, 1–7). Since the software implementation of Kircher's algorithm does satisfy modern theoretical requirements, I will note where I found it necessary to complete the specification.

2.1 Kircher's Data Structures: Syntagmata, Pinakes, Musarithms

Examining Kircher's data structures more closely will show the ways in which he has encoded musical data in order to make it possible for a human operator to manipulate musical materials in a mechanical way. Inside the actual arca or box there are three groups (the syntagmata) of long slats (the pinakes). The choice of syntagma is determined by the style of text-setting: respectively, simple homophony, florid counterpoint, or mixed. (This software fully implements only the completely specified Syntagma I and II, and it is not clear if Syntagma III could be fully automated based only on Kircher's specification.) Within each syntagma the choice of pinax is a function of the poetic metre of the input text.

The design of the pinakes recalls Napier's bones and similar calculating devices (Ifrah, 2001). The user removes the needed pinakes from the box and arranges them on a work surface to extract data from them. Fig. 2 shows the slat used to set the hymn *Ave maris stella* in simple style, Syntagma I, Pinax 4. Fig. 3 shows the one used to set the same hymn in florid style: Syntagma II, Pinax 2. Each pinax has two components: lists of integers one through eight and lists of note symbols. Kircher uses the term *musarithmos* (musical numbers) primarily to refer to the tables of numbers; for greater precision I call the pitch numbers *vperms* (voice permutations) and the durations *rperms* (rhythm permutations).

PINAX IV.

Iambica Euripedæa penultima longa.

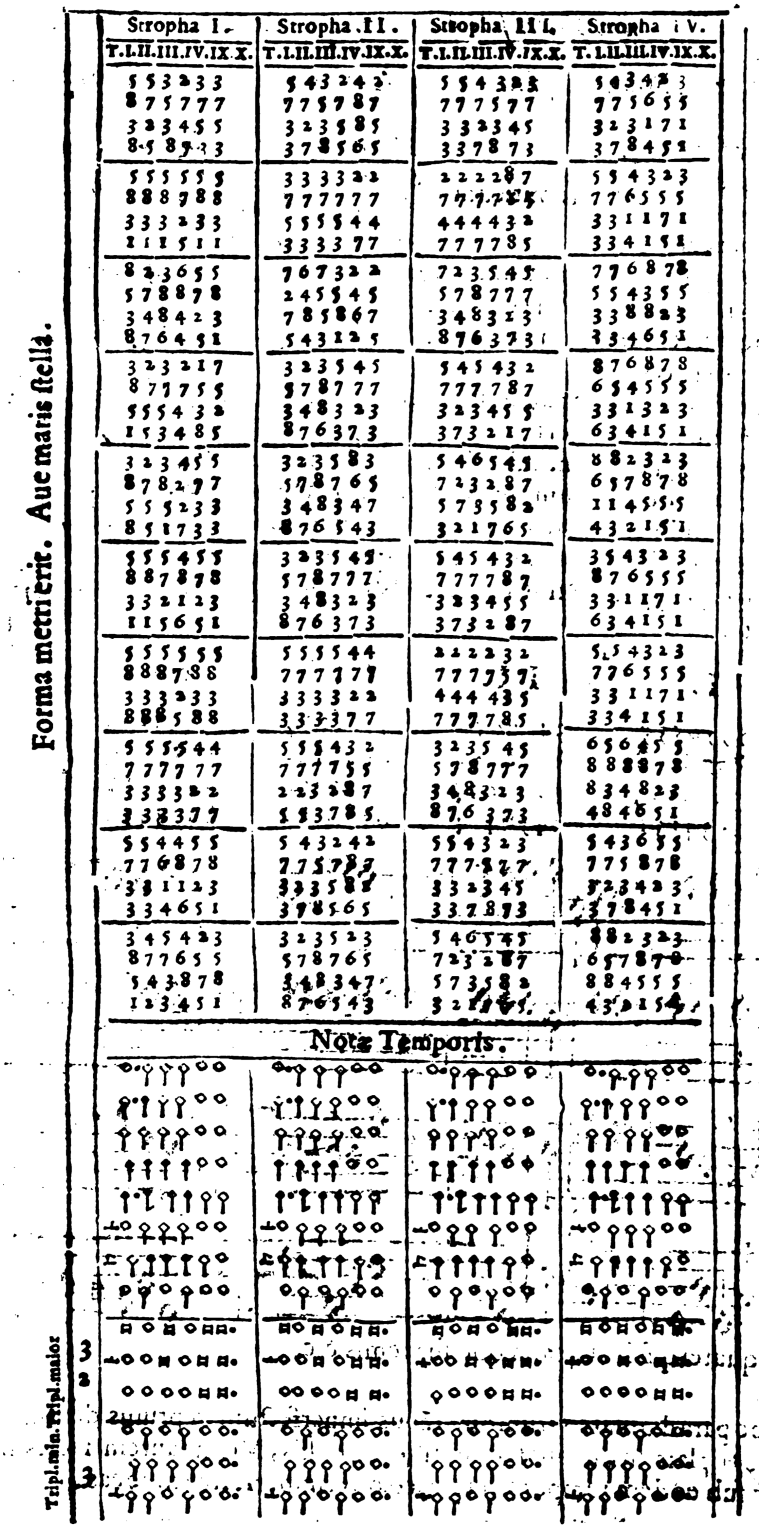


Figure 2: Arca musarithmica, Syntagma I, Pinax 4, in *Musurgia universalis* (1650), Book Two, fol. 83

PINAX II.

Musarithmi Melotheſias Floridæ ſiue Artificioſæ pro metris Iambicis, Euripedæis, Hectaſyllabis.

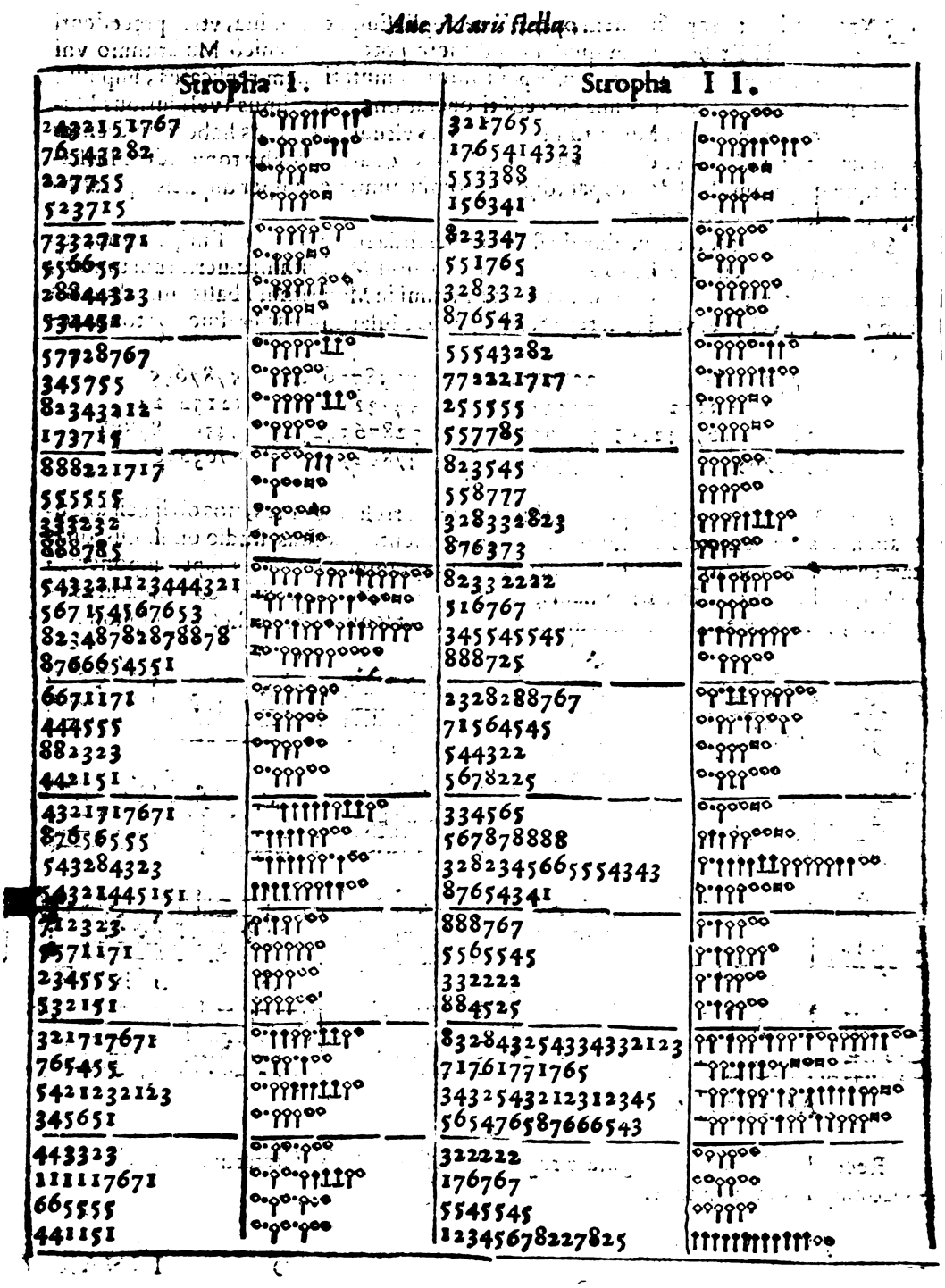


Figure 3: Arca musarithmica, Syntagma II, Pinax 2, in *Musurgia universalis* (1650), Book Two, fol. 106

2.2 Encoding Pitches and Rhythms

The vperms and rperms are arranged differently in each syntagma. In Syntagma I, all the voices use the same rhythm, so Kircher writes separate tables of vperms and rperms: the user pairs a four-voice vperm from the top of the pinax with an rperm on the bottom that is a single list of durations. Since the music in this syntagma is syllabic, the columns correspond to successive syllables in the input text. In Syntagma II, by contrast, Kircher enables complex counterpoint by pairing four-voice vperms with four-voice rperms, so that each voice has its own rhythm. Here the music is melismatic, and there can be a different number of notes in each voice. Since Kircher never provides precise rules for how to underlay the lyrical text in Syntagma II, it would be more accurate to think of these vperm rows as lists of notes rather than as table columns, which is how they are implemented in the software. In most pinakes of both syntagmata, the user is supposed to choose from a different column based on the order of poetic lines in the input text. Kircher calls these *strophae*, which despite the cognate means lines not stanzas.

Kircher does not encode pitch directly in the vperm tables. Instead, the integers are actually lookup keys for the *mensa tonographica* or tone table, where letters A through G plus sharp or flat symbols represent the pitch classes. In modern terms, the numerals represent scale degrees relative to the modal final of a particular tone, so that in tone I, with its final on D, $\hat{5}$ is A. Kircher uses both 1 and 8 to refer to the final, often to express stepwise motion between $\hat{7}$ and $\hat{8}$. Each pinax includes specifications of which tones are acceptable, some in a box to the side, others at the top of each column, and some in the introductory text. At this stage, the numbers only correspond to note names and also to potential accidentals; the octave and duration are still unspecified.

