Computational Creativity and Live Algorithms

Geraint A. Wiggins and Jamie Forth
Computational Creativity Lab
Queen Mary University of London
Mile End Road, London, E1 4FZ

Abstract

This chapter examines the field of algorithmic composition from the perspective of computational creativity. It begins by introducing the idea of computational creativity as a philosophical perspective. Next, it introduces a method for consideration of the properties of creative systems, the Creative Systems Framework (CSF; Wiggins, 2006a,b). The CSF becomes the starting point for a discussion of a system of comparison specific to algorithmic composition as an artistic and technical practice. Finally, the chapter sketches a road map for future developments in algorithmic composition and live coding, in these terms.

Keywords: computational creativity, algorithmic composition, creative systems frame-work, live coding

1 Introduction

Live algorithms have been present in Western music since as early as the eighteenth century. *Der allerzeit fertige Menuetten- und Polonaisencomponist* (Kirnberger, 1757) allows minuets and polonaises to be generated by choosing random numbers. While probably not necessarily intended for live operation, as opposed to prepared performance, the music is certainly performable in this way.

In the twentieth century, algorithmic processes in music became a feature of modernist composition, with composers such as Philip Glass (Potter, 2000) and John Cage (Revill, 1993) specifying processes in advance of performance and writing their output down. Live algorithmic music was less common, because of practical limitations, but there are examples, such as the chance-driven processes in Lutosłavski's *Jeux vénitiens* (Lutosławski, 1961). Perhaps the archetype of human-driven live algorithmic music is Terry Riley's *In C* (Riley, 1964; Potter, 2000), in which the performers, guided by a conductor, choose the transitions in a pre-specified algorithmic sequence.

However, to the best of our knowledge, there has yet to be a musically successful attempt to generate music live for humans to play, despite technologically interesting prototypes at specialist conferences, which, for example, generate instrumental scores live (e.g., Eigenfeldt et al., 2012; Eigenfeldt, 2014). For this reason, we base our argument on the restricted case where a human is programming a computer, live, to play sounds, which are specified by programmatic means: live coding. Live coding is a very specific microcosm of the broader live algorithms field, and its specificity helps make our model clear. However, we believe that the model we develop in this chapter is equally applicable to musical coding which happens not to be live.

Given that a computer is involved, a natural question to ask is "how involved is the computer?" In the majority of cases, we believe that the computer serves as a very powerful sequencer, where the specification of the sequence is given in *intensional* terms[1] (that is to say, specified as a generative process) rather than extensional terms (that is to say, specified as a set of notes, or by the positions of a sequence of knobs or

[1]These terms are borrowed from symbolic logic. An *intensional specification* is one which is couched in terms of properties, such as 'the integers between 0 and 100', while an extensional specification lists the set of things referred to. Clearly, intensional specifications can specify infinite things (e.g. 'all numbers greater than 0', where extensional ones cannot. Further, and most important to our purposes, an intensional specification can be a program that generates things.

switches on an analog synthesiser). As such, while the scope of such expression is clearly substantially broader, and the means of expression fundamentally different, the essential nature of the activity is not different from the complex-sequencer-based work of bands such as Tangerine Dream, in the 1970s[2]. The nature of the intensional specification of sequence is very clearly exemplified in languages such as Tidal (McLean, 2011), which are optimised from the perspective of easily, efficiently and intensionally specifying operations, live, that map between sequences specified extensionally or intensionally. This difference is crucial to the purposes of this chapter, not so much because of the breadth of expression afforded, but because programs and the numbers that drive them are capable of representing information at more than one level simultaneously. In particular, they are able to represent and reason about themselves, as well as about their outputs, affording the capacity for *reflection* (reasoning about one's own behaviour), which is not available to a hardware sequencer, whose knobs are (literally) hard-wired to whichever functions they control, and whose clock is just that, voltage control notwithstanding. Reflection is a key feature of creative autonomy, and our purpose here is to explore future paths for live coding, in which the computer is given more *creative responsibility* (Colton and Wiggins, 2012) for the outputs produced than is the case at present.

McLean and Wiggins (2010c) elicited opinions from practising live coders as to the current and future development of automation in live coding, particularly in respect of creative autonomy of the computer. Of those who responded, 40.7 percent believed that it was possible, at the time of the survey, for live code to modify itself in an artistically valued manner, and some of those who disagreed were optimistic that this would be possible in future. Exactly half of the respondents agreed that a computer agent has been developed that has produced a live coding performance indistinguishable from that of a live coder, or that one such will be developed within five years of the survey. Of the same cohort, however, 34.6 percent believed that such an agent will never be developed.

The aim of the current chapter, therefore, is to begin to lay out the path towards such valued creativity in a live coding agent. We begin by defining the Creative Systems Framework (Wiggins, 2006a,b) which will provide the context for our discussion, and illustrating its application with a very simple example concerning an imaginary live coder, and we very briefly introduce Tidal, our live coding language of choice. We then proceed to examine the consequences of following through the various possibilities to foresee a live coding system that might work in creative partnership with a human in a true hybrid creative system.

2 Creative Systems

The Creative Systems Framework (CSF; Wiggins, 2006a,b) takes as a starting point the following definition of a *creative system*.

Creative system A collection of processes, natural or automatic, which are capable of achieving or simulating behaviour which in humans would be deemed creative. (Wiggins, 2006a, p. 451)

This definition presupposes, not unreasonably, that creativity is best understood in terms of human behaviour, in that we can meaningfully discuss non-human creativity only with reference to behaviour exhibited by humans. However, depending on the vantage point one takes when considering a creative process, alternative conceptualisations of creative systems can emerge. For example, an improvisation context, comprising human and/or artificial agents, may be considered a creative system when viewed from a certain level of abstraction, as a "black box". Likewise, the abstraction boundary may be increased still further, resulting in creative behaviour at the level of societal dynamics, or lowered into hypothesised mechanisms underlying individual human creativity or more general cognition (Baars, 1988; Wiggins and Forth, 2015). Applying the CSF at various levels of abstraction it becomes possible to separate out the contribution of many disparate elements that together give rise to complex and emergent creative behaviour.

In the practice of live coding, computational systems are predominately viewed as tools or instruments under the control of the human performer, and thus as means of expressing human creativity. Some live coders tend to view systems more as collaborators, particularly when the systems exhibit behaviour that is complex and challenging (Bovermann and Griffiths, 2014). In this case it appears that sense of agency on

[2]Two ground-breaking examples of "live in studio" manipulation of sequencers by Peter Baumann can be found in the Tangerine Dream tracks *Phaedra* (1974) and *Stratosfear* (1976) from the albums of the same names.

behalf of the system becomes established by the perception of the system's behaviour in the mind(s) of the performer and/or audience. Taking the live coder together with the system as a basic level of abstraction for applying the CSF, we are able to identify where principle boundaries of responsibility lie for sustaining creative behaviour in the partnership of human and algorithmic processes. Clarifying these distinctions will enable the potential shifting of creative responsibilities and for artistic motivations leading to more inspiring interactive live coding partnerships, but also for motivations in the scientific study of creativity.

3 The Many Levels of Creativity in Live Coding

Creativity abounds, on multiple levels, within live coding. The software employed, or at least the core set of abstractions used to express musical concepts, are typically developed by the live coder as an integral part of the development of their musical aesthetic. Before a live coder takes to the stage, a long series of artistic and technical challenges have to be addressed, requiring a high degree of ingenuity and technical competence. In the cycles of software development or dedicated rehearsal sessions, the live coder will experiment and become more familiar with the system's idiosyncrasies and explore the potential scope of musical output. In a manner akin to the extended-mind theory of consciousness (Clark and Chalmers, 1998), the live coder becomes attune to thinking with and through the medium of code and musical abstractions, such that the software can be understood as becoming part of the live coders' cognition and creativity. Such fundamental engagement with musical structure through the medium of code leads to an musical aesthetic suffused with algorithmic elegance. This phenomenon is not restricted to live code practitioners: Magnusson (2014, p. 13) identifies from an extensive survey of live coding practice, that a common motivation amongst performers is to 'communicate algorithmic thinking'. More generally, Collins (2008, p. 240) characterises the role of the music analyst when considering computer-generated music as being to seek 'to explain a given output (a production) in terms of the originating program (a source)'. Given the prominence of code projection and other forms of algorithmic visualisation during live coding performance—enabling audiences to form, at least to some degree, an appreciation of the music with reference to the processes by which it is being generated—it is reasonable to assume that means of generation are integral to the aesthetic values of live coded musical performance.

