https://doi.org/10.48417/technolang.2023.01.08
Research article

Computer Programs and Musical Compositions as Technical Artifacts – Ontological Parallels

Lisa Marianne Borchert (✉)
Darmstadt Technical University, Karolinenpl. 5, Darmstadt, 64289, Germany,
Lisa.borchert@googlemail.com

Abstract

This article compares tonal music and computer programs on a level of technicity, their linguistic properties, and their ontology. In light of Raymond Turner's concept of a technical artifact, both artifacts are technical in the sense that they were created with intention by someone at a certain time. Computer programs as well as music come in physical notes, a symbolic composition, and a physical manifestation when being run or respectively played. They share important properties and similarities in their modes of usage. This investigation presents the parallels between computer programs and tonal music regarding compositionality and normativity. Incorporating Wittgenstein's approach to music as well as the works of Tim Horton and Hanne Appelqvist, the quasi-linguisticness of music and the consequences of this will be shown. Music as well as natural languages and programming languages are composed in a rule-governed way and form meaning through the syntactical combination of context-independent constituent parts. Adopting Raymond Turner's perspective on technical artifacts both are portrayed as such. The result of this ontological investigation is that they show certain parallels in their genesis as well as in the ways they are handled. We will therefore claim, that their ontological status may also exhibit parallels. Hence, the investigation of one of the two may contribute to gaining knowledge about the other's ontological status.

Keywords: Computational Artifacts; Music; Ontology; Technical Artifacts; Wittgenstein; Turner; Grammar; Semantics

Acknowledgment I would like to extend my sincere thanks to Alfred Nordmann for the fruitful exchange about this topic.

Citation: Borchert, L. M. (2023). Computer Programs and Musical Compositions as Technical Artifacts – Ontological Parallels. *Technology and Language*, *4*(1), 111-131. https://doi.org/10.48417/technolang.2023.01.08

УДК 130.2:62
https://doi.org/10.48417/technolang.2023.01.08
Научная статья

Компьютерные программы и музыкальные произведения как технические артефакты – онтологические параллели

Лиза Марианна Борхерт (✉)
Дармштадский технический университет, Каролиненплац 5, Дармштадт, 64289, Германия
Lisa.borchert@googlemail.com

Аннотация
В этой статье сравниваются тональная музыка и компьютерные программы на уровне техники, их лингвистических свойств и онтологии. Ссылаясь на концепцию технического артефакта Рэймонда Тернера, оба артефакта являются техническими в том смысле, что они были созданы кем-то намеренно в определенное время. Компьютерные программы, как и музыка, представлены физическими записями, символической композицией и физическим воплощением при запуске или воспроизведении. Они имеют общие важные свойства и сходство в способах использования. Это исследование представляет параллели между компьютерными программами и тональной музыкой в отношении композиционности и нормативности. Включая подход Витгенштейна к музыке, а также работы Тима Хортона и Ханны Аппельквист, будет показана квазилингвистичность музыки и последствия этого. Музыка, а также естественные языки и языки программирования составляются в соответствии с правилами и формируют смысл посредством синтаксической комбинации независимых от контекста составных частей. Используя точку зрения Рэймонда Тернера на технические артефакты, оба представлются как таковые. Результатом этого онтологического исследования является то, что они обнаруживают определенные параллели в своем происхождении, а также в том, как с ними обращаются. Поэтому мы утверждаем, что их онтологический статус также может иметь параллели. Следовательно, исследование одного из двух может способствовать получению знаний об онтологическом статусе другого.

Ключевые слова: Вычислительные артефакты; Музыка; Онтология; Технические артефакты; Витгенштейн; Тернер; Грамматика; Семантика

Благодарность: Я хотела бы выразить искреннюю благодарность Альфреду Нордманну за плодотворный обмен мнениями по этой теме.

Для цитирования: Borchert, L. M. Computer Programs and Musical Compositions as Technical Artifacts – Ontological Parallels. // Technology and Language. 2023. № 4(1). P. 111-131. https://doi.org/10.48417/technolang.2023.01.08

INTRODUCTION

Computer programs influence our daily lives to an enormous degree. We are monitoring our health and physiology wearing fitness trackers such as wristbands to record our data. We are retrieving news feeds that are based on algorithms that determine our preferences based on our previous behavior. We are creating new career paths by turning our use of video games into an entertaining video for others to see. Computer programs are in motion all around us and still we are unsure to answer the question about when, where and how they exist. One does not see them operate as such and yet they propel or structure our environment. This article suggests striking parallels between computer programs and music, which may be productive for such investigations. From the perspective that both can be described as technical artifacts we will compare the properties that both share, as in being created with intention, compositionality and normativity, to then investigate their ontological status in the third segment. In what way can we see music and computer programs to be the same and where do they differ?

WHAT MAKES A TECHNICAL ARTIFACT 'TECHNICAL'? – THE QUESTION OF TECHNICITY

This article investigates both music and computer programs as technical artifacts. The question what the word 'technical' even means is widely debated. To elaborate the question of technicity should be clarified. How do we understand the term 'technical artifact'? Is it the fact that it is created with intention, that is serves certain means to an end? This is the question the following section is trying to illuminate. Some common opinions on the issue will be presented whereafter the working definition is introduced with which this paper continues..

Computer programs and music can both be created for a certain purpose. While a piece of music may be prone to incorporate a certain type of emotion or tell a story, it could also be seen as a means to influence the listener in a certain way. Marching music may motivate to march, whereas a waltz might invite to dance. Music in movies is supposed to add musical accompaniment and illustrate the mood. We are listening to music to let us feel a certain emotion, to calm us down or make us feel happy. When it comes to computer programs, since we are mostly using them as a means to an end, they apparently represent a tool or an instrument in pursuit of a certain cause. Computer programs enable us to easily get groceries delivered to our door, book flights and let us check the weather forecast or our bank accounts any time of day.

This is not to paint computer programs or music merely as means to an end. They can and most of the time do represent a lot more. They may both be seen as pieces of art. For music this observation might be a little more intuitive than for computer programs. But by investigating the philosophy of computer programs we can easily find sources that do not define computer programs as to even requiring a certain type of purpose or use. Wiliam J. Rapaport for example argues for an approach that accepts lines of code as a program, even if they do not fulfill a certain goal, neither does a program require semantic content in order to be called a program (Rapaport, 2019). Rapaport is pointing out, that

as long as lines of code – as in instructions – are being followed, the structure may be called a program.