To encode rhythms, Kircher uses note symbols for the different rhythmic values: breve (𝅜), semibreve (𝅝), minim (𝅗𝅥), semiminim (♩), and fusa (♪), along with the corresponding rest symbols. The rperms in Syntagma I are divided into three sections by metre: for duple metre, "triple major", and "triple minor". In his notated realisations Kircher uses the mensuration sign 𝔠 for duple metre, which would normally indicate a metrical pattern in groups of two breves, each divided into two semibreves – i.e., imperfect *tempus*, *alla breve*. In practice, though, about half of his rhythm permutations suggest 𝐂, with groups of two minims. The system does not distinguish between these forms of duple metre, probably out of Kircher's stated desire to maximise the variety of the ark's output. This would not be a problem for someone choosing the permutations intelligently, but since Kircher does not provide a way to distinguish automatically, this implementation of the ark can produce absurd results in duple metre when the implied subdivision shifts abruptly.

In Syntagma I, based on the prior choice of musical metre, the user is supposed to choose one of the three metre categories, and then select one of the rperms from the same column as the chosen vperm. In many pinakes of Syntagma I, though, there is actually only one set of rperms, which is repeated at the bottom of each column. The triple-metre rperms are divided into two sections, one for a ternary proportion of 𝔠 and the other for a proportion of 𝐂. Kircher writes these interchangeably as $\mathfrak{c}_2^3$ and **𝔠3** for "Tripla maior", and as $\mathbf{C}_2^3$ and **𝐂3** for "Tripla minor". The *tactus* or metrical unit in **𝔠3** falls on groups of three imperfect semibreves, in a 3 : 2 proportion to the groups of two semibreves in 𝔠. In **𝐂3** there are groups of three minims, in the same proportion to the binary groups of minims in 𝐂. In Syntagma II all the permutations are in duple metre and the vperms and rperms are paired.

With this separate encoding of pitch and rhythm, Kircher discovered a compact way of representing fairly complex polyphonic music. As Chierotti notes, musicians already used numbers to represent pitches in figured-bass notation, where the numbers stood for intervals above the bass note (Chierotti, 1994). Kircher's rhythm notation also drew on existing practices like the German keyboard tablature of contemporary organists such as Matthias Weckmann, who wrote down letter names in one row with matching duration symbols above (Weckmann, 1980). The Arca user does not need to decode either numbers or duration symbols mentally, because the numbers are lookup keys for the tone table, and the durations need only to be copied directly onto the music paper. For the rhythm, Kircher might also have used integers keyed to a lookup table (as in the software implementation), but his use of duration symbols is more logical and efficient for a human operator.

2.3 Optimised for Automatic Human Operation

For a programmer, Kircher's use of graphical symbols for rhythm seems a hindrance because a digital computer cannnot understand them until they are converted into numbers. For this reason, the

software implementation uses enumeration labels for durations: it is more convenient for the human programmer to write `Sb` for a semibreve while the compiler converts this to an integer internally, but Kircher designed his system for an amusical human operator, for whom no thought at all would be required to simply copy the symbols from the rperm table onto the music paper. In that way Kircher's original graphical encoding is actually more automatic and requires less computation than the software implementation.

Kircher's use of mensural notation further illustrates his human-oriented design. For a computer, symbols must be unambiguous, so the contextual system of mensural notation in ternary metre poses an obstacle. In **C3** (rendered in modern notation as $\frac{3}{2}$), for example, the mensural figure 𝅝 𝅝 is equivalent to the modern 𝅗𝅥. 𝅗𝅥., while the mensural figure 𝅝 𝅝 𝅗𝅥 is equivalent to the modern 𝅗𝅥. 𝅝 𝅗𝅥. In the software implementation, it was simpler just to convert the ternary rperms to their modern equivalent, which in most cases simply required adding a dot to the final duration. In the original system, by contrast, no conversion or calculation was required at all; an "amusical" user could simply copy the durations and let the performers figure out the contextual notation.

For a human user of the physical Arca, once the input parameters are known, all of these data for composing the music can be accessed quite readily by a simple scan of the box and its contents. The human user must know the poetic metre of the text and parse the text into phrases and syllables, and then must choose the intended style, mood, and musical metre. The style tells the user which area of the box to search (which syntagma), and the user then matches the text metre to the label at the top of the appropriate pinax. The user removes the needed pinakes, lays them out in order, and moves them up and down to choose the needed vperm and rperm columns; it only remains to read across the tables and copy down the symbols. In most cases the in-order position of the poetic line determines the choice of column within a pinax. Then for vperms, the selection of a row is a free choice. For rperms in Syntagma I, the choice of musical metre determines which subtable of rperms to select and the particular rperm selection is a free choice. In Syntagma II, all the rperms are in duple metre and they are already matched to the vperms, so only one choice is needed. Kircher's design of the pinax limits its useability, though, because the user must select different rows from a single pinax. The rods would be easier to use if they were built like slide rules with moveable columns.

Once the numbers are copied or noted mentally, the user can turn to the tone table, where the choice of mood determines the tone, and the tone and the vperm integer are the keys for looking up pitch names in the table. The tone table is the least optimised part of the system for any operator (aside from the musical problems with Kircher's whole concept of *toni*, to be discussed below). Although the user must look up information in the table based on the two inputs of tone number and pitch number, the table is keyed by tone number and pitch name. Instead of a simple array lookup, then, one has to scan each column to find the pitch number and then look over to the axis to obtain the pitch letter. Because the table also includes potential *ficta* accidentals, it would make more sense, even with a human user in mind, to put the pitch numbers on the axis and store the pitch letters with their accidentals in the table cells. In terms of computational complexity, Kircher's table requires $O(n)$ time, versus $O(1)$ for a table indexed by tone and pitch numbers (Knuth, 1997, 107–111).

To make matters worse, Kircher actually provides two conflicting tone tables, and in contrast to the one shown in the engraving (Fig. 1), the one given in the text is actually unusable for computation (Kircher, 1650, II: facing 185, II: 51). The other table groups the tones out of numerical order to demonstrate patterns; it includes two conflicting entries for tone 8; and, among other differences, this table lists a tone 4 with an E final, $\sharp\hat{3}$, and $\flat\hat{5}$; while in the engraved table, tone 4 has an A final with $\sharp\hat{3}$ only (Bohnert, 2010, 65–77). The software implementation, therefore, is based on the engraved table. Notably, the copyist of the Arca in Puebla made the same choice and ignored the erroneous table (Cashner, 2022b, p. 54).

Kircher never actually specifies a rule for choosing a particular vperm or rperm within a column. In some places he implies that a random choice would increase variety, while elsewhere he suggests that the user discriminate based on musical considerations. Kircher seems ambivalent about creating a fully automated system, perhaps motivated by theological anxieties about superseding human free will and intelligence, or about chance operations. At the same time, there are numerous ways a human user can simply "pick any row" and achieve semi-random results. The digital automaton can only simulate a free choice by using a pseudo-random number generator, though in this implementation it would still be possible for a superstitious user to supply their own list of permutation choices.

Regardless of the inconsistencies, Kircher's encoding system should be recognised as a symbolic representation of music intended to allow a mathematical approach. Kircher compares his system to the working-out of an algebra problem, in which simply by applying a series of rule-based transformations to a set of symbols, one can achieve previously unexpected results (Kircher, 1650, II: 2). Kircher's effort moved in the same direction as contemporary mathematicians Fermat, who shared Kircher's interest in combinatorics, and Leibniz, with whom Kircher corresponded: his encoding system lifted music into a more abstract realm in which the sonic materials could be manipulated symbolically (Chierotti, 1994, p. 389; Ifrah, 2001, 70, 77–80, 88–92). Like Kircher, those mathematicians did not fully succeed in finding unambiguous symbolic representations or state their proofs according to the same laws of rigour practiced today. Even today the closest we come to a symbolic language for manipulating music abstractly may be MEI-XML; but that system still lacks full support for notating music of Kircher's era, and is designed more for machine readability than for human convenience (Music Encoding Initiative, 2022). Tables might not seem like sophisticated computing devices today but in Kircher's day they were cutting-edge technology in mathematics and engineering. Kircher's student Gaspar Schott published a version of the Arca as part of a larger work that attempted to solve nearly every type of problem with tables (Schott, 1661).

3 Implementing the Ark for a Digital Automaton

In contrast with Kircher's human-oriented design, the software implementation for a digital automaton requires a complete, fully precise algorithm reducible to binary numbers. By number alone, the machine must build the ark, read the input, access the ark data, transform it, and then write it in the output format. The Haskell language, with its pure-functional orientation, guaranteed that the implementation would be a single function (calling other nested functions), while its strict typing made it possible to treat the required data structures as distinct data types. The program uses the following modules:

`Aedifico` **("I build")** This module defines the software versions of Kircher's structures and the basic methods for accessing them.

`Arca musarithmica` This module stores the original data of Kircher's ark in nested vectors and lists of integer values, enumeration values corresponding to the rhythmic durations, and lists of mode scales and other information.

`Lectio` **("I read")** This module creates the system for reading the input texts, which must be written in a custom XML vocabulary. The functions in this module extract the parameters like text metre, music metre, and mode; and parse the text into sections, sentences, phrases, words, and syllables.

`Fortuna` **("Chance")** This module generates the pseudo-random numbers used to select vperms and rperms.

`Cogito` **("I think")** This module is where the actual "musarithmetic" occurs, where vperm numbers are converted into pitches with accidentals, rperms are converted into durations, and octaves are set and adjusted according to Kircher's rules. The module stores all this data in internal data structures that encode music sections, phrases, and pitches (with pitch class, octave, accidental, duration, and syllables).

- `Cogito.Musarithmetic` This submodule includes the functions that calculate musical pitches (such as adding pitches to shift octaves or subtracting to find and test intervals), and the functions for setting the octaves of pitches in order to create an optimal sequence within a given voice range.
- `Cogito.Ficta` This module makes the calculations required to apply Kircher's suggested accidentals, and others required by *musica ficta* conventions. Kircher's rules in this area are underspecified, and as a result this is the most experimental and interpretive component of the software.

`Scribo` **("I write")** This module writes the data generated by the `Cogito` module to a music-notation language that other software can use to put the actual notes on the screen or paper. The submodules output to MEI-XML, and there is also limited capacity for Lilypond output (used for testing).

3.1 The Ark in Haskell Data Types: Syntagmata, Pinakes, Perms

The `Arca` data type contains the same information as the physical ark though slightly rearranged:

```
-- | A vector of 'Syntagma' instances plus the other elements of the physical
-- device (tone table, vocal ranges, information matching tones to pinakes)
-- makes up the full 'Arca'.
data Arca = Arca {
    perms      :: Vector (Syntagma),
    tones      :: ToneList,
    systems    :: ToneSystem,
    pinaxTones :: PinaxToneList,
    ranges     :: VoiceRanges
}
```

(In Haskell, the string after the symbol `::` indicates the data type.) The `ranges` member holds the key information of the *Palimpsest phonotacticum* on the front of the box, while the `tones` and `systems` members correspond to the *Mensa tonographica* or tone table. The member `pinaxTones` supplies the list of legal tones per pinax, which Kircher lists in various places. The Haskell version of the tone table is optimised for quick lookup as a vector of vectors, indexed by tone number and pitch number.