Beyond the coupling of live coder and computational system, creative behaviour can be observed in group performance. Group live coding performances typically follow a model borrowed from improvised jazz where performers interact with fellow performers in an ongoing negotiation of musical development (McLean, 2014). Creativity here can be viewed as distributed among the participating creative agents. Audience members, simply by engaging with the performance can be understood as exhibiting creative behaviour by at the very least making meaning out of the experience, but also potentially influencing the direction of the performance by means of what McLean (2014) identifies as the inherent social feedback involved in live code performance.

4 Formalising Creative Systems

To formalise the idea of a creative system, we first introduce Boden's abstract model of creativity, and show how it can be formalised, to provide a tool set for discussing creative systems. Of course, the creative systems we have in mind here are hybrid ones, composed of a human programmer-composer and an algorithm. In particular, in this section, we introduce some specific properties of creative systems that will be useful in our taxonomy.

4.1 Boden's Model of Creativity

Boden's (2004) model of creativity revolves around the notion of a *conceptual space* and its exploration by creative agents. The conceptual space is a set of artefacts (in Boden's terms, *concepts*) which are in some quasi-syntactic sense deemed to be acceptable as examples of whatever is being created (Boden, 2004). Implicitly, the conceptual space may include partially defined artefacts too. *Exploratory creativity* is the process of exploring a given conceptual space; *transformational creativity* is the process of changing the rules which delimit the conceptual space. Boden (1998) also makes an important distinction between mere membership of a conceptual space and the *value* of a member of the space, which is defined extrinsically, but not precisely.

Bundy (1994) and Buchanan (2001) join Boden in citing reflection, and hence meta-level reasoning, as a requirement for "real" or "significant" creativity (though the definition of such creativity is so far left imprecise). Again, it is the capacity to reflect that we consider central here.

For completeness, we mention here that there are other views. Ritchie (2007), for example, presents a completely different account of what is going on in "transformational" creativity, in which the notion of transformation is not so clearly present. Colton et al. (2014, 2015) present the IDEA and FACE models, that attempt to characterise creativity from different perspectives. However, since the current chapter is primarily focused on the application of Boden's theory to live coding, via our Creative Systems Framework, explained next, we defer discussion of alternative approaches.

4.2 The Creative Systems Framework

The central idea of the Creative Systems Framework (CSF), the formalism presented by Wiggins (2006a), is that an exploratory creative system in Boden's (2004) terms, may be abstractly represented by a septuple, thus:

$$\langle \mathcal{U}, \mathcal{L}, [\![.]\!], \langle\!\langle ., ., . \rangle\!\rangle, \mathcal{R}, \mathcal{T}, \mathcal{E} \rangle.$$

The symbols here are defined in Table 1. The function of each is briefly explained below (Wiggins, 2006a, gives more detail)[3]

Table 1: The symbols used in Wiggins' description of Boden's exploratory-creative system.

$\mathcal{U}$	a universe of possible concepts (artefacts), both partial and complete
$\mathcal{L}$	a language in which to express concepts (artefacts) and rules
$[\![.]\!]$	a function generator, which maps a subset of $\mathcal{L}$ to a function which associates elements of $\mathcal{U}$ with a real number in [0,1]
$\langle\!\langle ., ., . \rangle\!\rangle$	a function generator, which maps three subsets of $\mathcal{L}$ to a function that generates a new sequence of elements of $\mathcal{U}$ from an existing one
$\mathcal{R}$	a subset of $\mathcal{L}$
$\mathcal{T}$	a subset of $\mathcal{L}$
$\mathcal{E}$	a subset of $\mathcal{L}$

$\mathcal{U}$ is the (abstract) set of all possible partial and complete artefacts describable in the creative system being modelled. $\mathcal{R}$ is a set of rules, expressed using the language $\mathcal{L}$, which select an "acceptable" or "relevant" subset of $\mathcal{U}$ which corresponds to Boden's (2004) conceptual space. In Wiggins' formulation, selection is *permissive* in the sense that it admits partial artefacts, even some of whose completions may eventually turn out not to be admitted. So applying a selector function generated from $\mathcal{R}$ by $[\![.]\!]$ and a suitable real comparator (*e.g.,* 0.5) gives Wiggins' formalisation of Boden's conceptual space:

$$\{c \mid c \in \mathcal{U} \wedge [\![\mathcal{R}]\!](c) \geq 0.5\}.$$

The ruleset, $\mathcal{R}$, then, defines what it is to be an artefact of the kind we are interested in creating: a piece of music, a joke, and so on. (Alternatively, the output of $[\![\mathcal{R}]\!]$ might be used directly in a fuzzy selector; we postpone discussion of this for now.)

$\mathcal{T}$ is a set of rules which, when interpreted, perhaps along with those in $\mathcal{R}$ and $\mathcal{E}$, by $\langle\!\langle ., ., . \rangle\!\rangle$, describe the behaviour of a creative agent as it traverses the conceptual space from known artefacts to unknown ones (much as the standard AI search framework; Wiggins, 2006b, explains the relationship in detail), and possibly back again. The first argument of $\langle\!\langle ., ., . \rangle\!\rangle$ takes a concept/artefact-definition ruleset, such as

[3]Ritchie (2012) presents a slightly different formalisation of broadly the same ideas.

$\mathcal{R}$, above, and the second a rule set such as $\mathcal{T}$, which is the specification of the traversal strategy. The third argument is $\mathcal{E}$, the rules by which *value* is attributed to a created artefact, new or otherwise (see below). $\mathcal{R}$ and $\mathcal{T}$ are included so that it is possible for $\mathcal{T}$ to include reasoning about them, but this is not a requirement; thus, $\mathcal{T}$ can in principle generate artefacts which do not conform to the rules of $\mathcal{R}$ and this can be used to trigger subsequent reasoning and reflection about the creative system under simulation (Wiggins, 2006b). There is no explicit equivalent of $\mathcal{T}$ in Boden's writing, though it is implicitly present at all times. To distinguish between transformation of $\mathcal{R}$ and transformation of $\mathcal{T}$ we write "$\mathcal{R}$-transformation" and "$\mathcal{T}$-transformation".

$\mathcal{E}$ is a set of rules which define the evaluation of the creative outputs resulting from the agent's activity, appropriately contextualised. The formalism does not specify what this context is; it might be the subjective judgement of the creating agent, or the subjective combined judgements of other agents, or comparison with some objective measure. $\mathcal{E}$ allows us to express the notion of value proposed by Boden (1998). (For completeness, we mention that we would expect $\mathcal{E}$ to be amenable to transformation, also, in particular ways, especially if this theory were applied in the context of a multi-agent system. However, for the moment we leave the interesting question of how usefully to formalise $\mathcal{E}$-transformation to future work.)

A further useful mechanism is the function $^\diamond$, defined such that

$$\mathcal{F}^\diamond(X) = \bigcup_{n=0}^{\infty} \mathcal{F}^n(X),$$

where $\mathcal{F}$ is a set-valued function of sets; this allows generation of all the concepts derivable under $\mathcal{T}$ from a given starting concept: below, we will substitute $\langle\langle ., ., , \rangle\rangle$ for $\mathcal{F}$ in this formula to capture iteration across the whole search space. A useful constant will be $\top$, the null (or completely undefined) concept, which inhabits all conceptual spaces.