Also Nurbay Irmak elaborates, there are digital audio and video files as well as video games, that do not seem to serve any certain practical purpose (Irmak, 2012).

While Irmak in general categorizes computer programs as artifacts, he as well does not define their serving towards a certain practical goal as a necessary property for computer programs. Therefore he refrains from calling computer programs in general 'technical.'

There are other ways to navigate the often drawn line between technical artifacts and what we call art. In his paper "*Die schöne Technik der Verschwendung [The fine Technique of Squandering]*" Alfred Nordmann is pointing out the interconnection of technical and artistic skills in piano music as well as in a show of fireworks (Nordmann 2021). In his elaboration he presents an immanent technicity and artistry in playing the piano as well as in shooting fireworks. Playing the piano as well as coordinating fireworks takes a certain trained skill and technique as well as knowledge about the instruments and materials being used (Nordmann, 2021). Using this knowledge and skill one can then create an artificial world for the audience of the composition (Nordmann, 2021). Nordmann questions the dividing line of artistry versus technicity and opens up a perspective of grasping them as collaboratives rather than opposites (Nordmann, 2021). The so-called goal does not have to be well-defined beforehand, neither does it ever have to be. Creating an artificial world that can be grasped as a whole relies on technicity and artistry in collaboration (Nordmann, 2021). The emerging creation is enabled by two practices that are falsely believed to be opposed to one another, when in reality they complement one another (Nordmann, 2021).

Raymond Turner is categorizing technical artifacts as material objects being created with intention by humans and serving a practical goal (Turner, 2018, p. 25). His definition of technical artifacts is opposed to natural things and occurrings, such as stars in space (Turner, 2018, p. 25). So for Turner the intention through which an artifact is created is a vital part of the definition of a technical artifact. It involves a practical function that needs to be fulfilled. This definition is more bound to the instrumentalist view, that a purpose should be defined, than Nordmann's. Still, we will use Turner's for now to explore what his approach to technical artifacts and computational artifacts might hold for our desired comparison of music and computer programs.

COMPOSITIONALITY

Having discussed the practical properties of a technical artifact we will now focus on structural properties. The first shared structural property of music and computer programs is compositionality. Generally, compositionality means that the semantic meaning of an expression is determined by the structural relation of the entities that make up this expression (Horton, 2001).

In most cases when we talk about compositionality we talk about it in a linguistic context and therefore in combination with "syntax" and "semantics". Rapaport explicates these terms in the following way: The syntax of, for example, a language determines the

correct way that words can be aligned, if the rules of the specific syntax are violated, the expression cannot be called correct regarding the language. So "*the syntactic domain is simply any domain (not necessarily a language) understood solely in terms of its components, their properties, and the relations among them* [...]" (Rapaport, 2019, p. 311-312).

For a language the syntactic domain is its grammar. "*A semantic interpretation of a syntactic domain requires a "semantic" domain: a distinct domain that is also understood solely in terms [...] of its syntax*" (Rapaport, 2019, p.312). The semantic interpretation requires these two domains to share a relation that may connect the domains but is not part of either. This function serves as a connection between the two to enable interpretation through understanding these relations. When one understands the relationships the syntactic domain can be unbundled regarding its entities and their relational status to others in the expression and translated into the semantic domain (Rapaport, 2019).

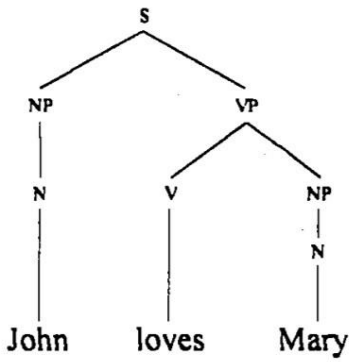

Figure 1. A syntactic analysis of a sentence in a natural language. Here: "John loves Mary." (S = Sentence; N = Noun; V = Verb; NP = Noun Phrase; VP = Verb Phrase). [Taken from Horton 2001, Figure 1].

The way in which the elements of the expression are put together determines the meaning of the entire expression. Deriving the meaning from the construction of the expression may be illustrated using a hierarchical tree graph in figure 1 (Horton, 2001). The sentence "*John loves Mary*" is being analyzed by arranging the different elements of the expression hierarchically in a tree graph in correspondence with their functional role in the expression. While the verb 'loves' forms a verb phrase (VP) with the noun 'Mary', this part is then combined with the noun phrase (NP), which in this case only consists of the noun 'John'. The syntactic combination of these constituents is rule-governed and shows the recursive nature of compositionality that will become important for our later investigations. By combining the individual meanings of the constituents in the way suggested by the rules of the structure we can then form the overall meaning of the sentence. The rule-governed nature of these combinations makes it possible to achieve

complex meanings (Horton, 2001). The property of compositionality is therefore achieved by the combinatorial possibilities of meaning in a certain structured language or any other domain. These foregoing elaborations are taken from Rapaport and Horton. While Rapaport illustrates compositionality in the context of artificial language artifacts such as software, Horton aims to prove the property of compositionality for pieces of music. We will go deeper into Horton's line of argument in the following. He shows that in the light of seeing the individual parts of a tonal structure as structural constituents of a rule-governed complex expression, we may prove that pieces of music – as in tonal structures – do exhibit compositionality.

Context-Indepence of Constituent Parts

Horton first names Context-Indepenence as an important feature of compositionality (Horton 2001). The individual constituents may be taken from their position in an expression and replaced by a structurally identical entity without hurting the structure. Horton (2001) here uses the example of a d minor triad in one expression that may be substituted with any d minor triad that can fulfill the same structural requirements, as in being a subdominant, and this will not hurt the overall tonal content. Horton therefore states that structurally equivalent parts in music are also tonally equivalent. So we will now assume that a triad, or any self-contained constituent part of a tonal structure for that matter, may be called context-independent.