The `perms` member of the `Arca` structure holds the pitch numbers and rhythmic durations from the pinakes. This member stores the contents of the ark box – the syntagmata and pinakes – as a set of nested vectors. Each column is a data type with two members: one vector of vperms and one of rperms. The rperms are further subdivided by metre. At the bottom level, the pitch numbers are stored in lists of `Int` types, while the durations are members of a custom `Dur` enumerator type. The following excerpts show how the Haskell data structures for rhythm permutations fit within the hierarchy of data types:

```
type Syntagma   = Vector (Pinax)
type Pinax      = Vector (Column)
data Column     = Column {
    colVpermTable :: VpermTable,
    colRpermTable :: RpermTable
}
type RpermTable = Vector (RpermMeter)

-- | An 'RpermMeter' includes a vector of 'RpermChoir's all in one metre (see
-- the 'MusicMeter' data type above) and the length of that vector, since
-- Kircher has a variable number of 'Rperm's in the different metres in each
-- column.
data RpermMeter = RpermMeter {
    rpermMax :: Int,            -- ^ length of 'rperms'
    rperms :: Vector (RpermChoir)
}

-- | In Syntagma I, there is only one set of rhythmic permutations that we
-- apply to all four voices of the 'VpermChoir'. But in Syntagma II, there are
-- groups of four 'Rperm's that match up with the four voices.
-- So we make a "choir" as a vector of 'Rperm's, though in Syntagma I this
-- will always just have a single member.
type RpermChoir = Vector (Rperm)

-- | The bottom part of the "rods" contain tables of rhythmic values written
-- with musical notes. We implement this using our 'Dur' data type for the
-- rhythmic values. An 'Rperm' is a list of 'Dur' values.
type Rperm      = [Dur]

-- | Duration values: We use the mensural names; first the base values, then
-- dotted variants, then a series marked as rest values.
data Dur =
      DurNil -- ^ unset
    | Lg     -- ^ longa
    | Br     -- ^ breve
    | Sb     -- ^ semibreve
```

```
    | Mn     -- ^ minim
    | Sm     -- ^ semiminim
    | Fs     -- ^ fusa
    -- ...
    deriving (Enum, Eq, Ord, Show)
```

Using these data types, then, it is possible to build the ark using the data directly from Kircher's tables. The similarity between Kircher's original encoding and that used in the program is clear in this excerpt of the code for the first column of Syntagma I, Pinax 4 (shown in Fig. 2), from the submodule `Arca_musarithmica.Syntagma1.Pinax04`:

```
c0 = (c0v, c0r)
c0v = [
        [ -- 0
            [5, 5, 3, 2, 3, 3],
            [8, 7, 5, 7, 7, 7],
            [3, 2, 3, 4, 5, 5],
            [8, 5, 8, 7, 3, 3]
        ],
        [ -- 1
            [5, 5, 5, 5, 5, 5],
            [8, 8, 8, 7, 8, 8],
            [3, 3, 3, 2, 3, 3],
            [1, 1, 1, 5, 1, 1]
        ],
        -- ...
        [ -- 9
            [3, 4, 5, 4, 2, 3],
            [8, 7, 7, 6, 5, 5],
            [5, 4, 3, 8, 7, 8],
            [1, 2, 3, 4, 5, 1]
        ]
    ]
c0r = [
        [ -- Duple
            [[SbD, Mn, Mn, Mn, Sb, Sb]],
            [[MnD, Sm, Mn, Mn, Sb, Sb]],
            [[Mn, Mn, Mn, Mn, Sb, Sb]],
            -- ...
        ],
        [ -- TripleMajor
            [[Br, Sb, Br, Sb, BrD, BrD]],
            [[SbR, Sb, Sb, Br, Sb, BrD, BrD]],
            [[Sb, Sb, Sb, Sb, Br, BrD]]
        ],
        [ -- TripleMinor
            [[Sb, Mn, Sb, Mn, SbD, SbD]],
            [[Mn, Mn, Mn, Mn, Sb, SbD]],
            [[MnR, Mn, Mn, Sb, Mn, SbD, SbD]]
        ]
    ]
```

The ark is built by converting these nested lists to the appropriate data types and storing them in a single variable, `arca`, which is imported into the main program environment. To go in the other direction and access the vperm and rperm data, the program establishes various forms of lookup functions to map the input parameters (style, text metre, music metre, tone) to the relevant data. For example, to select the proper pinax the computer looks up the text metre to find a pinax label and then looks up the pinax label to find the vector index of that pinax in its syntagma. A number of inconsistencies in Kircher's own structuring of the data and rules for its access – for one, Syntagma I, Pinax 3 actually includes data for two different textual metres – require explicit lookups rather than a simple mapping of enumeration tags to vector indices.

3.2 Reading the Input Text

Before the digital Arca can use these data and algorithms to compose music, it must read the input text prepared with all the information Kircher says is required. The Haskell program accepts input files in a custom XML format, using certain elements of the Text Encoding Initiative vocabulary (Text Encoding Initiative, 2022). The file must have a `head` element with the title and author of the words, followed by a `text` element that includes the words to be set to music, in which the words are divided by syllable with long (or accented) syllables marked. Here is an abridged input text for *Ave maris stella*, showing the possibility of changing style, metre and tone in different sections:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE arkText SYSTEM "arkText.dtd">
<arkText>
  <head>
    <title>Ave maris stella</title>
    <wordsAuthor>Liber Hymnarius</wordsAuthor>
  </head>
  <text>
    <section textMeter="IambicumEuripidaeum" style="Simple"
             musicMeter="TripleMajor" tone="Tone3">
      <lg>
        <l>`A-ve, `ma-ris `stel-la,</l>
        <l>`De-i `ma-ter `al-ma,</l>
        <l>`at-que `sem-per `vi-rgo,</l>
        <l>`fe-lix `coe-li `por-ta.</l>
      </lg>
      <lg>
        <l>`Su-mens `il-lud A-ve</l>
        <l>Gab-ri-`el-is `o-re,</l>
        <l>`fun-da nos in `pa-ce,</l>
        <l>`mu-tans `E-vae `no-men.</l>
      </lg>
      <!-- ... -->
    </section>
    <!-- ... -->
    <section textMeter="IambicumEuripidaeum" style="Florid"
             musicMeter="Duple" tone="Tone9">
      <lg>
        <l>Sit laus `De-o `Pat-ri,</l>
        <l>`sum-mo `Chris-to `de-cus,</l>
        <l>Spi-`ri-tu-i `Sanc-to</l>
        <l>`tri-bus `ho-nor `u-nus.</l>
      </lg>
    </section>
    <section textMeter="Prose" style="Simple"
             musicMeter="TripleMajor" tone="Tone9">
      <lg><l>A-`men.</l></lg>
    </section>
    </text>
  </arkText>
```

The text should be divided into one or more sections, and subsequently into one or more line groups (`lg` elements for stanzas or paragraphs) and lines (`l` elements). The following attributes are required at the start of each section, and set the main parameters needed to run the ark:

`style`: Either `Simple` or `Florid`

`tone`: One of the twelve tones (*toni*) as `Tone1`, `Tone2`, ... `Tone12`

`musicMeter`: `Duple`, `TripleMajor`, or `TripleMinor`

`textMeter`: One of the values in Tab. 1

The `Lectio` module provides the functions needed to parse this text and store it in internal data structures for further processing. These structures (including `Verbum` for a single word and `LyricPhrase` for a group of `Verbum`) store the text grouped in syllables and words, along with information about

Table 1: Poetic metres of Arca input text, specified in the `textMeter` attribute of the XML input encoding

Value	Syllables	Pattern
`Prose`	2–6	Free or irregular
`Adonium`	5	`–– –⏑⏑–⏑ ––`
`Dactylicum`	6	`–– –⏑⏑ –⏑⏑–⏑⏑ ––`
`IambicumEuripidaeum`	6	`–– –⏑–⏑–⏑ ––`
`Anacreonticum`	7	Penultimate long
`IambicumArchilochicum`	8	Penultimate short
`IambicumEnneasyllabicum`	9	Penultimate long
`Enneasyllabicum`	9	Generic
`Decasyllabicum`	10	Penultimate short
`PhaleuciumHendecasyllabicum`	11	Generic
`Hendecasyllabicum`	11	Generic
`Sapphicum`	11 and 5	Quatrains, 11.11.11.5
`Dodecasyllabicum`	12	Penultimate short

the poetic length of each syllable and the penultimate length in each phrase. The list of parsed phrase information is then passed on to the `Cogito` module for setting to music.

The `arca` program, like Kircher's original specification, assumes that its user is capable of preparing the text as required and making the simple decisions about style, mood, and musical metre. The program does go one step further than Kircher does to automate the composition of prose texts, though: where Kircher expects the human user to segment the text by hand into chunks of two to six syllables, the program does this automatically for input marked with `textMeter="Prose"`. A smarter algorithm could certainly be designed, but this one groups the text into the largest chunks possible and puts the longest groups toward the end; this avoids too many choppy phrases and increases coherence.

3.3 Setting a Phrase to Music

The `Cogito` module provides the functions to extract and transform data from the `arca` and match it up with the processed input text. This is the function to compose the music for single phrase of text:

```
-- | Compose the music for a whole 'LyricPhrase' with one permutation from the
-- ark, and package it into a 'MusicPhrase'. Note that this is for a single
-- voice only, not the four SATB voices.
-- Line up pitches and syllables, skipping rests. In Syntagma I, line up text
-- and notes syllabically (one syllable per note); in Syntagma II (florid),
-- lump the text into a single syllable and put it as an incipit text at the
-- beginning of the phrase.
makeMusicPhrase :: Arca
                   -> ArkConfig
                   -> VoiceName
                   -> LyricPhrase
                   -> Perm
                   -> MusicPhrase
makeMusicPhrase arca config voiceID phrase perm = MusicPhrase {
        phraseVoiceID = voiceID,
        notes = theseNotes
    } where

        -- Match up pitches and syllables, skipping rests
        theseNotes = map (\(pitch, syllable) -> Note pitch syllable)
            $ zipFill (music voice) syllables isPitchRest blankSyllable

        voice       = stepwiseVoiceInRange (ranges arca) voiceRaw :: Voice
        voiceRaw    = ark2voice arca config penult sylCount
                          lineCount voiceID perm
```

```
        range       = ranges arca
        penult      = phrasePenultLength phrase
        sylCount    = phraseSylCount phrase
        lineCount   = phrasePosition phrase
        words       = phraseText phrase