A brief example may help to clarify the usage of this mechanism. Consider the familiar task (Ebcioğlu, 1988, for example) of harmonising of seventeenth century German hymn tunes in the style of J. S. Bach. We can model this case as follows (but note that there are other ways, depending on what one wants to achieve). $[\![\mathcal{R}]\!]$ selects a subset of $\mathcal{U}$ which might be described as the set of all partial and complete harmonisations of the canon in question. $\mathcal{E}$ then selects those which are considered good, according to criteria that may be related to appropriate rules of music theory, psychological models of music perception, and/or socially inspired metrics designed to quantify aspects of value in relation to existent harmonisations. To see why there is a difference between $\mathcal{R}$ and $\mathcal{E}$, consider the comparison between the harmonisations produced by J. S. Bach himself, and those produced by a first-year music student: the latter are not usually valued as highly as those of the former, because even the best student is unlikely to produce music of the same quality as those Bach harmonisations which have been selected by several hundred years of history.

This same pair of subjects can help understand the need for $\mathcal{T}$, also. An extremely competent and experienced composer and improviser such as Bach will normally have the ability to "see" a harmonisation which is correct in syntactic terms and of high quality in value terms without too much conscious effort. This is rarely true of beginning composers, who need to develop their intuitions over a period of time, usually through a kind of problem-solving approach. $\mathcal{T}$ allows us to model these behaviours individually, and to study their interactions with the externally defined $\mathcal{R}$ and $\mathcal{E}$. Also, crucially for the evaluation of artificial creative systems, the process by which a system produces new artefacts, as defined by $\mathcal{T}$, is integral in determining the extent to which behaviour may be deemed creative. For example, brute-force search, or a very prescriptive approach based on hand-coded rules, is unlikely to be considered creative, especially compared to a process containing a set of learned, higher-level abstractions enabling the generation of highly valued artefacts with a high degree of efficiency.

Wiggins (2006b) gives examples to elucidate how the framework may be used, and shows how *transformational* creativity can be cast as exploratory creativity at the meta-level, where the conceptual space is the set of possible rule sets, generated by a given language, as informally suggested by Bundy (1994).

A substantive difference between Boden's formulation and that of Wiggins is the addition of the rule set, $\mathcal{T}$, which describes the actual behaviour of a creative agent as it goes about its business: Boden is not concerned with this level of detail. The difference gives Wiggins' formulation more power to describe the behaviour of *implemented* creative systems. Thus, it may be compared in detail with existing similar methods, such as those of AI state space search. Further, the introduction of $\mathcal{T}$, as an explicit component, admits a new kind of transformational creativity, in which an agent modifies its own behaviour by reflective reasoning. This may be appropriate for the description of behaviours exhibited by Lenat's AM, for example (Ritchie and Hanna, 1984).

4.3 Useful Properties of Creative Agents

The apparent supposition in Boden's work is that creative agents will be well-behaved, in the sense that they will either stick within their conceptual space, or alter it politely and deliberately by transformation. It can be argued, however, that this is not adequate to describe the behaviour of real creative systems, natural or artificial, either in isolation or in societal context. This section identifies some situations not covered by the assumption of good behaviour, and gives names to them. The important point is that some of these situations may appropriately trigger particular events, such as a step of transformational creativity, so it is useful to be able to identify them in the abstract. This leaves us with several general classes of small-scale conditions which might be observed in AI systems, of which we can then assess the creative potential.

These characterisations are only descriptively useful unless appropriate responses, categorised by condition, can be specified. This section does so. I assume some appropriate learning mechanism(s) which can adapt the rules (expressed in language $\mathcal{L}$ and categorised into $\mathcal{R}$, $\mathcal{T}$ and $\mathcal{E}$), from positive and/or negative training sets.

4.3.1 Application to Live Coding

For the purposes of this chapter, it is useful first to calibrate the application of the system, by assigning meanings to the various symbols in the formalism. To illustrate the use of the framework in as transparent a way as possible, we omit the more complex questions, such as interaction between human performers and any aspect of performance by our notional programmer which is not mediated via the live coding system.

First, the universe, $\mathcal{U}$, in our case, is all possible music that could potentially be produced (under any definition of "music" with respect to a given representation[4]), whether or not by our example live coding practitioner. At the most abstract level, the conceptual space, $\mathcal{C}$, specified by the rule set, $\mathcal{R}$, is the range of live coded music that our practitioner can imagine (which is therefore in all probability a subset of $\mathcal{U}$). $\mathcal{T}$, the transition rules, specify a combination of her craft as a live coder and the music that can be produced by the algorithms that she writes. $\mathcal{E}$, the evaluation rules, express her preferences in the outcomes of this process, and may refer to the quality of the code, or to the music, or both.

It becomes immediately clear that one could more precisely conceptualise this hybrid creative system, in which a human creates a program, which then creates for itself, as two distinct layers within the CSF. There would be two universes, one of live algorithms, and one of music, with a mapping between them, corresponding to the execution rule of the relevant programming language. Thus, we express our performer's creativity in programming, and in music, distinctly. Doing so would allow us to consider programming techniques, and the design of specialist languages for live coding (McLean and Wiggins, 2010b,a), and this is our aim later in the chapter. For now, however, to do so would over-complicate our example. Therefore, $\mathcal{T}$, in our first example, corresponds to the ability of the *code produced* to traverse $\mathcal{C}$, and *not* with the ability of the programmer to write it. Similarly, our evaluation function, $\mathcal{E}$, corresponds to musical value, and not to value judgements concerning the elegance of code, or other such matters of programming. What is more, we focus $\mathcal{E}$ specifically on musical value attributed by our practitioner, and not on that endowed by the approval of an audience, for example. This will come later.

Here and elsewhere, the sonic entity being evaluated may be any of a range of musical structures at various scales, depending on the code being used, and the focus of attention by the listener. We do not make these distinctions in our examples, because they do not add to our discussion: the reader may choose any or all of the possible facets of the generated music as his or her preferred area of interest.

[4]We will not explore issues of representation at this point, suffice it to say that music, fundamentally a psychological phenomenon, may be represented from multiple perspectives and at various levels of abstraction, such as digital audio signals, score-like discrete representations or in terms of psychological models of musical perception (Babbitt, 1965; Wiggins, 2012b,c,a).

4.3.2 Uninspiration

There are various ways that a creative system can fail to be creative in a valued way. These ways can be characterised through the rule set $\mathcal{E}$ and its relationship with the other components of the CSF.

Hopeless uninspiration is the simplest case, where there are no valued concepts in the universe:

$$[\![\mathcal{E}]\!](\mathcal{U}) = \emptyset.$$

This system is incapable, by definition, of creating valued concepts, and as such might be termed ill-formed (if such creative behaviour is the intention).

In this case, there is no solution within the specified universe; there is no capacity within the system to solve the problem. Therefore, it is up to the system designer to remedy the problem, like a *deus ex machina*.

For the purposes of our example, we suppose that this case does not arise. It corresponds to the situation where no valued music exists. (With a more specific application of the framework, however, hopeless uninspiration is possible: if we were to take as our universe all live algorithms music, we cannot necessarily assume that $\mathcal{E}$ will accept any members of $\mathcal{U}$.)

Conceptual uninspiration arises when there are no valued concepts in the conceptual space:

$$[\![\mathcal{E}]\!]([\![\mathcal{R}]\!](\mathcal{U})) = \emptyset.$$

We label this form of uninspiration "conceptual" because it entails a mismatch between $\mathcal{R}$ (which defines the conceptual space) and $\mathcal{E}$ (which evaluates concepts within it, and, more broadly, within $\mathcal{U}$). This condition is contradictory to the purpose of the two rule sets: if $\mathcal{R}$ is supposed to constrain the domain of a creative process, then it is inappropriate for $\mathcal{E}$ not to select some of the elements it admits. As such, like the hopeless case, conceptual uninspiration indicates ill-formation of the intended-creative system.

Conceptual uninspiration can only be addressed, within the system, by the transformation of $\mathcal{R}$.

In our live coding example, this situation is where our programmer does not value the kind of music which she conceptualises. It is probably not, therefore, likely to be an interesting case.