The Relevance of Syntactic Structure

The question about how to combine these constituent parts is answered by the specific structure of the respective syntactical domain (Horton, 2001). Horton analyzes the example expression in figure 2 to illustrate his statements. The components are assigned a certain structural role through the aid of the underlying grammar. Horton here demonstrates how the same sequence of notes may be read in two different ways. In the reconstruction on the left we see how the first three events may be combined to form a tonic phrase consisting of tonic, subdominant and tonic (Horton, 2001). In the reconstruction we can see on the right the first event stays as a tonic, while the events 2-6 are combined to form a more complex tonic articulation. The sequence may thus be read in two ways, both rule-governed and in agreement with the domain. These rules may be formulated in a grammar.

The rule-governed Nature of Syntactic Combination

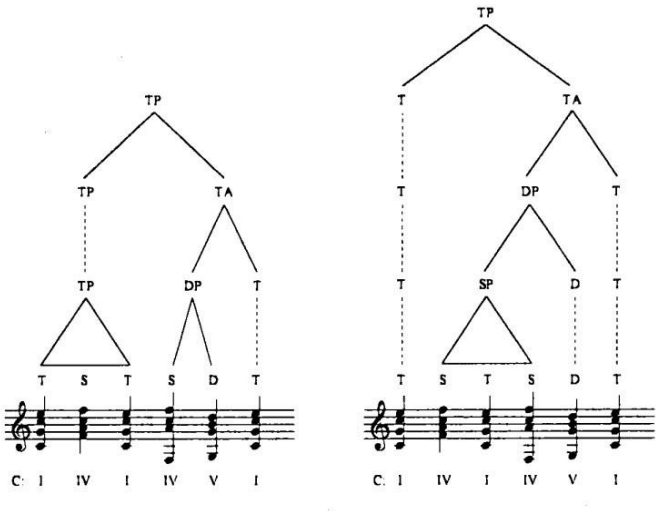

Figure 2. An ambiguous chord sequence. [Taken from Horton 2001, Figure 3].

All structures that are syntactically equivalent to one another may be substituted by a syntactically equivalent structure without changing the content even if they might differ in complexity (Horton, 2001). To illustrate this property we may take a look at two examples Horton (2001) presents. Looking at figure 3 it can be seen how Horton demonstrates the ‘recursive pumping’ of constituent parts of a structure. This can go on infinitely making the emerging structure more and more complex, without changing the underlying tonic articulation it poses. Horton proves in his paper, that these constituent parts then can still take the same role in the grammatical structure of the expression they form.

They may form the same kind of constituent part of the expression their simpler predecessors do. An impressive example of this is shown in figure 3.

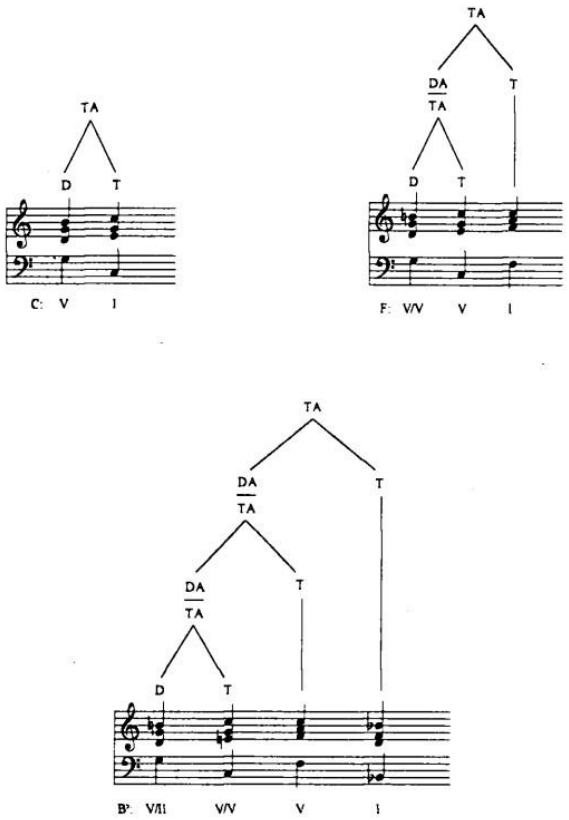

Figure 3. Three musical scores and their syntactic analysis showing the recursive nature of structural relations in tonal music. (DA= Dominant Articulation). [Taken from Horton 2001, Figure 6.].

Their functional properties remain the same while they may get more complex in themselves. The resulting construct may still be (in this case) a tonic articulation. After we have already discovered that self-contained components are context-independent and replaceable by structurally equivalent components, we only need to show now the fact, that the recursive pumping of these components (according to the rules of the grammar) results in also context-independent components, that may be used in the same way as the previous ones (Horton, 2001).

If they can still fulfill the same role in the expression we will assume this to be successful. In figure 4 we see, how this aspect is fulfilled.

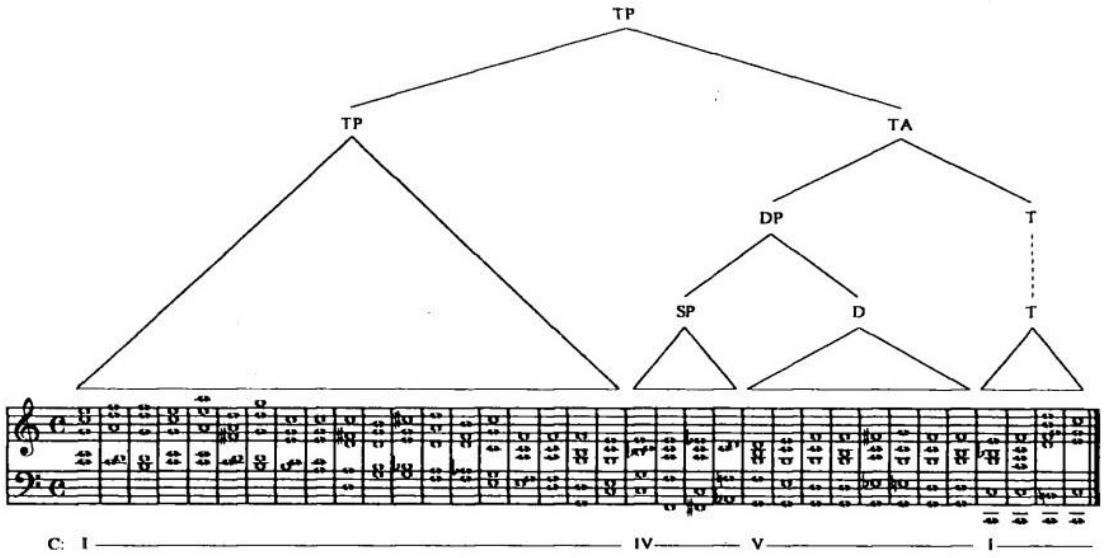

Figure 4. A strcutural description of the large-scale of J.S. Bach's C Major Prelude from Book I of "The Well-Tempered Clavier". [Taken from Horton 2001, Figure 7.]