        -- In Syntagma II, put the whole phrase of lyrics as a single
        -- syllable under the first note
        syllables = case arkStyle config of
            Simple -> concat $ map makeSyllables words
            Florid -> [Syllable {
                        sylText = unwords $ map verbumText $ phraseText phrase,
                        sylPosition = Only }]
```

Within `makeMusicPhrase`, the function `ark2voice` does the work of accessing the "musarithm" data for a specific voice (e.g., tenor) and transforming it into pitch classes and durations in the appropriate tone. The function `stepwiseVoiceInRange` sets the octaves for the pitch classes in order to produce an optimal voice leading without illicit leaps. These transformation steps involve the most challenging programming in the project and reveal some of the difficulties inherent in Kircher's specification.

3.4 Setting the Octaves for Optimal Voice Leading

In the original system, before notating a pitch a user first needed to know its letter name, duration and accidental, and then the user had to choose the appropriate octave for the pitch based on the range of a particular voice. In the software these data make up the elements of the `Pitch` data type:

```
-- | A 'Pitch' stores the essential information for notating a single note.
data Pitch = Pitch {
    pnum  :: Pnum,          -- ^ Enum for diatonic pitch number
    oct   :: Int,           -- ^ Helmholtz system, middle C = 4
    dur   :: Dur,           -- ^ Duration, one of @Dur@ enum
    accid :: Accid,         -- ^ Accidental
    accidType :: AccidType  -- ^ Type of accidental for display
                            --   ('Written', 'Implicit', or 'Suggested')
} deriving (Show, Eq, Ord)
```

Everything but the octave is pulled directly from the pinax and the tone table. Setting the octave, however, requires more computing than it might seem, and in fact more than Kircher specifies.

Once again, we see a marked difference between computation devices designed for human operators as opposed to what a digital machine requires. Kircher's original *palimpsest phonotacticum* is effectively a graphical computing device for setting the octave of pitches. Before matching a pitch name and rhythm, Kircher tells the user to write a temporary dot on the appropriate staff line as shown on the illustration affixed to the front of the ark. Having located the right series of rhythmic durations, the user is to copy the symbol onto the staff in place of the dot. Kircher does not provide rules for choosing between multiple valid pitch-classes within range; he only says that the user should find the next closest pitch that is on the staff. The ark's chart of staves with clefs, signatures for *cantus durus* or *mollis*, and pitch names written on the lines establishes the possible range and gamut of pitches for a particular voice. A cantus part with a C1 clef, might range from C_3 to D_5 if limited to notes actually on the staff. With a *cantus mollis* (one flat) signature all the Bs in that range are flat. To set the octave for a pitch, then – to know which line to place the dot on – one simply selects from any of that pitch class within range. To find the next pitch, one calculates geometrically the shortest distance on the staff-line graph from the previous pitch to the new pitch, provided that the new pitch is still in range.

Kircher does not give a precise definition of the ranges, but in his example Arca compositions he will extend up to a single ledger line above or below the staves. That would give the voices the following ranges, which are used in the software (in the `_vocalRanges` member of the `arca` structure):

Cantus	A_3	F_5
Alto	D_3	B_4
Tenor	B_2	G_4
Bass	E_2	C_4

Even for a human operator, though, this simple algorithm is not sufficient to establish optimal voice leading within the range. In Kircher's written-out realisations of the musarithms, he does not in fact always choose the smaller interval. Especially in the bass, he sometimes opts for an ascending fifth (especially moving to the final) rather than a descending fourth. Posing more of a problem are those permutations in which there is a long sequence of stepwise movements in one direction (as in Ex. 2), because if the tone transposes this sequence to a position that overlaps the range boundary, it is impossible to realise it without either going out of range or turning a second into a seventh. Kircher warns against intervals that are too large but does not define them precisely. Human users would need at least a little musical knowledge to resolve these situations on paper.

Kircher's specifications were not sufficient to translate them into a computer program. Instead the `Cogito.Musarithmetic` module provides the function `stepwiseVoiceInRange`, which attempts to capture in programming logic the intuitive steps a human user would follow to evaluate the different possible realisations of a vperm in a given range.

```
-- | Find a melody for a voice with an optimal blend of avoiding bad leaps and
-- staying within range. This is the main function used in @Cogito@.
stepwiseVoiceInRange :: VoiceRanges -> Voice -> Voice
stepwiseVoiceInRange ranges v = Voice {
    voiceID = voiceID v,
    music   = adjust
}
    where
        pitches     = music v
        range       = getRange (voiceID v) ranges
        candidates  = pitchCandidates range pitches
        options     = stepwiseTree candidates
        adjust      = bestPath range pitches $ paths [] options
```

We begin with a set of `Pitch` objects with the octave still unset (taken from the pinax tables), and make a nested list containing all the possible octave values for each pitch. Because we know that we might need to go out of range in the end, we first expand the permissible range by a third in each direction. From these permutations we construct a tree (implemented as a left-child/right-sibling binary tree) of every possible permutation of pitches, ending a branch if the interval between parent and child is illegal. We define a legal leap as any interval less than a seventh, or an octave. Traversing the resulting tree produces a list of the possible paths.

To simulate the human user's cultural-stylistic discretion in evaluating the possibilities, we assign each path a "badness" score based on three factors:

1. its ambitus, the total spread from lowest to highest pitch;
2. the sum of all intervals larger than a fourth;
3. for all the pitches that exceed the range, the sum of the interval by which they exceed the range boundary:

```
-- | Calculate weighted "badness" score for a list of pitches.
badness :: VoiceRange -> [Pitch] -> Int
badness range ps = sum [ ambitus ps,
                         sumBigIntervals ps * 2,
                         sumBeyondRange range ps * 10 ]
```

The concept of badness comes from Knuth's TEX program (Knuth, 2011). These scores are weighted and combined, and then the path with the lowest score is chosen (or the first path in order with the lowest score, in case of a tie).

This algorithm yields melodies that fit reasonably within range and avoid awkward leaps. The function does what Kircher says it should do, just in a different way. Kircher's graphical system would need to be adapted to a numeric one in any case, but the program required more specific rules than he provided to evaluate the possibilities. It obviates the need for some of the makeshift remedies that Kircher provides in place of a rigorous algorithm, which include swapping voices.

3.5 Accidentals and Church Keys

The other area in which the software must fill in an incomplete specification is the problem of setting the accidental for each pitch. Kircher's tone table only provides potential accidentals, to be applied according to *musica ficta* conventions in each *tonus*. In practice, the problem of accidentals is a major detriment to making the ark fully automatic, regardless of whether the operator is electronic or human. From Kircher's table of twelve *toni ecclesiastici*, accidentals are derived from the designation of each tone as being in *cantus durus* or *mollis*, and from the accidentals included with the pitch numbers. *Cantus durus* meant there were no accidentals in the signature and the tones were untransposed (with tone I on D); *cantus mollis* meant there was one B flat in the signature and the tones were transposed down a fifth (tone I on G) (Barnett, 2002). Kircher does provide some rules for when to apply his suggested accidentals, but he also gives examples where additional accidentals must be added, and his own notated examples contradict his specifications.

The accidental problem arises not just because of Kircher's incomplete instructions, but because of the structure of the tone table itself. Kircher's table and his rules for using it represent an underappreciated attempt to theorise the rapidly changing, chaotic state of modal and harmonic practice in the seventeenth century. Previous scholars have seen Kircher's tones too simply in terms of modes or keys (Chierotti, 1994; Bumgardner, 2009; Bohnert, 2010). Kircher actually blurs the distinction between the twelve modes defined by Glarean and Zarlino, and the new system of *toni ecclesiastici* or "church keys" (Barnett, 2002). The latter developed in part from keyboardists' practices of playing polyphonic intonations for the eight traditional psalm tones of Gregorian chant, but more and more musicians were coming to think of "tone" as something like a key. Meanwhile there was a separate way of understanding mode in polyphonic composition, based primarily on the final in the lowest voice and the ambitus of the tenor voice (plagal or authentic). But Kircher also demonstrates a growing tendency in the seventeenth century to speak prescriptively of mode as dictating not just the final but also patterns of internal cadences as well as character and affect. Performers using *musica ficta* practices tended to raise "leading tones" at cadences and sharp thirds at the ends of phrases, and with these adjustments along with greater emphasis on bass-line movements by fifth and other elements of tonality, the *toni* increasingly came to resemble either major or minor keys.

Most of the other seventeenth-century classification systems that Barnett describes feature eight church keys instead of Kircher's twelve. Perhaps Kircher was trying to combine the concepts of mode, tone, and church key, or perhaps he did not understand the difference; either way, as Barnett notes, he was hardly alone in his confusion. Practice was changing rapidly and no one had yet provided a consistent theory. The way Kircher codifies the tone system in the Arca should be understood as his contribution to harmonic theory, imperfect as it is.

The tone table is typical of Kircher's paradoxes: he maintains that there are twelve tones out of faithfulness to tradition, but he also concedes that the traditional approach does not fully address modern practice. As in other parts of the book his attempt to synthesise old and new results in something that neither ancients nor moderns would recognise. Although composers of the time wrote differently in each mode or tone, Kircher's system simply transposes the same voice-leading patterns. Had he used full key signatures instead of suggested accidentals, Kircher would have produced something close to the modern system of transposable major and minor keys. As it is, Kircher must limit the selection of tones that can be used with certain pinakes, because not every voice-leading pattern works in every tone, given the different pattern of intervals in each. Kircher saw a variety of *toni* as necessary for conveying distinct affective influences to listeners, and a system with only major or minor keys would have broken consistency with centuries-old traditions of modal affect associations. Kircher's system actually highlights the affective impact of distinct *toni*, because the same musarithm pattern can produce markedly different effects depending on the interval and accidental pattern of the chosen tone.

Moreover, the user is also supposed to add other accidentals as needed in order to avoid illicit melodic intervals in the bass, and thereby to avoid bad harmonic intervals above the bass. The basic rules according to Kircher are these: make all Bs flat if the tone is in *cantus mollis*, and then, before adjusting the other voices, one must first adjust the bass voice according to a table he provides to avoid illicit intervals (changing B–F to B♭–F, for example). Then in the upper voice apply the accidentals by rule: ♭$\hat{6}$ in the table is always flat, but Kircher says that where there is a ♯$\hat{7}$ in the tone table, the user should only apply the sharp when the voice is ascending to $\hat{8}$. In his realisations, though, he

goes beyond these rules: sometimes he raises a $\hat{6}$, often he writes a $\hat{3}$ at cadences, and there are numerous other exceptions. Many corner cases are unclear, such as the common idiom in the florid Syntagma II of $\hat{7}$–$\hat{6}$–$\hat{8}$ in tones with $\hat{7}$ and $\hat{6}$. What if one voice has $\hat{6}$–$\hat{7}$–$\hat{8}$ while another voice (in a voice-exchange pattern) has $\hat{8}$–$\hat{7}$–$\hat{6}$? The top voice would have $\hat{7}$ according to Kircher's simple rule, but the lower voice would have $\hat{7}$ at the same time, making an unacceptable cross relation. It would not be hard for a competent musician of the time to resolve such cases, but Kircher does not provide enough information for a true Tyro to complete this complex task mechanically.

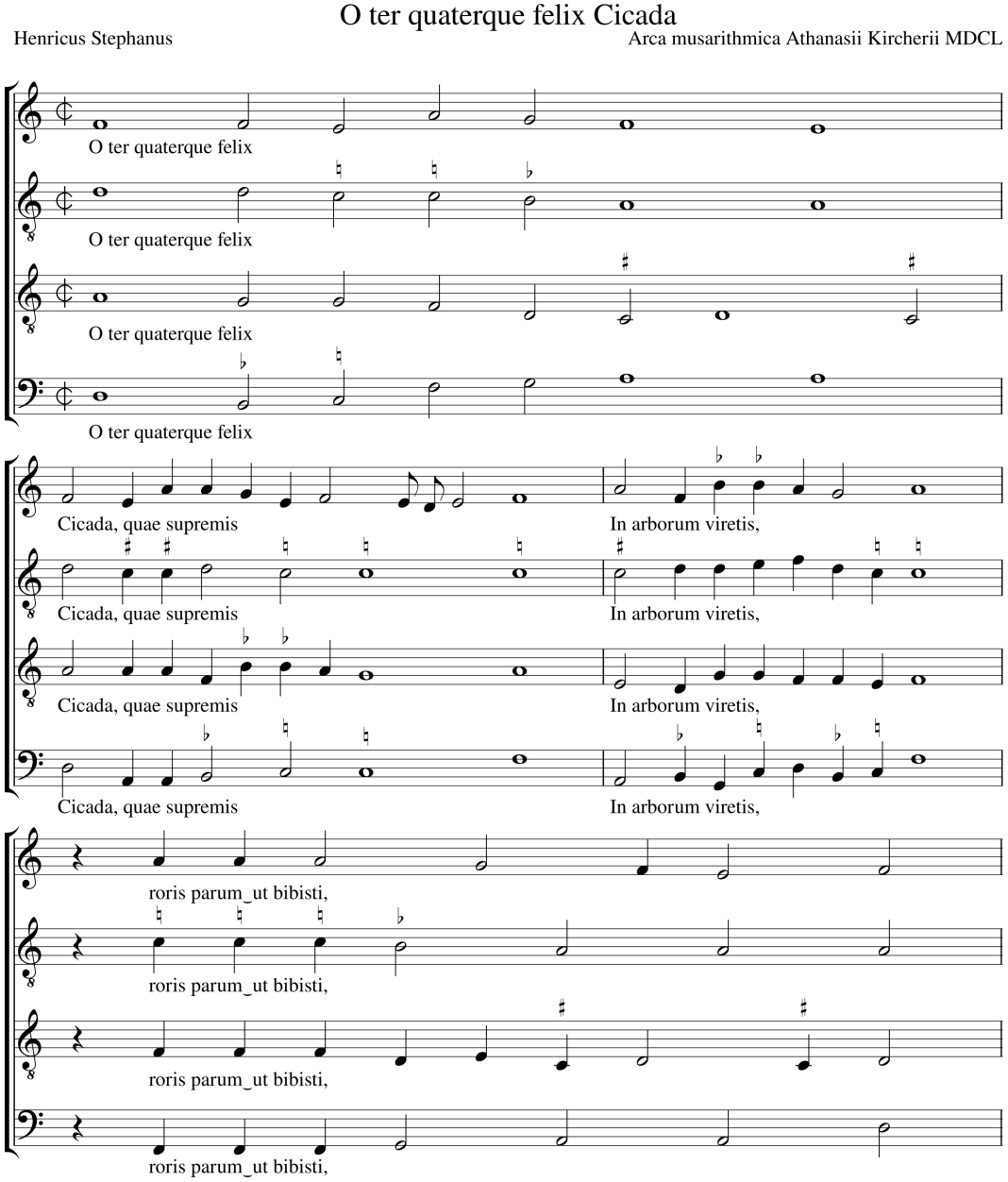
The software implementation uses the rules Kircher does provide but expands them, using a multiple pass system to adjust the accidentals in layers. We first adjust the bass line (including fixing tritones and avoiding $\hat{7}$ when it is not moving to $\hat{8}$) then adjust the upper voices based on their own internal patterns (raising $\hat{7}$ when ascending). Then we adjust the upper voices relative to the bass, and finally adjust the upper voices relative to each other. The adjustments fix cross relations and impossible intervals like an augmented fifth between bass and upper voice.

The automatic setting of *O ter quaterque felix cicada* (Ex. 3) demonstrates a successful output in tone 1 with florid counterpoint. The small accidentals above the staff are the result of *ficta* adjustments: sharps and flats show where the program has applied Kircher's suggested accidentals, and naturals show where it has canceled them. The current algorithm does not handle every case appropriately, but does avoid the most egregious problems. It still remains unclear whether with enough rules the program could automatically produce stylistically appropriate music, or whether some features of Kircher's tone system and musarithm patterns make that impossible.

3.6 Encoding the Music Output

In Kircher's final stage, the user simply copied the music symbols onto staff-ruled paper, but the computer implementation encodes the musical output so that other programs may render it visible and audible. This project uses the XML format of the Music Encoding Initiative, which may easily be converted to other formats like MusicXML and Lilypond or imported into graphical notation software (Music Encoding Initiative, 2022). The software system, then, translates two sources of encoded data – the input text encoded in XML, and the ark data encoded in Kircher's vperm and rperm tables – into a third encoding system. Ex. 1 shows a portion of the automatically generated setting of *Ave maris stella* in simple style as rendered by Verovio. This example represents the music for the first line of text in the Soprano part:

```
<staff n="1" corresp="Cantus">
  <layer n="1">
    <note pname="c" oct="4" dur="breve">
      <accid accid.ges="n"></accid>
      <verse><syl con="d" wordpos="i">A</syl></verse>
    </note>
    <note pname="b" oct="3" dur="1">
      <accid accid.ges="n"></accid>
      <verse><syl con="d" wordpos="t">ve,</syl></verse>
    </note>
    <note pname="c" oct="4" dur="breve">
      <accid accid.ges="n"></accid>
      <verse><syl con="d" wordpos="i">ma</syl></verse>
    </note>
    <note pname="d" oct="4" dur="1">
      <accid accid.ges="n"></accid>
      <verse><syl con="d" wordpos="t">ris</syl></verse>
    </note>
    <note pname="e" oct="4" dur="breve" dots="1">
      <accid accid.ges="n"></accid>
      <verse><syl con="d" wordpos="i">stel</syl></verse>
    </note>
    <note pname="e" oct="4" dur="breve" dots="1">
      <accid accid.ges="n"></accid>
      <verse><syl con="d" wordpos="t">la,</syl></verse>
    </note>
    <barLine></barLine>
```



Example 3: Automatic setting of Stephanus, *O ter quaterque felix Cicada* in tone 1, florid style, with *musica ficta* adjustments (Arca MEI output rendered by Verovio)

```
    <!-- ... -->
  </layer>
</staff>
```

The program's `Note` data type was designed to map easily onto the MEI `note` element, which combines pitch information with a syllable text. The members of the `Pitch` structure translate directly to the XML attributes or subelements of the MEI `note`. To render the whole `Phrase` (one line of input text, set to music by the ark) into MEI, we map `note2mei` over the phrase and add barlines at the `section` divisions marked in the input text. The program takes advantage of MEI's alternative mensural mode, in which music may be structured one voice (MEI `layer`) at a time rather than being organised by measure. The web application uses Verovio to render the MEI to graphical notation (SVG) and sound (via MIDI).

Putting all these elements together, the application's main function builds the ark, reads and parses the input text, and generates a list of random numbers to select permutations for each phrase. The function `makeMusicScore` generates the music and `score2mei` to converts the internal data structures to MEI output. This is the central portion of the application:

```
main :: IO ()
main = do
-- ...
let
    input       = readInput rawInput
    sections    = prepareInput input
    lengths     = inputPhraseLengths sections
    metadata    = arkMetadata input

perms <- inputPerms lengths

let
    score = makeMusicScore arca sections perms
    mei   = score2mei arca metadata score
-- ...
writeFile outfile mei
```

The simple web app (in Javascript and PHP) wraps a convenient user interface around the Haskell program to enable users to select either fully prepared input texts, or make their own choices about style, tone (based on mood), and metre.

4 Musical Computing and Mechanical Composition

In sum, even before its implementation as a computer program, Kircher's Arca musarithmica was already a computational system. The Arca includes multiple kinds of structured data and algorithms, and in the words of Niklaus Wirth's memorable title, *Algorithms + Data Structures = Programs* (Wirth, 1976). The system does not meet every modern definition of a computer, since its data structures are not always consistent and its algorithms not completely specified. Considered as a physical device with a human operator, though, Kircher's system is reasonably optimised and specified fully enough that most people capable of reading the Latin instructions would be able to follow the steps and with enough patience, generate a composition. The ark's structure makes the process nearly automatic for a human user, including the visual indexing of the box's internal structure, the ability to physically move and rearrange the pinakes in their order of use, and the *palimpsest* system for finding in-range pitches graphically. For Kircher's learned readers, parsing and metrifying a Latin poetic text would not perhaps have been an obstacle, but some of the musical decisions required to achieve optimal stepwise voice leading and appropriate accidentals would require more knowledge than an *amusicus* would have.

The digital implementation demonstrates that with some modifications Kircher's system can be made to work as a single function transforming an input text and parameters into an output of encoded music. Most of the modifications are just implementation details needed for a user that at base level only understands arithmetic, Boolean conditionals, and memory registers, though we have seen there are also several areas where greater specification was needed.

Being written as a Haskell program clarifies the structure of flows from input to output. Every function in the program includes a signature showing a series of data types moving from input to output. The arguments to the initial function are shown in series because Haskell functions are actually curried: functions with multiple arguments are actually composed from a series of functions, each of which receives a single input and returns a single output. The `arca` program's primary function takes as input the ark data, the processed input text, and the random permutation numbers, and outputs a data structure for the entire music score:

```
-- | Compose the music for the whole document as a 'MusicScore',
--   pulling the data from the 'Arca'.
makeMusicScore :: Arca -> [LyricSection] -> [SectionPerm] -> MusicScore
makeMusicScore arca lyricSections sectionPerms =
    zipWith (makeMusicChorus arca) lyricSections sectionPerms
```

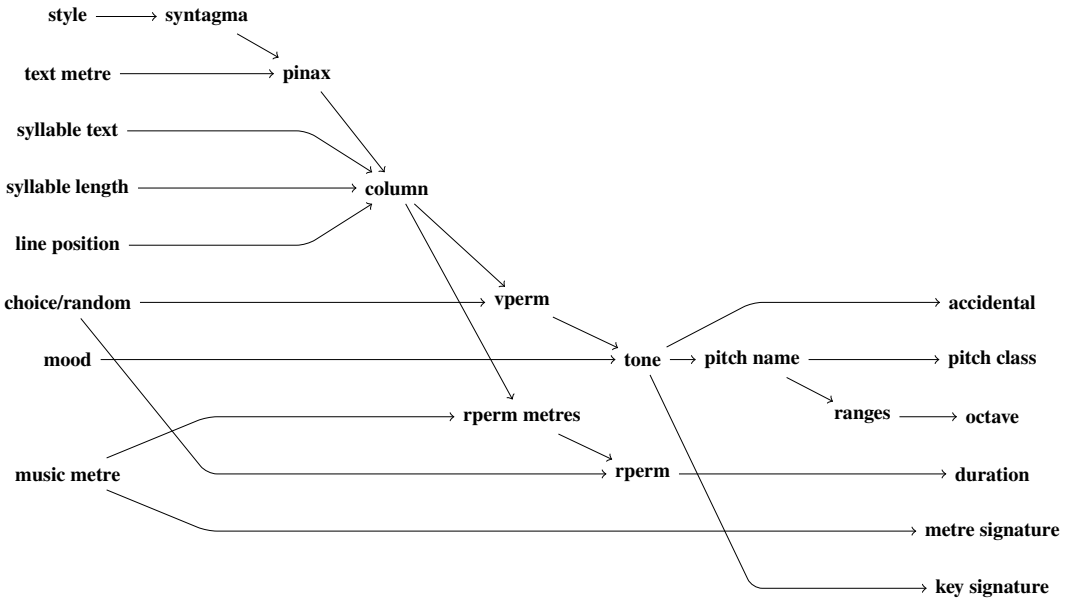


Figure 4: The entire ark system: Flows from inputs through functions to outputs

The flow of inputs to outputs in the digital implementation is the same as in the physical system. The chart in Fig. 4 focuses on the output of a single note, ignoring the larger structural divisions. The chart does not account for the adjustments we must make only after we have put together the whole series of notes (setting the octave and accidentals based on intervals between notes).

Many of the flows in the chart can be seen clearly in the `pair2pitch` function, which takes the pair of pitch and rhythm values from the ark tables and returns a `Pitch` data structure that includes pitch name, accidental, octave and duration:

```
pair2Pitch :: ToneList -> ToneSystem -> Tone -> (Dur, Int) -> Pitch
pair2Pitch toneList systems tone pair
    | isRest thisDur = newRest thisDur
    | otherwise      = newPitch
    where
        newPitch = Pitch {
            pnum      = thisPnumInTone,
            accid     = thisAccid,
            accidType = thisAccidType,
            oct       = 4, -- dummy value, will be adjusted
            dur       = thisDur
        }
        thisPnum       = (snd pair) - 1 -- adjust to 0 index
        thisDur        = fst pair
        thisPnumInTone = fst tonePitch
        thisAccid      = snd tonePitch
        tonePitch      = pnumAccidInTone thisPnum toneList tone