Generative uninspiration occurs when the technique of the creative agent does not allow it to find valued concepts within the space constrained by $\mathcal{R}$:

$$[\![\mathcal{E}]\!](\langle\!\langle \mathcal{R}, \mathcal{T}, \mathcal{E} \rangle\!\rangle^{\diamond}(\{\bot\})) = \emptyset.$$

This kind of uninspiration is less serious than the other two, and does not necessarily indicate an ill-formed creative system: it merely indicates that a creative agent is looking in the wrong place. This raises the question of *why* there is such a mismatch. Boden's underlying assumption seems to be that the conceptual space is in some sense definitive, and, certaintly, in a multi-agent environment, it is the only place in the formalism where the consensus about a creative domain can logically be represented. Generative uninspiration can be remedied within the framework. Transformational creativity is required. To transform the set $\mathcal{T}$ in a useful way, we need to identify one or more valued concept(s), in the conceptual space constrained by $\mathcal{R}$ (otherwise, we may have aberration, discussed below), and to use it (them) to guide the transformation. However, there is a methodological problem here: there is no clear way to pick the concept(s) automatically, except at random or by use of an oracle. The "oracle" might in fact be systematic search of $\mathcal{R}$ (assuming this is possible in finite time), or, again, the *deus ex machina* of user intervention.

In the live coding context, this situation corresponds to a programmer who has not written an algorithm that generates music that she values. She must transform her algorithm so that it can do so.

4.3.3 Aberration

Now, consider the following more interesting set of scenarios, which also concerns the relationship between $\mathcal{R}$ and $\mathcal{T}$. A creative agent, **A**, is traversing its conceptual space. From any (partial) concept(s) in the conceptual space, **A**'s technique will enable it to create another. Suppose now that the new concept does not conform to the constraints required for membership of the existing conceptual space (note that there is no guarantee that it should do so – there is only an assumption in Boden's work), and is therefore not selected by $[\![\mathcal{R}]\!](.)$. In this case, the set $\mathcal{A}$ given by

$$\mathcal{A} = \langle\!\langle \mathcal{R}, \mathcal{T}, \mathcal{E} \rangle\!\rangle^{\diamond}(\{\bot\}) \setminus [\![\mathcal{R}]\!](\mathcal{U})$$

is non-empty. The CSF terms this *aberration*, since it is a deviation from the notional norm as expressed by $\mathcal{R}$. The choice of this rather negative terminology is deliberate, reflecting the hostility with which changes to accepted styles are often met in the artistic world.

The evaluation of this set of concepts is actually slightly more complicated than the single-concept motivating case outlined above. The aberrant but valued subset, which I call $\mathcal{V}_{\mathcal{A}}$ here, is calculated thus:

$$\mathcal{V}_{\mathcal{A}} = [\![\mathcal{E}]\!](\mathcal{A}).$$

Because we are working in the extensional limit case, with all the created concepts notionally elaborated, we have to consider the possibility that all aberrant concepts, some aberrant concepts or no aberrant concepts may be valued. The CSF terms these *perfect* ($\mathcal{V}_{\mathcal{A}} = \mathcal{A}$), *productive* ($\mathcal{V}_{\mathcal{A}} \subset \mathcal{A}$) and *pointless* ($\mathcal{V}_{\mathcal{A}} = \emptyset$) aberration, respectively.

In the case of aberration, there is a choice as to whether to value the result or not, and therefore we have the three categories, perfect, productive, and pointless. Acceptability is determined in terms of evaluation by whatever audience the agent, **A**, is playing to—our live coder in this case. If a new concept is accepted, then a sensible solution might be to revise the notion of what the correct domain (as constrained by $\mathcal{R}$) is, so as to include the new concept. This, of course, might have consequences: other new concepts might be included and/or existing ones might be excluded along the way. If the new concept is not accepted under evaluation, then a reasonable recourse would be to adapt **A**'s technique, $\mathcal{T}$. This may have similar consequences with respect to added and existing concepts available to **A**: valued concepts may be lost, and new aberrant behaviour may be made possible.

One approach is to use the sets $\mathcal{A}$ and $\mathcal{V}_{\mathcal{A}}$ to generate training examples to modify $\mathcal{R}$ and $\mathcal{T}$, using our learning mechanism(s), as follows. Note that there are open questions here about some of the training sets required, since that choice is a major factor in the behaviour of the system. The main issue here is a standard one for AI: how much of what an AI program does is simply programming a computer directly to do something, and how much is emergent behaviour which was not directly programmed? In particular, if we first simply train $\mathcal{T}$ to match $\mathcal{R}$ we might be "coaching" our creative agent too directly, instead of allowing it to develop, and, second, in doing so we might be restricting its creative capability.

Perfect aberration yields new concepts, all of which are valued, and so should be added to $\mathcal{R}$. $\mathcal{T}$ has enlightened us as to new possibilities. We therefore attempt to revise $\mathcal{R}$, by whatever learning methods are available, in such a way that all the concepts in $\mathcal{A}$ (and $\mathcal{V}_{\mathcal{A}}$) are included, so $\mathcal{V}_{\mathcal{A}}$ is a positive training set, and the negative training set is either $\emptyset$ or $\mathcal{U} \setminus [\![\mathcal{R}]\!](\mathcal{U}) \setminus \mathcal{A}$ or some subset of the latter, depending on the effect desired.

In our running example, perfect aberration is the case where the programmer's algorithm generates unexpected music, all of which is valued. Obviously, on defining a hit in a way that she hadn't previously conceptualised, she will want to adapt her notion ($\mathcal{R}$) of what is live coded music.

Productive aberration means that we need to transform both $\mathcal{R}$ and $\mathcal{T}$, because we wish valued concepts to become accepted, and unvalued ones not to be generated. $\mathcal{V}_{\mathcal{A}}$ and $\mathcal{A} \setminus \mathcal{V}_{\mathcal{A}}$ constitute positive and negative training sets for $\mathcal{R}$, since $\mathcal{R}$ needs to expand just enough to include only the valued concepts in $\mathcal{A}$. $\mathcal{T}$, on the other hand, needs to be transformed to restrict its coverage: $\mathcal{A} \setminus \mathcal{V}_{\mathcal{A}}$ is a negative training set for $\mathcal{T}$, while, again, a positive training set might be $[\![\mathcal{R}]\!](\mathcal{U})$, or simply $\emptyset$.

For our example live coder, productive aberration is more difficult than perfect. It requires deeper introspection to identify which aspects of the aberrant music should be retained and which should be rejected. She will need to open her mind ($\mathcal{R}$) to the new concepts that she had not previously entertained, while adapting her algorithm so that it no longer produces the aberrant music that was not valued.

Pointless aberration suggests the need to transform $\mathcal{T}$ only, so as to prevent the unvalued aberrant concepts from being generated. There is a negative training set: $\mathcal{A}$. Again, the nature of the positive training set is an open question.

For our example programmer, pointless aberration is an indication of failure. She will need to rewrite her algorithm to preclude the unvalued musical concepts.

5 Tidal

The Tidal programming language (McLean, 2011; McLean and Wiggins, 2010b) is a real time, embedded domain specific language for live coding[5]. It consists of a conventional command line interface, which its inventor uses within the Emacs programmable editor, to enable easy reference and reuse of past commands. The language itself is implemented as an extension of the strongly-typed functional programming language, Haskell (Thompson, 2011). Functional languages are particularly well suited to this kind of task, partly because they are symbolic, making it very easy (for the live coder) to associate program fragments with easy-to-remember symbols (that the live coder has chosen); these program fragments, which may be simple constant values, or complex sound-generation routines, can then be composed into sequential structures, stacked into simultaneities, or both, and then operated on by high-order combinators, expressed directly in Tidal syntax. For example, one can construct a sequence of drum beats by writing down the names of the relevant sounds in sequence, then reverse it by the application of one simple combinator, and then execute performance of both simultaneously by the application of another.