The combinatorial possibilities remain the same, however complex the individual components may be. With another example (figure 4), Horton shows "that the same combinatorial mechanisms apply equally to constituents that differ widely in their structural complexity [...]" (Horton, 2001, p. 146). While two bars that come in a sequence have a certain structural relation to one another, let us say posing as the subdominant and then the dominant, more complex constructs may also have the exact same relation to one another. As we can see in bars 20-23 (subdominant) and bars 24-31 (dominant) in figure 4 these chords share the same syntactic relation (Horton, 2001). Their complexity levels do not interfere with their structural roles.

Establishing these three properties, Horton (2001) has explicated the assumption of compositionality for tonal music. So it may be assumed that music has a quasi-linguistic form. Horton makes it clear how music and language may correlate structurally, but also where they differ. To understand the meaning of a piece of tonal music we rely on the compositionality of music to analyze its parts and overall structure, but we are missing the conceptuality that language inhabits. While certainly their content is understood in terms of the syntactic principles that underlie the structure and the content of their constituent parts, tonal structures may be understood without knowledge of the characterizing concepts. To understand language we need to understand the concepts that are carried through it. This is not the case for tonal music (Horton 2001).

Compositionality of Computer Programs

In this paper it will be taken as fact that computer languages find their basis in language systems and therefore rely on the principles of such, hence exhibiting compositionality. Computer programs are written in formal and artificial languages. Their structure is based on mathematical notation systems, such as the lambda-calculus for Haskell and Ocaml. Compositionality therefore is already their precondition and a requirement for their creation. Therefore they must exhibit compositionality.

Still, we are going to look into some remarks by Raymond Turner (2018) on the role of semantics (pp. 77-81). Turner states that programmers need to be able to determine whether a code is fulfilling a certain goal in order to be able to program in the first place. They need to be enabled to see the semantic impact of a construct of a certain programming language (p. 77). This requires the property of compositionality (pp. 80-83). For Turner this means that for all complex expressions in a language their semantic meaning is defined through their structure in combination with the meanings of their constituent parts (p. 80). This property is especially important when it comes to programming languages. Compositional semantics allows for the substitution of equivalent entities for one another, as we have also seen in Horton's elaborations (p. 81). If a semantically well-defined entity is denotationally equivalent to another one, they can be substituted for one another without changing the semantic meaning of the expression (p. 81). This property is analogous to Horton's example of the substitution of certain musical structures for others. We can substitute the constituent part X for Y, if they share the same functional properties. If they share the same type or produce the same output for the same inputs, and therefore realize the same function, we can use either of them and the meaning of the whole expression remains the same. For example we can substitute 2x for (4*x)/2 and the expression would stay the same, as both may receive, for example, integers and put out integers. Here, also the input-output table would be the exact same. These two expressions do of course vary in cost and elegance, but will produce the same output. It is important to highlight the fact that for programming languages low-cost programming is of relevance for good practice while this aspect may be less important in natural languages.

Moreover this property makes structural induction possible for a language. This means that having shown a property for a certain statement we can use formal reasoning to also prove it for a more complex structure. It is a useful mechanism for structural reasoning about the language as a whole (Turner, 2018, p. 81). This works in a way opposite to the 'pumping' the tonal structures which were referred to in figure 3. After having proven that the expression X has property A, we can also show it for a more complex expression Y whether it fulfills the same functional role. Rules can be proven for all valid structures in a language, however complex they might be. But what does 'valid' regarding a language even mean?

NORMATIVITY

The property of normativity allows for determining whether an expression is valid or not. Normativity – as in the decidability of correctness – applies to languages, as in computer programs as well as other realms. Normativity means that there is a rule or axiom system that regulates what a valid expression is and what is not. These rules apply without conditions. They guide the design process and make it possible to judge the correctness or rather the correct use of the language or the music.

Normativity in Computer Programs

We will look into two perspectives of normativity in computer programs. On the one hand the structural or formal normativity – the question whether an expression is correct regarding the underlying grammar. And on the other hand the normativity regarding the relation to the underlying semantic sense a given program aims to implement – the content-wise normativity.

Formal Normativity – Normativity of Syntax in Computer Programs

Turner cites Boghossian (1989) in the 9th chapter of his book and derives Boghossian's definition of normativity:

> The fact that the expression means something implies, that is, a whole set of normative truths about my behaviour with that expression: namely that my use of it is correct in application to certain objects and not in application to others. (p. 513)

Regarding programming languages, normativity of meaning therefore means "that any semantic account must provide a criterion for correctness" (Turner, 2018, p. 79). Especially for formal languages these norms facilitate the design of compilers (pp. 79-80). A compiler is the instance that examines the code it gets as an input for correctness and then translates it into a lower level language for lower instances to execute the assignments. Finally, compiler architectures facilitate the step-by-step translation from higher languages like Java to lower languages and pass this down to the next layer until we reach the level of machine language that directly corresponds with the physical processes in the device that is called a computer. The compiler is therefore the instance that evaluates whether an expression is correct and if this is the case translates the expressions into expressions in another language, maintaining its correctness (pp. 79-80). Determining a formally correct expression is accomplished by only consulting the normative rule system, while determining correctness in terms of meaning requires a lot more effort.

Content-wise Normativity – Normativity of Semantics in Computer Programs

Turner (2018) states that the evaluation of the correctness of the meaning would represent a complex and tedious task (pp. 205-206). Going from the step of specification to the symbolic program, hence being in the design phase, we are taking paths that are implying certain assumptions and preconditions as well as making decisions on certain details. The resulting symbolic program must then later be compared to the original idea

in the specification in order to determine whether this phase has been successful. Verifying these steps is conceivably complex. It requires a the semantic understanding what the specification is and which key aspects need to be preserved in the symbolic program. Turner names four challenges we are facing when aiming to prove the correlatedness of a program with its specification: the mathematical, the mechanical, the pragmatic and the scientific challenge (p. 205-212). The focus in the upcoming section will be on the first three challenges in order to illustrate the complexity of judging content-wise correctness.