        thisAccidType = case thisAccid of
            Na -> Implicit
            Sh -> Suggested
            Fl -> if isBflatInSignature thisPnumInTone thisAccid tone systems
                    then Implicit
                    else Suggested
            _  -> None
```

The `arca` program is as faithful to Kircher's original design as possible, but it also surpasses the incomplete specification to become a true, fully automatic composition machine. Thus an erstwhile

Table 2: Number of *musica ficta* corrections applied by `arca` software in permissible tones

Input text	Style	Tone 1	2	3	4	5	6	7	8	9	10	11	12	Total	Mean
Ave maris stella	Simple	20	20	20	10					20	0			90	15.0
Ave maris stella	Florid	16	16	14	9	4	4	14	14	16	4	3	4	118	9.8
Ave Regina angelorum	Simple					5	0		7		4		0	16	3.2
Ave Regina angelorum	Florid	27	27	22	11					27	4			118	19.7
Nubibus atris	Simple	8	11	8	6	5	0	3	3	8	5	0	0	57	4.8
Nubibus atris	Florid					0	0	5	5			0	0	10	1.4

"recipe box" full of tables, taken along with Kircher's written specifications, has become a single, complete algorithm for computing music.

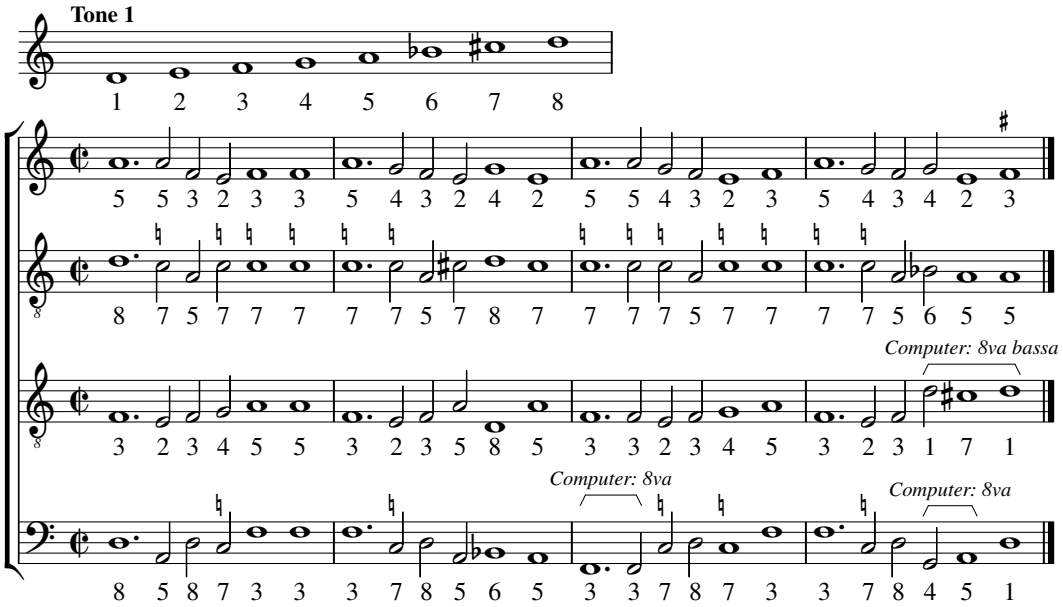
4.1 Testing and Evaluation

I have tested the system by running the program with each of Kircher's paradigm texts and others, including in Spanish and English, that match his poetic metres; and with Latin psalm texts in Prose metre. For each input text I have tried every permissible tone and metre, as well as constructing more complex input by putting different sections in different tones and metres. Several other users have helped me test the web app version, including by writing their own texts (which the web app sets in Prose metre). The program sometimes fails because of errors that still lurk in Kircher's original data or in my transcription; but I have seen no evidence of problems with the basic logic of the program. Problems with the web app usually arise from the interaction between the Haskell executable and third-party software – the Javascript implementation of Verovio notation software and the MIDI engine. (Bug reports are welcome!)

To verify that the Haskell program works as intended and avoids non-deterministic side effects, I created a test application that bypasses the random-number generator in the `Fortuna` module and instead requires the user to specify permutation numbers manually. Since the program only has two non-deterministic elements – the input file and the random choice of permutations – I wrote scripts to run the program repeatedly with the same input file and the same permutation selections and compare the SHA1 checksums of the output files. As expected, the output files were identical.

To evaluate the basic functioning of the program I compared its output with settings I made by hand using Kircher's original materials. Ex. 4 shows my manual setting of *Ave maris stella*, using the first voice and rhythm permutation in each column of Syntagma I, Pinax 4. The example shows how voice permutation numbers match up with Kircher's tone table and highlights where accidentals from that table were altered. Annotations show the two places where the software output differed; both resulted from different choices of octaves in the bass voice. In my version, I opted to have the tenor cross above the alto voice at the end; the computer had the tenor cross under the bass. Both were plausible, and neither, perhaps, was ideal, though certainly by post-1650 standards it would be preferable to preserve the Bass as the lowest voice. Though my own choices of *musica ficta* were more intuitive – e.g., recognising that harmonic movement to F required C naturals – the program, which does not analyse harmony but only applies contextual rules for counterpoint, nevertheless made the same choices. These tests confirmed that the program successfuly finds the appropriate permutations of pitch numbers and rhythm symbols, converts the pitch numbers to pitch names and accidentals according to the tone, fits the pitches into the required vocal ranges while preserving stepwise voice-leading, and applies a reasonable number of *musica ficta* alterations to produce correct output according to common standards of Kircher's time. The program may make different choices than a human user about stepwise voice-leading in range, or about discretionary accidentals, but in most cases the results match up with those of a human with training in music and basic familiarity with late-Renaissance European polyphony.

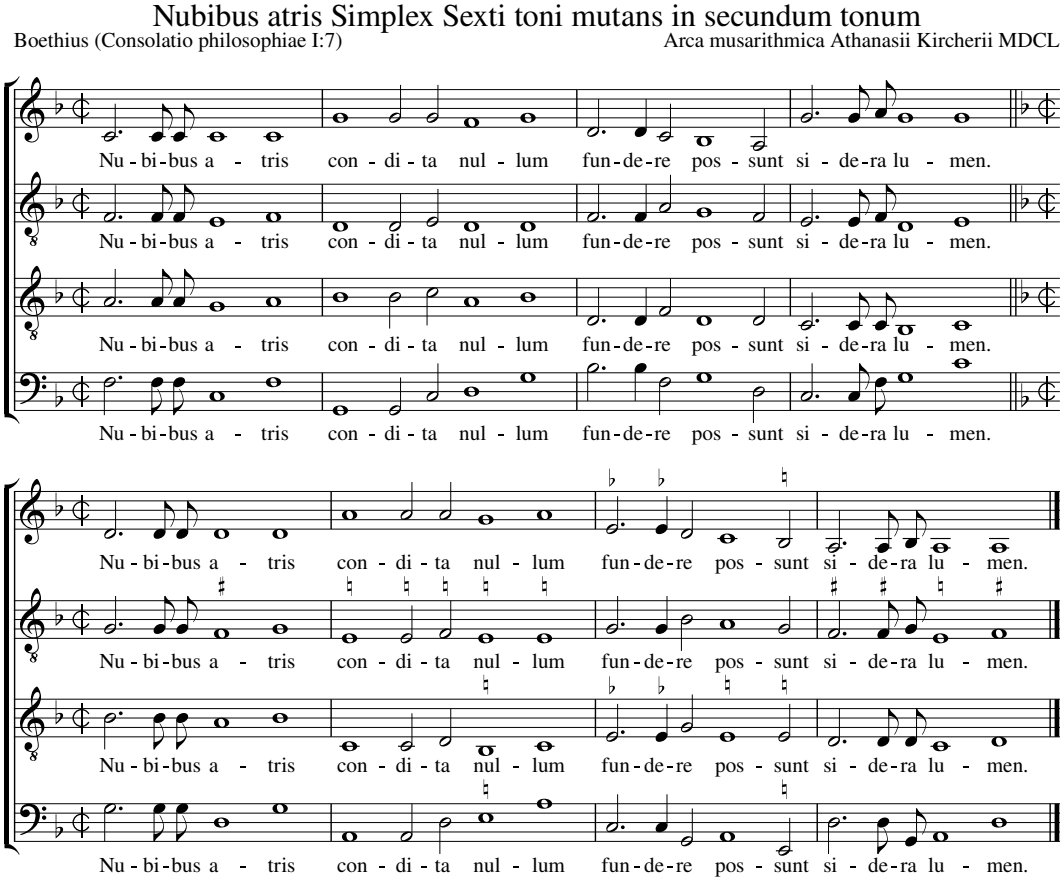
The *musica ficta* functions are far from perfect, but to allow for debugging and improvement, the program logs each adjustment. I ran the program with one stanza of a given input text in both simple and florid style, in every tone that was permissible for that poetic metre and style, and counted the number of adjustments in each. The results (Tab. 2) show that different tones require dramatically different numbers of accidental adjustments. In general there are more adjustments in florid style than simple, primarily because there are more notes in florid style.



Example 4: Manual setting of *Ave maris stella* using the original Arca musarithmica system, compared with automatically-generated software setting (rendered by Lilypond)

It is notable that not every text can be set in every tone, and that some tones seem to be problematic for texts in certain metres, as demonstrated by comparing the mostly reasonable output for *Nubibus atris* in tone 6, with zero *ficta* alterations, to the heavily adjusted and musically uncouth setting in tone 2 (Fig. 5). These disparities in tonal treatment would cause headaches for someone trying to create a complex composition with the Arca that went back and forth between simple and florid style and shifted modes. Another way of looking at it, though, is that Kircher is already very close to a major/minor modal system, and that the further a tone is from major or minor, the more adjustments his rules require in order to bring it back in line with major or minor. In other words, Kircher's system applies complex and arcane rules in order to allow Kircher to maintain the notion of twelve modes while effectively producing near-tonal music in major or minor keys. Kircher's system, as implemented in software, makes it possible to test Kircher's tonal theories in practice; and the results suggest that, typically for Kircher, there is a considerable tension between the traditional theory of twelve modes and the emerging contemporary practice of effectively major and minor tonality, as scholars have already observed in this period (Barnett, 2002). Kircher's tone system deserves much deeper exploration, but this example suggests ways the software implementation could aid in that research.

The pure-functional nature of Haskell and these tests of the software validate that the program does exactly what it says. Does that mean the program is a faithful exemplar of the Arca musarithmica, and that we can therefore make conclusions about the Arca based on the software? To a large but not complete extent, the digital Arca is both functionally equivalent (the same inputs yield the same outputs) and semantically equivalent (the inputs are transformed into outputs using the same process) to Kircher's specification. We have verified that in the simple example of *Ave maris stella*, the program does generate the same output that a human user of the original analogue system would produce. The software algorithm might therefore be considered a subset of the procedures Kircher describes for a human user. In other words, the software does things that Kircher prescribes and in the way that he prescibes them; but Kircher also includes other discretionary procedures, less completely specified, that a human operator could perform additionally. Some of Kircher's algorithms required translation for a digital automaton working with binary numbers, but for the most part the Haskell functions are semantically equivalent to the original tasks: the computer is using different techniques to make the same calculations as a human user. Kircher's full specification is not strict enough or complete enough to translate exactly into a computer program; the system gives more creative and discretionary options to a human user. Kircher seems to have expected a human user to adapt and refine the results, smooth over rough edges, and correct errors in the data tables. The most ambitious



Example 5: Arca setting of Boethius, *Nubibus atris*, with the same manually-selected permutations, comparing automatic *musica ficta* in tone 6 and tone 2 (MEI output rendered by Verovio)

goals of his system, including the ability to match rhetorical features in the poetry with appropriate musical figures, depend on the mysterious third syntagma, the details of which apparently died with Kircher. The program also does a few things *more* than Kircher specifies, such as automatically parsing texts marked as prose, and, in the web-app version, it also performs the music by generating audio.

With some reservations, therefore, we may treat the Arca program as a faithful representation of Kircher's system. That means we can use the software as a tool to explore questions about the original system. Lines of research could include detailed comparison of Kircher's example Arca compositions with software-generated output, and systematic analysis of the individual permutations built into the Arca and comparison with known music, possibly leading to identifying specific sources or schemata-like musical building blocks in other music of the time.

If Kircher's device is to be judged by the output of this software implementation, my tentative conclusion would be that Kircher either oversells the Arca or he was not as knowledgeable about contemporary composition as he claimed to be. The building blocks are premade, and on their own the permutations do sound like fragments of mid-seventeenth-century sacred choral or organ music from Italy or elsewhere in the Catholic world; their contrapuntal figures and resulting harmonic motion are correct and stylistically appropriate (except for sporadic mistakes). The problem is in the core function of the Arca, which is assembling those blocks into a flow of musical ideas. Kircher may have imagined that a human user would select permutations with informed discretion in order to create a good flow, but the computer chooses randomly and the results sound like it. In simple counterpoint, there is no discernable melody, and harmonic centers jump around chaotically. In florid counterpoint, each permutation provides a new point of imitation on a motive unrelated to any previously heard, except when whole permutations repeat; the texture is sometimes more like a fugue,

other times more like a motet, and again the harmonic motion seems unpredictable. Despite Kircher's emphasis on rhetoric, then, the compositions produced by his device do not present any coherent musical ideas (what Bach would later call *inventiones*) that would produce a unified musical discourse (Chierotti, 1994). In terms of prosody, the Arca will put the same rhythms to every line of a poem in a certain metre. For unambitious, regular texts such as the indigenous vernacular hymns that Kircher imagined his Jesuit brothers setting to music using the Arca, this would work fairly well, but it does not adapt to the subtle shifts of accentuation and stress that are found in real-world poems. Rather like early versions of today's AI chatbots, Kircher's system produces a kind of stream-of-consciousness music that works at the moment-to-moment level but does not add up to anything intelligible at the larger level. Nevertheless the program's output is still realistic enough that a musicology student overhearing the web app's generated MIDI organ performance of the Arca output from a room away might come in to ask who composed that odd organ prelude. Given that to my knowledge there is no other system even today that can generate contrapuntal music in a circa 1650 style, that seems like an impressive accomplishment.

Is the Arca musarithmica (in original or digital forms) a creative system? For the software implementation, I would not argue that it is a creative system except in the most literal sense that the program creates new output each time. As noted, the only non-deterministic component is the random selection of permutations. Though deterministic, the algorithm for generating stepwise melodies in range does involve an estimation of stylistic "badness" that could perhaps be considered a kind of artistic discernment on the part of the computer, but that is the only element of the system that would raise such a possibility.

For Kircher's original, human-operated system, however, the answer is more complicated. First, I am not convinced that Kircher intended the system to be as deterministic as the software implementation is. Throughout Kircher's description, he continually blurs the line between algorithmic process and human discretion. He expected users to pick an *appropriate* permutation to fit in a given position in the composition, and he allowed many extensions, such as modulating modes mid-phrase or adapting the permutations through transposition, switching voices, or even taking apart permutations and repeating individual voices to create canonic passages. As is often the case with Kircher, though, what might seem like inconsistency actually points to the difference between his early modern Catholic worldview and ours, and bears witness to the considerable tensions that were destabilising that system in the early years of the Scientific Revolution.

5 The Ark as Encapsulation of Compositional Knowledge

The Arca musarithmica in its original form, I conclude, is a creative system for mechanical composition, but one that embodies distinctive early modern concepts of creativity, composition, and machines. Kircher's concept of creativity was based on finding and combining patterns that were latent in a divinely ordered universe. When Kircher says his goal is to create a system for "artificial" (*artificiosus*) composition, he is using the word in an early modern sense where *art* more readily connoted a practice or technique. The seventeenth-century European or colonial composer was, in modern terms, more an artisan than an artist. An artificial system would be one built on artifice, one created by tools and ingenuity rather than a natural part of the created world. The point of artifice was to imitate the natural world, to draw out its hidden patterns and order.

The basic principle of the ark is to arrange preformulated patterns into a new combination, and as Chierotti argues, this practice is none other than the rhetorician's art of *dispositio* (Chierotti, 1994, 407–409). In the Classical practice of preparing a speech (derived from authorities like Quintillian and Cicero) that stage of arranging an argument was preceded by *inventio*, coming up with an idea. It may seem that the ark eliminates this first step by providing a set of musical ideas which only need to be combined, obviating any expressive or communicative role of the human creator, but such a view would stem from a modern concept of "invention" as original creation. In Kircher's day, invention meant as much "discovery" as creativity. One had to *find* the ideas, and the artisan's craft was displayed in how the craftsman manipulated existing materials. A carpenter needed a pattern, tools, and lumber to begin, but his reputation depended on his ability to turn the raw materials into a finished product according to the plans.