Importantly from the perspective of live performance, Tidal is a live compiling language. Commands are implicitly looped, and whatever is playing currently continues until a new command has been successfully compiled. What is more, there is a notion of completion, which ensures that execution of a new command begins at a time which is musically appropriate, according to McLean's particular aesthetic. This, coupled with Haskell's very powerful type-checking system, helping the live coder to produce correct code, yields a highly expressive and flexible performance interface.

The final crucial ingredient is synchronised parallelism: Tidal is capable of running several commands at once, and implicit rules ensure that their output is synchronised, again in keeping with McLean's musical aesthetic.

Underlying Tidal is a scheduling system based on OSC (Wright and Freed, 1997), which means that, ultimately, anything that can be done in Tidal can be done in the reader's favourite generative composition system, given an OSC interface—but probably not as easily. This means that Tidal can form a conceptual framework for the rest of the current discussion, while not limiting its scope, because the modes of expression it affords are general.

6 Live Coding in the Creative Systems Framework

What, then, does the philosophy of computational creativity have to offer the hybrid creative system formed by a live coder and her Tidal performance system? We now consider the components of the hybrid system in terms of the CSF, generalising from our earlier illustrative example. First, we formalise the representations of the conceptual spaces and the relationship between them. Then we formalise the dynamics of the system. This allows us, finally, to identify where some of the creative responsibility in live coding performance might be shared with the computer.

6.1 Intentional and Extensional Representation of Knowledge

Our original universe, $\mathcal{U}$, of all possible musics must be expanded to include Tidal programs, as we now consider these explicitly. We introduce a conceptual space of well-formed Tidal programs, $C_{\mathcal{TP}}$. Since the execution rule of Tidal is deterministic[6], there is a many-to-one mapping from $C_{\mathcal{TP}}$ to the conceptual

[5]It grew out of the earlier Petrol language, but it is intended to be more sustainable. It may be downloaded from `https://tidalcycles.org/`.

[6]Assuming, as we do here, that randomness is not involved.

space of Tidal music, which we call $\mathcal{C}_{\mathcal{TM}}$. The mapping, which we call $\mathcal{X}$ (for "eXecution") is many-to-one because there is more than one way to express the production of some items of music, with no audible difference (e.g., two bars of four beats or four bars of two beats in a performance that does not emphasise metrical structure). In an intuitive sense, $\mathcal{C}_{\mathcal{TM}}$ gives semantics to $\mathcal{C}_{\mathcal{TP}}$, which potentially opens interesting questions about music similarity as a measure of program similarity, and which will enable part of our proposal, below. Note that these two conceptual spaces are objectively defined by the syntax and execution rule of Tidal. This is illustrated in Fig. 1a. We also introduce an inverse mapping, $\mathcal{X}'$, from points in $\mathcal{C}_{\mathcal{TM}}$ to sets of points in $\mathcal{C}_{\mathcal{TP}}$ such that $\mathcal{X}'(m_i) = \{p_i : \mathcal{X}(p_i) = m_i\}$. This partitions the conceptual space of Tidal programs into equivalence classes on the basis of identical musical output.[7]

Now we move on to the subjective part of the system: the Live Coder, whom we will call Elsie. For simplicity, we assume that Elsie will program without making audible errors—while this would be a big assumption in most programming languages, Tidal is specifically designed not to degrade on error, so it is not unreasonable here. Supposing that Elsie is only human, and therefore not perfect, it is reasonable to assume that her personal conceptual space of Tidal programs is a strict subset of $\mathcal{C}_{\mathcal{TP}}$. Equally, the likelihood is that her personal conceptual space of Tidal-produced music will be smaller than $\mathcal{C}_{\mathcal{TM}}$. It may also have elements in it that are *not* members of $\mathcal{C}_{\mathcal{TM}}$, because the coder's prediction of what her code will do may sometimes be incorrect. So we give ourselves the extensional sets $\mathcal{C}_{\mathcal{P}}$ and $\mathcal{C}_{\mathcal{M}}$, respectively, and the corresponding intensional rule sets $\mathcal{R}_{\mathcal{P}}$ and $\mathcal{R}_{\mathcal{M}}$, respectively, to express these points. The extensional nature of set $\mathcal{C}_{\mathcal{P}}$ should not be confused with the intensional nature of its constituent artefacts: Tidal programs are intensional representations of musical sequences, but within the CSF, component sets are considered extensionally. These are illustrated in Fig. 1b.

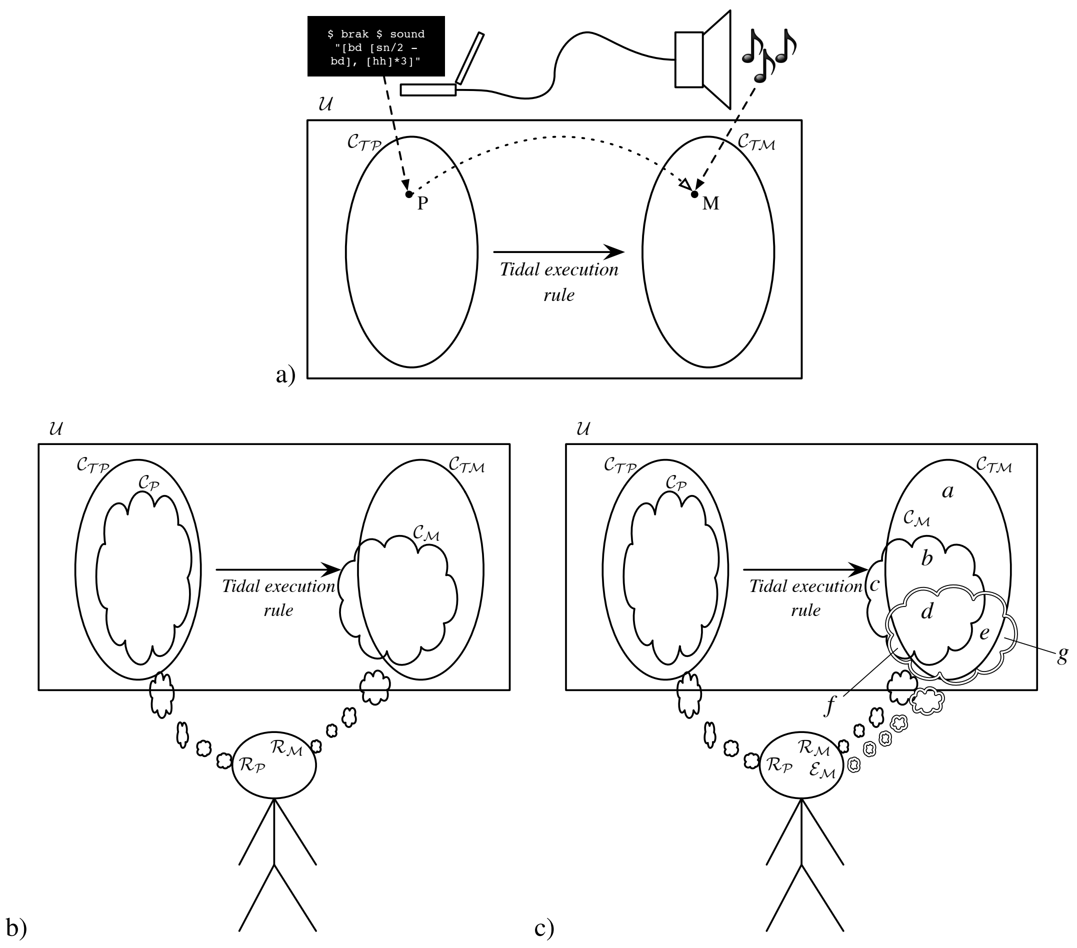
Because we are focused on a wider remit than just live coding in this chapter, we omit consideration of Elsie's aesthetic preference regarding coding style, because it complicates our model beyond what is necessary to convey our message. In an equivalent model specifically of live coding, this would be an indispensable component. Tidal is a very concise language, and therefore there is not very much range of expression in this sense. We therefore use the empty set, $\emptyset$, instead of the more predictable $\mathcal{E}_{\mathcal{P}}$.