The mathematical challenge concerns the workload, or cost in IT-jargon. Especially when it comes to bigger computer programs than the example of the GCD, the formal proof will be complex and take a long time to be carried out. (Turner 2018, pp. 205-206). The result of such a process then is only valid for the one entity it was carried out for and will not hold any general knowledge gain, even though Turner points out that this is debatable (pp. 206-207). While proofs in the Cartesian tradition demand the forth-bringing of general mathematical knowledge, the Leibnizian perspective argues for a mechanical and meticulous way of carrying out mathematical proof, neglecting the property of bringing forth generalizable knowledge. The emergence of general mathematical knowledge (in the Cartesian view) is achieved by deduction from normative, valid axioms, assuring that the arising rules will also have normative character. Mathematical proof according to the Cartesian notion involves abstraction and generalization (p. 206). The Leibnizian proofs, such as the formal proofs of construction and verification which Vladimir Voevodsky argues for, focus on the procedure of working in small steps and being easy to comprehend. Voevodsky sees the future of mathematical proof in the implementation of computer systems supporting automated ways of proving the correlation of a program with its specification. (Turner, 2018, p. 207 and Voevodsky, 2010).

With this prospect Turner presents to us the second challenge, the mechanical challenge (Turner, 2018, pp. 207-209). We do have the possibility to check a computer program's correlation by using another computer program to evaluate it. If we can now judge the correctness of a program with the aid of another computer aided system, who will inspect this system and prove its correctness (pp. 207-208)? This poses the same challenge as in translating a problem of meaning of one language into another one. It is only shifting the problem, but not solving it (pp. 207-208). "So, we have replaced the correctness problem for one program by another, and we have the beginning of an infinite regress" (pp. 208). Where is the point at which one can settle semantic correctness? No computer system itself can solve this question.

The third challenge Turner mentions revolves around the empirical testing of a program (Turner, 2018, pp. 209-210). Testing a program serves to verify for certain inputs the exhibition of the desired behavior. This is easier to do and goes without having to face thousands of lines of code (Turner, 2018, p. 209). The tests then must be designed in a manner that would cover (the most relevant) model cases. So in reality most computer programs are tested empirically and not proven formally. When testing you can only find bugs, as in mistakes in the code. This is cheaper concerning resources and more pragmatic. Via empirical testing we can never fully verify the program for every possible

input. If it results in an unwanted output, we can fix it. But as long as we do not enter a certain input we will not know if this might produce an unintended behavior. Also we can only empirically test the specific physical implementation, not the symbolic program directly (pp. 209-210). Proving the correctness of a program regarding its aim, its specification, therefore still poses a tedious task.

Normativity in Music

Normativity, as stated before, is a property a variety of crafts exhibit. And even more so than other realms of art, music comes with a range of rules that exceed genres. Even people who have no idea about musical theory are able to hear if there is something wrong with the sounds they hear – may it be a wrong tune in a structure or wrong timing in a beat (Appelqvist, 2013). Such disagreements between what we hear and what we think we should hear leave people alerted (Wittgenstein, 1967, LC I 15). This observation sheds light on the normative nature of tonal music.

Normativity in Form – Normativity of Syntax in Music

To answer the question whether music also follows normative rules, we will look into Hanne Appelqvist's elaborations on musical grammar and the philosophy of Ludwig Wittgenstein. Appelqvist (2013) describes a Kantian understanding of Wittgenstein's discussion of language and music. She highlights that Wittgenstein holds music to be rule-governed, in the same way language is. "He [Wittgenstein] states, that correctness is an aesthetic attribute far more important than beauty and compares the understanding of a sentence to the understanding of a musical theme, thereby suggesting that musical understanding too is a form of rule-following" (Appelqvist, 2013, p. 299). Using Wittgenstein's philosophy Appelqvist shows us that music, just like language, is a rule-following enterprise and by no means arbitrary. In the Tractatus Wittgenstein point out this analogy himself when he describes the gramophone record, musical thoughts and notations, and also the sound waves produced by the record being played as having an internal relationship to one another, that exists between language and the world (Wittgenstein, 1922, TLP 4.014). "The gramophone record, the musical thought, the score, the waves of sound, all stand to one another in that pictorial internal relation, which holds between language and the world. To all of them the logical structure is common". He further elaborates that they share the property of being logically constructed (Wittgenstein, 1922, TLP, 4.014). Being constructed in a formal way brings the constituents of the record, the notations and the thoughts of music together to exist as one. They are all constructed following a certain set of rules, just as language and the world are. In the following section of the Tractatus, namely 4.141, Wittgenstein describes the inner similarity between the musical score and the audible symphony on the gramophone record.

> In the fact that there is a general rule by which the musician is able to read the symphony out of the score, and that there is a rule by which one could reconstruct the symphony from the line on a gramophone record and from this again-by means of the first rule-construct the score, herein lies the internal similarity between these things which at first sight seem to be entirely different. And the rule is the law of

> projection which projects the symphony into the language of the musical score. It is the rule of translation of this language into the language of the gramophone record. (Wittgenstein, 1922, TLP, 4.0141).

He calls this the law of projection meaning that one of these instances may be projected into the other following the rules (Wittgenstein, 1922, TLP, 4.0141). This possibility implies the existence of an underlying normative set of rules. And "[W]ithout knowledge of the rules of an art form one cannot make aesthetic judgements" (Appelqvist, 2013 referring to LC I 15).