Recent research has shown that European musicians of the seventeenth and eighteenth centuries, as late as J. S. Bach and Mozart, constructed their music from preexisting patterns, according to

well-known, standard techniques (Dreyfus, 1996; Gjerdingen, 2007). Gjerdingen has shown that composers trained in eighteenth-century Neapolitan conservatories were taught to build up a repertoire of conventional voice-leading patterns and to assemble these, with elaboration, into full compositions. Using such practical methods did not inhibit composers from conveying sophisticated poetic concepts and creating dramatic effects; instead, they actually enabled that communication by appealing to common tropes familiar to a community of listeners (Cashner, 2022a). Completely apart from its status as a computational system, Kircher's Arca merits study as a repository of conventional mid-seventeenth century voice-leading patterns.

In this light, then, Kircher's device fits well within an early modern concept of musical craft. As a human-operated system, preserving the user's ability to choose permutations intelligently and adjust them with discretion, the ark merely codifies and reduces the set of combinations available to the user and makes it easier for the human composer to "find" them. By ensuring that each individual combination is already a coherent and rule-compliant musical idea (*inventio*), and by providing an algorithm for arranging the ideas (*dispositio*), the ark eases the burden of a human operator, who no longer has to check each bar for such errors as parallel fifths or octaves. (This would be even more effective if Kircher himself had successfully avoided such errors in the ark!) In that way the ark fulfills the same purpose as other computing devices, from multiplication tables through Napier's bones and slide rules, which is to reduce the number of calculations required by storing precalculated results.

Kircher's procedure of finding appropriate music for one phrase of text at a time does correspond to the text-setting practice of many composers of his age. On the other hand, the lack of any motivic connection between the permutations contradicts a core principle of rhetoric, that an oration should focus on one central theme. For master musicians of Kircher's time, applying that principle to music meant carrying a single idea ("invention") or motive throughout a work (Jacobson, 1982). This discrepancy between Kircher's stated rhetorical ideal and actual contemporary musical-rhetorical practice reflects a tension throughout the *Musurgia* between the idea of music as an orderly, rule-based system, and the idea of music as a form of rhetoric and an expression of human passions.

If Kircher's system reflects widespread attitudes about composition, then, it also reflects widespread misunderstandings. The fact that composers communicated with existing conventions does not mean that their work was purely mechanical. The task of creating music could not be reduced to numerical calculation, and Kircher's system does not in fact succeed in doing so. The machine does something with music that is analogous to mathematical calculation, but it also demonstrates the limits of trying to reduce music to pure number.

The gap between theory and practice also shows when trying to take up Kircher on his promise that the Arca may be used to set texts in any language. Kircher hoped that his fellow Jesuit priests in overseas missions could use the Arca to generate music in local languages (Kircher, 1650, II: 126–141). The `arca` program does show that this is technically possible. The Shakespeare text "If music be the food of love, play on" (from *The Tempest*) included on the website demonstrates the ark's ability to set non-Latin texts with reasonably good results. Though finding or creating non-Latin texts that fit one of Kircher's Classical metres is a major impediment, if the `textMeter` is set to `Prose`, the program will automatically set any kind of text. As a manual system, though, Kircher's Arca cannot have proved very useful to the pragmatic needs of missionary priests in Goa or Nagasaki.

Kircher was never strong on practicalities, in any case, and I would argue that his real goal for the Arca was to encapsulate universal musical knowledge in one device. This idea of epitomising all knowledge of a domain through describing an imaginary computing device is echoed in many respects by Donald Knuth's invention of the MIX computer in *The Art of Computer Programming*, especially given that Knuth's first major application of the device is to permutations (Knuth, 1997, 124–144, 164–175). The Arca epitomises the overall theological argument of the *Musurgia* that music embodies the order of God's creation and reflects the perfection of God's nature, while it also has a powerful effect on the human body and soul, and serves as a form of communication and expresssion.

Like many of his seventeenth-century contemporaries, Kircher understood music as "sounding number", an embodiment of the mathematical order inherent in God's created universe, and a reflection of the perfection of God himself. The Puebla copy of the Arca musarithmica is preserved in a manuscript miscellany of notes on mathematical subjects, including geometry (with attempts to solve the impossible problems of squaring the circle and trisecting the angle), astrology, chronology, and tax accounting (Cashner, 2022b). All of those forms of speculation and inquiry involved number

in the attempt to trace hidden patterns of connection, and this is what Sor Juana termed "Kircherising". Out of the infinite variety of possible combinations of notes and rhythms, the user of the ark selects a specific set of permutations and thus imitates God in his original act of creation.

It does seem that Kircher thought of the Arca musarithmica as a kind of machine, but in saying that we must keep in mind that Kircher and all his contemporaries were working with a preindustrial conception of machines. In 1650 there were no industrial machines on the planet that could, under their own power, generate a given output from a given set of inputs. Kircher simply could not have imagined the seemingly infinite replicability of modern machines. Kircher's idea of musical creativity was mechanical in many ways, but it could never be fully mechanised.

Kircher understood the world to be a well-ordered, rational system, governed by God, in which humans had free will, though their actions were subject to judgment under God's laws. This idea is epitomised by one of the paradigm texts Kircher chose for the Arca, the hymn *Iste confessor Domini*:

Sit salus illi, decus, atque virtus	Let salvation, goodness, and power
Qui supra caeli residens cacumen,	be unto him who, seated over the heavens,
Totius mundi machinam gubernat	*governs the mechanism of the whole world*,
Trius et unus.	three in one.

For Kircher, the whole world was a machine, but in his preindustrial imagination that meant only that it was a harmonious system. Kircher was fascinated, as Newton would be a few decades later, with the idea that the universe ran like a giant clockwork. In the *Musurgia* Kircher used another complex machine, the pipe organ, as a picture of God's act of creation, playing the world into being (Cashner, 2020, 44–52, 159–165). Built into the organ – perhaps the most complex machine yet devised at the time – were all the patterns necessary to make good music, from the harmonic series to modes and scales, temperament, acoustics, and instrument timbres. But a creative musician still needed to sit at the bench and draw out those possibilities. Elsewhere in the *Musurgia* Kircher pictured mechanical organs and other automata, but these were only imitations of God's act of creation, the prelude that he performed to craft the magnificent machine of the world. Likewise the Arca musarithmica could never have been a replacement for human composers, but on the one hand (Kircher imagined) it might have been a practical tool to ease their labours, and on the other hand it stood as a demonstration both of what creativity was (assembling building blocks, working within an orderly system) and what it was not (random, deterministic, automatic).

In our age, the Arca has an additional role as an embodiment of Kircher's own knowledge of music, and a window into early modern concepts of creativity. In this digital implementation, the Arca musarithmica not only enables someone with no knowledge of music to compose, as Kircher intended; it also brings to life a forgotten body of musical knowledge. The ark encodes precalculated knowledge of seventeenth-century music, a library of musical invention, that generates output that one of Kircher's Roman contemporaries like Carissimi would probably have recognised as music, even if somewhat monotonous and with occasional quirks and errors. The awkward results of operating Kircher's system with a digital automaton only highlight how much of a human element was assumed in the original.

There are manifold possibilities for extending Kircher's design, including using machine-learning techniques to improve the permutation data or the composition process, and even replacing the pinax tables with music in other styles or metrical patterns better suited to other languages. The machine-readable MEI output of the Arca program could be analysed by computer and compared to other digitised corpora, which might even reveal the sources for Kircher's permutations. With a digital implementation accessible to anyone on the Internet, musical Tyros and experts alike can do what Kircher hoped, and with no more work or knowledge than the click of a few buttons, they can generate and hear limitless permutations of never-before-heard, newly composed seventeenth-century music.

References

Annibaldi, C. (1995). Froberger in Rome: From Frescobaldi's craftsmanship to Kircher's compositional secrets. *Current Musicology*, *58*, 5–27.

Barendregt, H., & Barendsen, E. (2000). *Introduction to lambda calculus* [Unpublished textbook]. http://www.cs.ru.nl/~herman/onderwijs/provingwithCA/lambda.pdf

Barnett, G. (2002). Tonal organization in seventeenth-century music theory. In T. Christensen (Ed.), *The Cambridge history of Western music theory* (pp. 407–455). Cambridge University Press.

Beuchot Puente, M. (1995). Sor Juana y el hermetismo de Kircher. In *Los empeños: Ensayos en homenaje a Sor Juana Inés de la Cruz* (pp. 1–9). Universidad Nacional Autónoma de México.

Bohnert, A. C. (2010). *Die arca musarithmica Athanasius Kirchers* [Dissertation, Technische Universität Berlin]. Mensch und Buch Verlag.

Boni, E. (2020). L'arca musurgica di Athansius Kircher alla Biblioteca nazionale centrale di Firenze. *Accademie & Biblioteche d'Italia*, *15*(1), 7–13.

Bumgardner, J. (2009). *Kircher's mechanical composer: A software implementation* [Paper presented at the Bridges Conference, Banff]. https://jbum.com/papers/kircher_paper.pdf

Cashner, A. A. (2020). *Hearing faith: Music as theology in the Spanish Empire*. Brill. https://doi.org/10.1163/9789004431997

Cashner, A. A. (2021). *Arca* [Computer program]. https://github.com/andrewacashner/kircher

Cashner, A. A. (2022a). Bringing heavenly music down to earth: Global exchange and local devotion at Segovia Cathedral, 1678. *Music and Letters*, *103*(1), 27–59. https://doi.org/10.1093/ml/gcab106

Cashner, A. A. (2022b). Kircherizers and trisectors: Athanasius Kircher's automatic composition system in the Spanish Empire. *Anuario musical*, (77), 51–75. https://doi.org/10.3989/anuariomusical.2022.i77

Cashner, A. A. (2023). *Arca musarithmica: A device for automatic music composition from 1650*. https://www.arca1650.info

Chierotti, C. M. (1992). La musurgia mirifica di Athanasius Kircher: la composizione musicale alla portata di tutti nell'età barocca. *Musica/realtà*, *13*(37), 107–127.

Chierotti, C. M. (1994). Comporre senza conoscere la musica: Athanasius Kircher e le "Musica mirifica": un singolare esempio di scienza musicale nell'età barocca. *Nuova rivista musicale italiana*, *28*(3), 382–410.

Dreyfus, L. (1996). *Bach and the patterns of invention*. Harvard University Press.

Findlen, P. (Ed.). (2004a). *Athanasius Kircher: The last man who knew everything*. Routledge.

Findlen, P. (2004b). A Jesuit's books in the New World: Athanasius Kircher and his American readers. In P. Findlen (Ed.), *Athanasius Kircher: The last man who knew everything* (pp. 329–364). Routledge.

Fletcher, J. E. (2011). *A study of the life and works of Athanasius Kircher*. Brill.

Gjerdingen, R. O. (2007). *Music in the galant style*. Oxford University Press.

Godwin, J. (2009). *Athanasius Kircher's theater of the world: The life and work of the last man to search for universal knowledge*. Inner Traditions.

Haskell.org Committee. (2024). *Haskell: An advanced, purely functional programming language*. https://www.haskell.org

Ifrah, G. (2001). *The universal history of computing: From the abacus to the quantum computer* (E. F. Harding, Trans.). John Wiley & Sons, Inc.

Irving, D. (2010). *Colonial counterpoint: Music in early modern Manila*. Oxford University Press.

Jacobson, L. (1982). Musical rhetoric in Buxtehude's free organ works. *The Organ Yearbook*, *13*, 60–79.

Kircher, A. (1650). *Musurgia universalis, sive Ars magna consoni et dissoni in X. libros digesta*.

Klotz, S. (1999). "Ars combinatoria" oder "Musik ohne Kopfzerbrechen": Kalküle des Musikalischen von Kircher bis Kirnberger. *Musiktheorie*, *14*(3), 230–245.

Knuth, D. E. (1992). *Literate programming*. Center for the Study of Language and Information.

Knuth, D. E. (1997). *The art of computer programming, volume 1: Fundamental algorithms*. Addison-Wesley.

Knuth, D. E. (2011). *The TeXbook (millenium edition)* [Originally published 1984]. Addison-Wesley.

Lewis, F. D. (1996). *Fundamentals of theoretical computer science* [Unpublished textbook, University of Kentucky]. http://cse.ucdenver.edu/~cscialtman/foundation/Essentials%20of%20Theoretical%20Computer%20Science.pdf

Murata, M. (1999). Music history in the *Musurgia universalis* of Athanasius Kircher. In J. W. O'Malley, S.J., G. A. Bailey, S. J. Harris, & T. F. Kennedy, S.J. (Eds.), *The Jesuits: Cultures, sciences, and the arts 1540–1773* (pp. 190–207). University of Toronto Press.

Music Encoding Initiative. (2022). *Guidelines (4.0.1)*. https://music-encoding.org/guidelines/v4/content/

Osorio Romero, I. (1993). *La luz imaginaria: Epistolario de Atanasio Kircher con los novohispanos*. Universidad Nacional Autónoma de México.
Pangrazi, T. (2009). *La* Musurgia universalis *di Athanasius Kircher: Contenuti, fonti, terminologia*. Leo S. Olschki.
Schott, G. (1661). *Cursus mathematicus sive absoluta omnium mathematicarum disciplinarum encyclopædia, in libros xxviii digesta*.
Text Encoding Initiative. (2022). *P5: Guidelines for electronic text encoding and interchange*. https://tei-c.org/release/doc/tei-p5-doc/en/html/index.html
Trabulse, E. (1984). *El círculo roto*. Fondo de Cultura Económica.
Wald, M. (2006). *Welterkenntnis aus Musik: Athanasius Kirchers "Musurgia universalis" und die Universalwissenschaft im 17. Jahrhundert*. Bärenreiter.
Weckmann, M. (1980). *Choralbearbeitungen: Für Orgel* (W. Breig, Ed.). Bärenreiter.
Wikibooks contributors. (2018). *Haskell*. Wikibooks. https://en.wikibooks.org/wiki/Haskell
Wirth, N. (1976). *Algorithms + data structures = programs*. Prentice-Hall.