The formalisation starts to become interesting when we add in Elsie's music-aesthetic preference, expressed as a rule set $\mathcal{E}_{\mathcal{M}}$, which selects a subset of $\mathcal{U}$, which may contain some or all of each of $\mathcal{C}_{\mathcal{TM}}$ and/or $\mathcal{C}_{\mathcal{M}}$. This gives us the arrangement illustrated in Fig. 1c. The different combinations of intersection and non-intersection between $\mathcal{C}_{\mathcal{TM}}$, $\mathcal{C}_{\mathcal{M}}$ and the extension of $\mathcal{E}_{\mathcal{M}}$, labelled with lower case letters in the diagram, indicate areas into which actual or imaginary pieces of music might fall, and each of them corresponds with a different possibility, from the perspective of computational creativity. We now consider each in turn, not in terms of the constructive process necessary to build a program, but in terms of the knowledge and/or imagination required to generate the computational and/or musical concept. The details are summarised in Table 2.

6.2 Representation of Dynamics of the Hybrid Creative System

Now we have mapped out the landscape of possible outcomes of our human-computer hybrid creative system, we must look at the dynamics. The Tidal techniques invisaged by McLean (2011) involve a somewhat incremental approach to programming, where one often constructs a basic musical structure extensionally (that is, in literal notes or sounds), and then elaborates on it by a mixture of added extensional structures (for example, a counter-rhythm to be played simultaneously; the approach lends itself well to strict additive process, such as that used in the early work of Philip Glass: Potter, 2000), or intensionally, by applying Tidal functions that manipulate the material as part of the performance. This approach lends itself well to description by the CSF, where the function $\langle\langle .,.,.\rangle\rangle$ is envisaged as an enumeration process which traverses a conceptual space, stepping from one concept to the next, in a sequence defined by $\mathcal{T}$ and possibly influenced by $\mathcal{R}$ and $\mathcal{E}$. In our case, there is a more complicated interaction, because the conceptual space being traversed is not that of music, but that of programs. While we would like to argue that Elsie's knowledge and capability is such that she would be able to traverse the space of Tidal music

[7]The issue of the representation of $\mathcal{U}$ influences the notion of identity. In this case we may most usefully consider identity in terms of a psychological space, since any such space will typically be smaller than the mathematical space of program output (Collins, 2008, p. 240).



```
$ brak $ sound
"[bd [sn/2 -
bd], [hh]*3]"
```

$\mathcal{U}$
$\mathcal{C}_{\mathcal{TP}}$
$\mathcal{C}_{\mathcal{TM}}$
P
M
Tidal execution rule
a)
$\mathcal{C}_{\mathcal{P}}$
$\mathcal{C}_{\mathcal{M}}$
$\mathcal{R}_{\mathcal{M}}$
$\mathcal{R}_{\mathcal{P}}$
b)
$\mathcal{E}_{\mathcal{M}}$
a
b
c
d
e
f
g
c)

Figure 1: a) The defined conceptual space of Tidal programs, and its corresponding conceptual space of music. This structure, represented here as a Venn diagram, forms the basis of our argument. The program at point P in the conceptual space of Tidal programs, $\mathcal{C}_{\mathcal{TP}}$, corresponds with the music at point M in the conceptual space of music generated by Tidal programs, $\mathcal{C}_{\mathcal{TM}}$. The dashed arrows indicate the relationship between the program and the music and their respective points in the conceptual spaces; the dotted arrow represents the process of execution of Tidal. b) Elsie's personal conceptual spaces of Tidal programs and Tidal music may not match exactly to the objective spaces. Specifically, $\mathcal{C}_{\mathcal{P}}$ is smaller than $\mathcal{C}_{\mathcal{TP}}$, and $\mathcal{C}_{\mathcal{M}}$ may be smaller than $\mathcal{C}_{\mathcal{TM}}$ and also include music that is not included in $\mathcal{C}_{\mathcal{TM}}$. c) Elsie's music-aesthetic preferences, $\mathcal{E}_{\mathcal{M}}$, are expressed as a rule set which select a range of the available possibilities. We use a simple yes/no approval rating here for simplicity, but a continuous fuzzy set membership could be used if richer expression were required (Wiggins, 2006a; Ritchie, 2012).

directly, merely selecting the appropriate program to achieve what she wants, any programmer knows that such exactitude may be expected only for trivial cases. Therefore, such a model would be unrealistic.

We model Elsie's traversal of the conceptual space of programs with a rule set, $\mathcal{T}_{\mathcal{P}}$, and the corresponding notional traversal of the space of Tidal music with $\mathcal{T}_{\mathcal{M}}$. Because, as mentioned above, the execution function, $\mathcal{X}$, of Tidal maps from $\mathcal{C}_{\mathcal{P}}$ to $\mathcal{C}_{\mathcal{M}}$, many to one, there is an interaction between $\mathcal{T}_{\mathcal{P}}$ and $\mathcal{T}_{\mathcal{M}}$ which can be partly explained in terms of $\mathcal{X}$. For each program, $p_i \in \mathcal{C}_{\mathcal{P}}$, there is a corresponding musical performance, $m_i = \mathcal{X}(p_i)$. Elsie traverses $\mathcal{C}_{\mathcal{P}}$ by means of application of $\langle\langle \mathcal{R}_{\mathcal{P}}, \mathcal{T}_{\mathcal{P}}, \emptyset \rangle\rangle$ to a vector of programs, $\bar{p}$, which Elsie has already conceptualised. In some cases, this will merely result in selection: Elsie will choose a code fragment that she uses frequently, perhaps to achieve a known effect, or to begin an improvisation sequence with a personal signature. In other cases, she will be generating new programs from old, perhaps

Table 2: Analysis of knowledge required to refer to music in one of the labelled areas in Fig. 1b, in context of Elsie's aesthetic preference.

a	music achievable through a Tidal program that is neither imaginable nor liked by Elsie
b	music that Elsie can imagine, and that is achievable through a Tidal program, but Elsie does not like it
c	music that Elsie can imagine but does not like, and which is not programmable in Tidal
d	music that Elsie can imagine and likes, and that is programmable in Tidal
e	music that is programmable in Tidal, and that Elsie would like, but that she has not (yet) conceptualised
f	music not achievable in Tidal, but which Elsie can imagine and likes
g	music not achievable in Tidal, which Elsie cannot conceptualise, but which she would like if she could

by conceptual blending (Turner and Fauconnier, 1995) or bisociation (Koestler, 1964; Berthold, 2012). At our current level of abstraction, however, the specific function is unimportant: its details are tucked neatly away inside $\mathcal{T}_{\mathcal{P}}$. The application of the function generates a new vector of programs, $\bar{q} = \langle\langle \mathcal{R}_{\mathcal{P}}, \mathcal{T}_{\mathcal{P}}, \emptyset \rangle\rangle(\bar{p})$. At this point, we can identify the nature of the latest product: the Tidal music at point $\mathcal{X}(q_i)$ in $C_{\mathcal{TM}}$(where $q_i \in \bar{q}$) will fall into one of the areas *a*, *b*, *d*, *e*, in Table 2, which we examine in the next section.

6.3 Sharing Creative Responsibility

We are now in a position to describe abstractly, with some precision, the actions that Elsie can take as she performs, and the nature of the resulting outputs, in terms of what she knows and likes, and what is possible in Tidal. The question, then, is: what can be done to change the components of this system, so that some of the creative responsibility in Elsie's performance can be shared with the computer, as is the aim of computational creativity (Colton and Wiggins, 2012)?

Clearly, given the hybrid nature of the creative system under discussion, different parts of it are subject to different kinds of modification: Tidal could be enhanced with implementations of one or more of the components of the CSF formalisation; or Elsie could be modified in a way that is necessary less straightforward. However, perhaps the computer can help. Essentially, the potential modifications to Tidal are in two categories: generative power, and reflection. Generative power, here, refers not to generation of music, but to generation of programs that make music. Reflection, in the current model, refers to evaluation of music, and not of programs. Elsie on the other hand, as a healthy human, can reflect; she already has some notion of what she expects from her programs, and an aesthetic by which she judges them.