Appelqvist further summarizes that a musical theme, just like a mathematical sentence has meaning, but only in the sense of showing us something about the world (Appelqvist 2013). Wittgenstein proposed that with the aid of our knowledge about the nature of logic we may gain knowledge about the nature of music (Appelqvist, 2013). "A tune is a kind of tautology, it is complete in itself, it satisfies itself" (Wittgenstein, 1961 NB, 40). And "[m]usical themes are in a certain sense propositions. Knowledge of the nature of logic will for this reason lead to knowledge of the nature of music." (Wittgenstein, 1961, NB 40). Wittgenstein's tautologies characteristically lack sense as they are empirically empty, are always true and are complete in themselves (Appelqvist, 2013). For example the axioms underlying our logic are tautologies too and "show the logical form of language" (Appelqvist, 2013). "The propositions of logic are tautologies." (Wittgenstein, 1922, TLP 6.1) – "The propositions of logic therefore say nothing. (They are the analytical propositions.)" (Wittgenstein, 1922, TLP 6.11). They cannot show us any content of reality but can only reveal the logical structure of the world (Appelqvist, 2013). Since logic itself cannot be shown, it can only be shown in propositions (Appelqvist 2013).

> The fact that the propositions of logic are tautologies shows the formal-logical-properties of language, of the world. That its constituent parts connected together in this way give a tautology characterizes the logic of its constituent parts. In order that propositions connected together in a definite way may give a tautology they must have definite properties of structure. That they give a tautology when so connected shows therefore that they possess these properties of structure." (Wittgenstein, 1922, TLP 6.12).

So the musical tune itself does not formulate any content-wise proposition about the world, but can only represent its own form (Appelqvist, 2013). Hence, music as well as logical propositions show the so-called logic of the world, that represents its essence according to Wittgenstein. Exhibiting a logical structure is a necessary requirement for having content for Wittgenstein (Appelqvist 2013). Only within a clear structure we can form a complex meaning from the combination of individual meanings. A sentence or here a proposition by itself is not just a jumble of words. And also a theme in music is not a jumble of tones, hence it is also structured in a certain way (Wittgenstein, 1922, TLP, 3.141). In most translations the word used for "Gemisch" is "mixture"; here we translate it to "jumble" to stress the structuredness or rather non-structuredness of the German expression "Gemisch". And this structure is made possible by their shared logical form (Appelqvist, 2013). As Wittgenstein describes it, the constituents match with one another

and form a whole in the same way as links of a chain do (Appelqvist, 2013 and Wittgenstein, 1922, TLP 2.03.). Here we can experience the logic in the rules that make this possible – logic is apriori (Appelqvist 2013).

Self-evidence, of which Russell has

> said so much, can only be discarded in logic by language itself preventing every logical mistake. That logic is a priori consists in the fact that we cannot think illogically. (Wittgenstein, 1922, TLP 5.4731)

While a musical tune does not exhibit any content that may correspond to the world, it shows through itself by way of being a rule-governed structure the logical form of the world (Appelqvist, 2013). But there is a difference between logical propositions and musical tunes. Musical tunes are bipolar. By this, Appelqvist means that descriptions, pictures and other means of communication represent something in the real world and have a projective relation to reality. Being bipolar, the musical tune cannot do that. Musical tunes have no representative character and no counterpart in the real world that they would relate to (Appelqvist, 2013).

Following these statements we may assume that music is rule-governed.

Normativity referring to Content – Semantic Normativity in Music

Investigating correctness is not as relevant for music as it is for a computer program. As long as the structure itself follows the rules of the art, it may not be questioned. Moreover since music does not correlate directly with concepts in the world in the way language does accordingly there is no way of formally proving any correspondence. If we desire a way to do so, we might have to turn to empirical testing in an analogous manner to the way we verify a program empirically, e.g. by playing the music to an audience. This endeavor may not be *formally* decidable, or verifiable for music, however it is able to show the successful outcome of a piece of music. Also a piece of music, if played as conventionally written, does not accept different inputs and therefore does not produce different outputs. Once written, it is played in the exact same way each time, if played correctly. The question whether it is being played correctly is decidable again. Since we have seen that the direct correlation of music and things in the world does not exist in the way it does for languages, we cannot unambiguously judge their correspondence on the criteria of correctness.

Using Raymond Turner's Understanding of Technical Artifacts on Computer Programs and Music

Since we have examined the quasi-linguistic properties of music, the aspect of compositionality and normativity, we can now move on to the investigation of their ontology. For this we are going to give an outline of Turner's approach to the genesis of technical artifacts to then apply it to the emergence also of a musical composition.

The notion that music and computer programs are comparable on multiple levels is not entirely new. Among others, Nurbay Irmak describes in his article that both computer programs and pieces of music may be classified as abstract artifacts. He uses the word 'abstract' to underline the importance of the non-physical qualities of the two (Irmak,

2012). They can therefore not be reduced to their physical form but are reliant on it to exist in the first place. They cannot be reduced to their physical manifestations. Both are created intentionally and exist only after a certain point in time (Irmak, 2012).

Raymond Turner sheds light on the notion that not only do the two in some way depend on a physical manifestation (be it notes on a sheet of paper or being run or played on a certain device), their relation enables them to be integrated into a single definition of technical artifacts. Turner (2018) describes technical artifacts to exist in a trinity of specification, structure and artifact (p. 52). This construction unifies the widely stated opposition of the physical and ideal understanding of technical artifacts. We want to argue that his approach also contributes to an understanding of music. In agreement with what we learned from Irmak, Turner also states that computer programs also exist only after a certain point in time, they do not exist as immanent ideas (Irmak, 2012 and Turner, 2018, pp. 25-26). Just as music does. This allows us to incorporate the ideas and thoughts that go into a technical artifact.

Raymond Turner's Genesis of a Computer Program as a Technical Artifact

In chapter five of his book on "*Computational Artifacts: Towards a Philosophy of Computer Science*" Turner focuses on the genesis of a computer program as a technical artifact (Turner, 2018, p. 52). The following reconstructs Turner's approach to later show its applicability to music. From the trinity that is depicted as a process of specification, symbolic program and physical program the technical artifact that is a computer program.

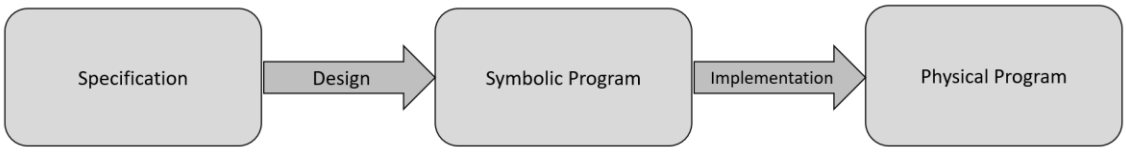

Figure 5. The Trinity of Specification Symbolic Program and Physical Program and their transitions. Figure configurated after (Turner, 2018, pp. 43-44).

While in Turner's view computer programs in general appear in a formal manner he adds the aspect of the contributing factors for their creation as parts of their existence. He uses GCD – the greatest common divisor of two numbers to illustrate his view (Turner, 2018, pp. 44-45). With the definition of the GCD, we already know *what* the output is supposed to be for any inputs, but we do not know *how* exactly the function may be realized (p. 45). Turner calls this the functional specification or rather the program specification (p. 44). This specification in the case of computer programs poses a normative character for the following steps in the emergence of the technical artifact (as in the symbolic program and the physical program) (p. 46). So these steps will be judged regarding their correctness according to this formal definition. If they correspond to the specification they will be called correct (pp. 45-46).

In the following design step the intended realization of the program will be determined, namely the way exactly *how* we are going to fulfill the specification. What

we will end up with after this step may then be called the symbolic program (Turner, 2018, pp. 46-47). When it comes to computer programs we can choose between different algorithms, as in what certain structure of assignments is going to produce the output we wish for (Turner, 2018, pp. 46-47). They may differ in their approach, goal-directedness and overall efficiency (some algorithms work more efficient for different types of inputs, i.e. different sorting algorithms for different lengths of lists) and still produce the same output. The then chosen algorithm needs to be compared to the specification in order to determine their correctness (pp. 46-47, pp. 205-206). Turner therefore sees algorithms as part of the second step, and not part of the specification itself (pp. 47). Turner then calls this structural description, which aims to be as mechanical as possible (in the best case already in a programming language) the symbolic program. This describes the certain way in which the input is permutated and the output is computed (pp. 47-49). While the specification can, at least in parts, be described in a natural language, the symbolic program may not (pp. 44-49). Algorithms for Turner are therefore the dedicated structures of assignments that are realizing the specification and constituting the symbolic program.

In the implementation step the symbolic program is implemented into a physical process, this physical realization is therefore called the physical program. While the structural description of the symbolic program does not inhibit all of the physical details of the resulting technical artifact, the physical program, realized by running the program on a certain piece of hardware eventually sets them via producing these physical details (Turner, 2018, p. 49-50). The specific way the program is run on a certain device takes part in shaping aspects of the artifact's properties. Turner uses the example of a program in form of a punch card to illustrate this. Only in combination with the interpreting device (Turner calls it the "*underlying mechanism*") of the certain punch card we designed, the artifact becomes what it is (Turner, 2018, pp. 49-50).

In summary we can say that Turner presents us a trinity that the technical artifact passes the course of its emergence. He makes the certain stages where the artifact's existence might still be considered vague graspable and enables us to analyze these. We can decide if we are talking about a specification of a technical artifact, the *what* that is aimed to be realized, the symbolic program, the structural description telling us *how* exactly it is realized and the physical program, the eventual manifestation of that specific artifact.

Turner also stresses the fact, that all technical artifacts, however abstract, may be resolved into this trinity metaphor of his (Turner, 2020). He elaborates that a technical artifact that is the construction of a car can also be dissected into these parts:

> The manufacturing process for cars is itself an artifact. The structural description of the process does not describe cars; it is a structural description of a process for building them. The inputs to the manufacturing process are car plans. When executed, the process outputs cars. The structural description of the implementation process for programs also describes a process. But its output are not objects but processes. The input to the implementation process are process plans (programs) and the output, when executed, are running processes (Turner. 2020, p. 362-363).

We have seen how broad Turner's approach of a technical artifact is and therefore how various constructs may be seen as technical artifacts. This also holds for musical compositions, which we want to investigate in the following.

Application of Turner's Theory to Pieces of Music

This understanding of the genesis of a technical artifact can be applied also to tonal musical compositions. If we take a certain piece of music that we are creating with a certain intention, perhaps to illustrate a character's feelings in a movie scene, we can go through the steps described by Turner. The composer is intending to write a piece of music that realizes the idea of the scene. Understanding how the character would feel and imagining what the music is to transport through the intended piece of music would make for the specification. This step does not yet include a certain way of how to later arrange the notes, it only circumscribes the intention. And we will have to point out that the specification of a piece of music may not be able to be written down as formal and unequivocally as the specification of a computer program.

In the design phase the composer would then focus on using their musical skills to compose the notes in a fitting way. Knowing the way notes are combined to exhibit certain patterns and themes, they are able to compose a certain arrangement of notes that will fulfill the previously set specification. There will be endless possibilities of how to realize this Specification. This is analogous to the endlessly possible algorithms one can use to satisfy a certain in- and relations. Multiple melodies and patterns of sound as well as different lengths, volumes, et cetera can satisfy the specification. Hence the analogy holds for pieces of music that are created with a certain intention.

Still we have to mention one restriction that comes with the composition of music. When it comes to music the effect it shows on people may be considered highly subjective. While we can mathematically prove that a certain algorithm meets a specification, this is not easily done for pieces of music. Some might say it is impossible per definition to do so. As we have stated earlier it is at least debatable whether this is a decidable matter. Since music does not exhibit conceptuality in the same way as language does (and is not prone to exhibit a direct correspondence with formal concepts or even certain emotions), this decision is finally up to the audience and its interpretation, as we have already pointed out in the previous section. The verification of the fulfillment of the set specification lacks formal principles that would enable us to make a clear and provable decision.

Having gone through the design phase we now have a symbolic program. For a computer program this means the lines of code are set, whereas for a piece of music this means that the individual notes are set in a chosen sequence and pattern. The next step is the implementation of the symbolic program that will yield the eventual physical process of the technical artifact. Taking our arrangements of notes, they can be implemented on a fitting device such as a (certain) musical instrument. Playing the arranged notes (the symbolic program) on this device will make for the implementation. It takes a learned skill to be able to correctly translate the notes into a way of playing the instrument. Here it is possible to decide whether this is carried out correctly. Is the beat on time? Is the piece played with the correct notes at the correct volume? Correct here means

corresponding to the symbolic program or musical score that was adopted in the previous step. In the thereby realized physical process the piece is finally brought into its desired manifestation. The device, here the specific instruments also influence the way of the realization of the artifact. The way the specific piano functions and sounds affects the piece of music that is performed. This part of the process works truly in analogy to Turner's elaborations.