In sections 4.3, we explain some useful tests, under the general headings of *uninspiration* and *aberration*, that can be applied to a CSF formalisation. The same ideas will be useful in this extended hybrid formalisation. We now consider the cases in Table 2 in turn. Because we have multiple conceptual spaces represented concurrently, it is important to pay attention to the subscripts in the symbology. We will be thinking in terms of $C_{\mathcal{M}}$, the space of music; however, because the objective conceptual space of Tidal music, $C_{\mathcal{TM}}$, has elements that are not in $C_{\mathcal{M}}$,

Area *a* (in Fig. 1) is an area of both *hopeless* and *conceptual uninspiration*, in terms of the CSF. This is the case because, even though the objective conceptual space of Tidal music, $C_{\mathcal{TM}}$, has elements here, these elements are not valued, and because $C_{\mathcal{M}}$ does not include it. To remedy the hopelessness would be to change Elsie's aesthetic, so we do not consider this possibility further. To remedy the conceptual uninspiration without addressing the hopelessness would merely produce unwanted music, so we treat it in the same way.

Area *b* contains music that Elsie does not value, which means that, presumably, she would prefer not to generate it. This entails some kind of filter on the production of code using $\mathcal{T}_{\mathcal{P}}$, and this brings us to the nub of our proposal. We propose two separate ways in which the Tidal system might be enhanced, to allow creative responsibility to be passed to the computer. The first approach is to restrict the syntax that

Elsie is able to use in her programs, in such a way as to divert her performance away from the areas of $\mathcal{C}_{\mathcal{TP}}$ and $\mathcal{C}_{\mathcal{P}}$ that generate music that she does not value. This enhances the creativity of the system, in so far as it improves Elsie's chances of producing a result that is satisfactory to her; and some of the creative responsibility is definitely passed to the computer. It might be argued that the description of a restricted syntax in terms of a filter within $\mathcal{T}_{\mathcal{P}}$ could instead be modelled as a change in $\mathcal{C}_{\mathcal{TP}}$ or $\mathcal{C}_{\mathcal{P}}$, i.e., as examples transformational creativity. However, $\mathcal{C}_{\mathcal{TP}}$ is objectively defined by Tidal, so it cannot be transformed. Changes in $\mathcal{R}_{\mathcal{P}}$, resulting in a smaller $\mathcal{C}_{\mathcal{P}}$ (in order to maximise the intersection of $\mathcal{C}_{\mathcal{P}}$ and $\mathcal{E}_{\mathcal{M}}$) could indeed be conceptualised in terms of transformational creativity. However, the core issue here is that $\mathcal{T}_{\mathcal{P}}$ could still take us beyond any restricted subset of $\mathcal{C}_{\mathcal{P}}$, because by definition $\mathcal{T}_{\mathcal{P}}$ traverses $\mathcal{U}$ and is not therefore restricted to $\mathcal{C}_{\mathcal{P}}$, so an explicit modification of $\mathcal{T}_{\mathcal{P}}$ is necessary in either case. Thus, as noted above, the CSF gives us two distinct (but related) notions of transformational creativity.

The second approach is to replace Elsie's direct manipulation of program code with automated generation of code fragments, which are subsequently selected and/or approved by Elsie. The fragments offered can be filtered in the same way as Elsie's own programs, above, so as to be within Elsie's preferred range. However, both of these approaches entail knowledge about Elsie's preferences that is not currently encoded in Tidal, or, indeed, in other systems of which we are aware. To make the hybrid system effectively creative, we need a mechanism for Elsie to feed back approval to Tidal. We return to all these points in the next section, once our analysis is complete.

Areas c, f and g are imponderable from within the closed system formed by Elsie and her computer. However, given examples of other musics that Elsie values, it would in theory be possible to use the mapping $\mathcal{X}'$ to identify examples within $\mathcal{C}_{\mathcal{P}}$ that would generate music with similar properties. These could then be used to adapt the generation process, away from c, because it is not valued, or towards a member of $\mathcal{C}_{\mathcal{M}}$ that is similar for cases f and g.

Area d is the comfort zone for Elsie: she has conceptualised this music, and she values it. So no action is required in this area, except to gather feedback so that the computer records Elsie's approval.

Finally, area e is interesting because it offers an opportunity to change Elsie's programming behaviour in ways that she will value. In this area, Elsie has not yet conceptualised the music, so it will be surprising to her, and her programming style does not give her access to it; so to have the computer lead her programming towards this area would be of high creative value.

All this reasoning serves no purpose unless a system could be built with the necessary knowledge. In the next section, we identify the capabilities required by a cooperative creative system based on Tidal, to enable it to fulfil the potential suggested by our analysis.

7 Proposal: A Hybrid Creative System Based on Tidal

In order to fulfil the potential that the above analysis suggests, we need three key ingredients. The first is the ability for our computer to relate the meaning of a program to its syntax. The second is for our computer to have a model of our coder's preference. The third is for our computer to manipulate the syntactic contructs available to our coder so as to take on some of the creative responsibility for the music. We outline the potential to build systems that address each of these in turn, with the intention of raising a challenge to builders of systems for algorithmic music of the future.

7.1 Semantics

The key difficulty with any computational art system (or indeed any computational system of any kind), is in predicting its output for any given non-trivial program. The theoretical reasons for this relate to Turing's halting problem (Turing, 1936) and their detail is beyond the scope of this chapter; however, we may summarise by saying that the only way to understand what a program does is to run it and see—but it, or parts of it, may not terminate, in which case we cannot know what it can do in full.

The upshot of this is that, in general, it is very hard to say what a program means, to give it *semantics*. One way of doing so is to consider the "answer set", the fixpoint[8] of the set of possible outputs of the

[8]Recall our operator, $^\diamond$, which computes this in the CSF, from section 4.2. Note that the halting problem does not affect the CSF formulation because there is no actual attempt to enumerate the various sets involved: all the constructs are theoretical.

program iterated until there are no more available. However, this idea is clearly a hostage to the halting problem, because some outputs may be prevented from appearing by non-termination of code that precedes their output in the executing sequence.

Strongly typed functional programming languages, such as Haskell, on which Tidal is based, are particularly well-behaved in terms of understanding their semantics mathematically. That is not to say that they are exempt from the halting problem—they are not—but their strong type checking does make the notion of program well-formation much, much stronger than that in other languages.

Our case, however, is a special one. Tidal is designed to execute programs in loops, and its syntax is designed to work in this way. Specifically, cycles within Tidal are represented by the set of natural numbers, and the principle datatype, `Pattern`, is a map of time to events, which is notionally infinite in length and can be queried given any time interval expressed as a pair of rational numbers (McLean, 2014, pp.64–65). If Elsie restricts her code to operators that are part of Tidal and not part of the underlying language, we can be sure that the programs will halt, and Haskell's type checking confirms that they are well-formed before they are run. This means that it is possible to construct a theoretical space of syntax trees, in which each runnable program is a point. Indeed, it is possible to do this for Haskell programs in general[9]. Such a space is still a representation of syntax, not semantics, but it does allow us to realise an implementation of $C_{\mathcal{T}\mathcal{P}}$, as required by our argument above.

The behaviour of Tidal as a means of controlling the generation of sound gives us an exciting way to provide semantics for our programs. There is extensive research in the literature on methods for analysing sound, in terms of *features*—analytical aspects—which may be more or less perceptually motivated: for example, the ISMIR[10], ICASSP and WASPAA[11] conferences afford extensive possibilities for the analysis of sound along dimensions that may or may not be salient for a given human listener. These features allow the sonic outputs of Tidal to be represented, more or less approximately, as points in a multi-dimensional space in which dimensions correspond to perceptually meaningful qualities (Gärdenfors, 2000). This feature space constitutes an additional level of representation, or domain of information, within $C_{\mathcal{T}\mathcal{M}}$, providing a perceptually motivated space mapping the lower-level acoustic space. Dimensional reduction using standard mathematical techniques such as Principal Component Analysis (PCA: Jolliffe, 2002) may be used to throw away features that add little information, for parsimony. Now, we are in a position to enumerate a relation approximating the function, $\mathcal{X}$, introduced above, and thence to compute its inverse mapping, $\mathcal{X}'$, though we must remember that $C_{\mathcal{T}\mathcal{P}}$ is infinite, and therefore compromise is necessary in doing so: there are various principled ways of limiting search through $C_{\mathcal{T}\mathcal{P}}$, based on techniques from genetic programming and static program analysis.