In summary we can therefore maintain that Turner's description of technical artifacts can also be applied to musical compositions. We have found a core difference when it comes to the conceptuality of music and therefore a constraint on proving the correlation of the certain arrangement of notes with the set Specification. It is still up to us what to make of this difficulty. We could use empirical proof instead of mathematical proof where we accept that in case the majority of the audience agrees with the arrangement of notes we may consider them fitting. Just as we do when testing computer programs. The remaining aspects of Turner's theory were shown to be equally applicable to music.

The disclaimer might be added, that one need not consider all music to be composed in this exact way. Restrictions in the creation of art and music is not the aim of this investigation, only to show that a comparison *is* possible if we think of their creation from this perspective. It was done to show how we can perceive and investigate pieces of music as technical artifacts according to Turner's approach.

CONCLUSION

The aim of this paper was to illuminate the striking similarities between music and computer programs when we investigate them as technical artifacts. Especially through the arguments of Appelqvist, Horton und Turner we tried to underline these parallels. Nordmann shows us that the borders between technology and art are in fact drawn more firmly and arbitrarily in our everyday understanding than is called for by the facts.

Music as well as computer programs may be written on a sheet of paper manually, be it as notes on a score sheet or a list of assignments in a formal language. Whoever is able to read these, and therefore has learned the skill to grasp their content, is enabled by the given notes to understand the structure of the intended technical artifact that is contained in this structure. If implemented into the required device, be it a piano or a computer, they can play the program or the music and bring the artifact into its desired form as a physical process; through the sound waves leaving the body of the piano as well as through the physical running of the code on the computation device the artifact manifests itself.

Of course nonetheless some characteristic differences remain. When it comes to evaluating how 'good' they are, we do judge computer programs based on their efficiency with resources such as time and other costs. For music this is generally not the case. Also determining whether a piece of music meets its desired specification is an even more complex task than it already is for computer programs.

Though music and computer programs are mostly perceived as contrary, as contrary, they might be very similar in their ontology. Both can be viewed as technical

artifacts when it comes to their process of creation or genesis, and both become manifest and fully defined only through being played or run on a certain instrument or device. The ontological investigation of one may therefore contribute also to the ontological investigation of the other.

OUTLOOK

There is another striking similarity of the two that we have not considered as yet. Both, musical compositions and computer programs can keep their identity even through change. This is a remarkable point made by Nurbay Irmak (2012). A certain piece of code that inhabits a bug does not prevent us from seeing a program as still the same video game. And a certain piece of music that may have a note or a fraction missing may still be identified as the same song. The computer program, as in the video game might go through a bug-fix and realize a certain function by way of another algorithm than before (Irmak, 2012). A piece of music might be played with another yet accompanying line than originally and still we would call it the same piece of music. Computer programs might be emulated on another device and a piece of music could be played by a different instrument than originally set. The keeping of their original identity will remain only up to a certain point of course, beyond which they will assume a new identity. These common properties are what motivate further investigation. Especially the question of identity holds another fascinating challenge.

In this article we have approached pieces of music and computer programs as technical artifacts and shown the productivity of such a perspective. Turner's theory enables us to structure and categorize the steps of vague existence before the manifestation of the fully described artifact. Horton and Appelqvist make possible the use of the quasi-linguistic features of music for this comparison. Thanks to this approach tonal music and computer programs benefit each from ontological research of the other.

REFERENCES

Appelqvist, H. (2013). Wittgenstein and the Limits of Musical Grammar. *British Journal of Aesthetics, 53*(3), 299–319. https://doi.org/10.1093/aesthj/ayt017

Boghossian, P. (1989). The Rule-following Considerations. *Mind, 98*(392), 507-549.

Horton, T. (2001). The Compositionality of Tonal Structures: A Generative Approach to the Notion of Musical Meaning. *Musicae Scientiae, 5.2*(2001), 131–159.

Irmak, N. (2012). Software is an Abstract Artifact. *Grazer Philosophische Studien, 86*(1), 55-72. https://doi.org/10.1163/9789401209182_005

Nordmann, A. (2021). Die schöne Technik der Verschwendung. Größte Kleinigkeiten [The fine Technique of Squandering. Momentous Trifles.]. In S. Maasen & D. Atwood (Eds.), *Immanente Religion–Transzendente Technologie: Technologiediskurse und gesellschaftliche Grenzüberschreitungen*, 269–285. Verlag Barbara Budrich. http://www.jstor.org/stable/j.ctv25c4zbv.14

Rapaport, W. J. (2020). Syntax, Semantics, and Computer Programs. *Philosophy & Technology 33*(2), 309–321. https://doi.org/10.1007/s13347-019-00365-8

Turner, R. (2018). *Computational Artifacts. Towards a Philosophy of Computer Science*. Springer.

Turner, R. (2020). Computational Artifacts: the Things of Computer Science. *Philosophy & Technology*, *33*(2), 357–367. https://doi.org/10.1007/s13347-019-00369-4

Voevodsky, V. (2010). Univalent Foundations Project. NSF Grant Application. https://www.math.ias.edu/vladimir/sites/math.ias.edu.vladimir/files/univalent_foundations_project.pdf

Wittgenstein, L. (1922). *Tractatus Logico-Philosophicus*. International Library of Psychology Philosophy and Scientific Method 8. Edinburgh Press.

Wittgenstein, L. (1961). *Notebooks 1914-1916*. (G.H. Wright, and G.E.M. Anscombe, Eds.; G.E.M. Anscombe, Trans.). Basil Blackwell Oxford.

Wittgenstein, L. (1967). *Lectures & Conversations on Aesthetics*, Psychology and Religious Belief. University of California Press Berkeley and Los Angeles.

СВЕДЕНИЯ ОБ АВТОРЕ / THE AUTHOR

Лиза Марианна Борхерт
Lisa.borchert@googlemail.com
ORCID 0009-0005-1387-4385

Lisa Marianne Borchert
Lisa.borchert@googlemail.com
ORCID 0009-0005-1387-4385

Статья поступила 14 декабря 2022
одобрена после рецензирования 22 марта 2023
принята к публикации 25 марта 2023

Received: 14 December 2022
Revised: 20 March 2023
Accepted: 22 March 2023