The infinite size of $C_{\mathcal{T}\mathcal{P}}$ is less of a problem than we might expect, for two reasons. First, $C_{\mathcal{P}}$ is a subset, and, given the finite nature of humans, is probably not infinite. Second, given an initial estimate of $C_{\mathcal{P}}$, it can be expanded piecemeal as Elsie produces her work, and so exhaustive enumeration becomes unnecessary: instead, the system learns about its user as she uses it. $C_{\mathcal{M}}$, for non-infinite $C_{\mathcal{P}}$, can be computed off line, and there is an excellent application here for cloud computing: a shared effort to identify as much as possible of $C_{\mathcal{T}\mathcal{M}}$ would generate a valuable resource indeed.

7.2 Identifying Value

Given the semantic mapping proposed above, it becomes possible to learn Elsie's preference, expressed as rules in $\mathcal{E}_{\mathcal{M}}$, whose evaluation, $[\![\mathcal{E}_{\mathcal{M}}]\!]$, yields a value which can be viewed as an extra dimension in the extensional set of points in $C_{\mathcal{M}}$ subtended by the range of performances she has made with her system. To do this, feedback from Elsie is required. It may be given in terms of explicit ratings of the music that is currently happening (perhaps by buttons expressing positive, neutral and negative affect, or by a slider over a similar range); alternatively, affective response might be measured indirectly, for example by timing how long Elsie allows a given program to run (assuming that she will replace music she does not value quite quickly), or by measuring physiological responses, such as skin conductance or heartbeat, though the physiological approaches are subject to the drawback that being in a performance situation may cause them

[9]Forth (2012) applies a similar approach to musical-metrical trees, with syntactic considerations drawn from music theory instead of computer science.

[10]`www.ismir.net`

[11]`www.waspaa.com`

to fluctuate. However they are gathered, the responses will allow us to categorise regions of $\mathcal{C}_{\mathcal{M}}$ according to how much Elsie values them. Having done so, we can use $\mathcal{X}'$ to lead us back to the program(s) that create that music, and it is this possibility that admits computational creativity into our hybrid system. This involves making assumptions about the nature of $\mathcal{E}_{\mathcal{M}}$, and this is an interesting area for further research: would a further PCA of the perceptual features of $\mathcal{C}_{\mathcal{M}}$ with the addition of the value dimension, give a different, possibly more useful, reduction applicable to the larger space of $\mathcal{C}_{\mathcal{TM}}$?

7.3 Transforming Human Creativity

Given an estimate of the value that Elsie places on each piece of music, the computer can analyse each program that Elsie writes, mapping its $\mathcal{C}_{\mathcal{TM}}$ value via $\mathcal{X}$, to its equivalent point in $\mathcal{C}_{\mathcal{M}}$, whose value, given by application of $[\![\mathcal{E}_{\mathcal{M}}]\!]$, is known. From this, the system could feed back to Elsie, before executing a piece of code, if it will generate music in a region with which she has associated negative value previously. Thus, the computer system detects pointless aberration (see section 4.3.3), and is able to apply transformational creativity of its human, by influencing her $\mathcal{T}_{\mathcal{P}}$. This corresponds to rejecting areas *a* and *b* in Fig. 1c.

Conversely, and more interestingly, in the event that Elsie is exploring an area of $\mathcal{C}_{\mathcal{TM}}$ that is new to her, the computer may be able to make predictions from $[\![\mathcal{E}_{\mathcal{M}}]\!]$ about which nearby points, so far unexplored, are likely to be valued. It is thence possible to map back to corresponding points in $\mathcal{C}_{\mathcal{P}}$, and present Elsie with a range of programs to try. Again, this is a kind of transformational creativity: $\mathcal{C}_{\mathcal{M}}$ and $\mathcal{C}_{\mathcal{P}}$ are expanded, and $\mathcal{E}_{\mathcal{M}}$ would be modified to reflect Elsie's evaluation of the result. Here, points in areas *a* or *e* is being moved into area *b* or *d* in Fig. 1c.

8 Conclusion: Transforming Computational Creativity

We now look beyond the analysis possible in the restricted space of the chapter. Any or all of the above operations can in principle lead to changes in $\mathcal{C}_{\mathcal{P}}$ and $\mathcal{R}_{\mathcal{P}}$. Given an appropriate metric on $\mathcal{C}_{\mathcal{TP}}$ (which may also be aesthetically motived), we can consider beginning to traverse $\mathcal{C}_{\mathcal{P}}$ automatically, using, for example, genetic programming (GP). At this point, Elsie can begin to relax her artistic control, and really work with the system: she can, for example, restrict her "coding" to telling the computer when to change, or to evaluating the outputs, perhaps intervening when things go too far from her preference. If she observes the results in detail, there will be feedback into her own $\mathcal{C}_{\mathcal{P}}$ and $\mathcal{C}_{\mathcal{M}}$, which themselves will feed back into her use of the system and thus inform future transformations.

Thus, Elsie achieves not the "singularity" of science fiction, but a duality in which she is working on an equal creative basis with the computer, with shared notions of artefact and of meaning. It would in principle be possible to estimate Elsie's $\mathcal{E}_{\mathcal{M}}$ as a function, and thus simulate her musical aesthetic. However, we propose that this would be pointless: the computer, as far as all evidence suggests, has no qualia, and therefore aesthetic response, we suggest, is best left to the entities that seem most likely to be conscious of it.

Acknowledgements

The authors are supported by the Lrn2Cre8 project, which acknowledges the financial support of the Future and Emerging Technologies (FET) programme within the Seventh Framework Programme for Research of the European Commission, under FET grant number 610859.

References

Baars, B. J. (1988). *A cognitive theory of consciousness*. Cambridge University Press.

Babbitt, M. (1965). The use of computers in musicological research. *Perspectives of New Music*, 3(2):74–83.

Berthold, M. R., editor (2012). *Bisociative Knowledge Discovery*, volume 7250 of *LNCS/LNAI*. Springer.

Boden, M. A. (1998). Creativity and artificial intelligence. *Artificial Intelligence Journal*, 103:347–356.

Boden, M. A. (2004). *The Creative Mind: Myths and Mechanisms*. Routledge, London, UK, 2nd edition.

Bovermann, T. and Griffiths, D. (2014). Computation as material in live coding. *Computer Music Journal*, 38(1):40–53.

Buchanan, B. G. (2001). Creativity at the metalevel. *AI Magazine*, 22(3):13–28. AAAI-2000 presidential address.

Bundy, A. (1994). What is the difference between real creativity and mere novelty? *Behavioural and Brain Sciences*, 17(3):533–534.

Clark, A. and Chalmers, D. (1998). The extended mind. *Analysis*, 58(1):7–19.

Collins, N. (2008). The analysis of generative music programs. *Organised Sound*, 13:237–248.

Colton, S., Pease, A., Corneli, J., Cook, M., Hepworth, R., and Ventura, D. (2015). Stakeholder groups in computational creativity research and practice. In Besold, T., Schorlemmer, M., and Smaill, A., editors, *Computational Creativity Research: Towards Creative Machines*. Atlantic Press.

Colton, S., Pease, A., Corneli, J., Cook, M., and Llano, M. T. (2014). Assessing progress in building autonomously creative systems. In *Proceedings of the Fifth International Conference on Computational Creativity*.

Colton, S. and Wiggins, G. A. (2012). Computational creativity: The final frontier? In de Raedt, L., Bessiere, C., Dubois, D., and Doherty, P., editors, *Proceedings of ECAI Frontiers*.

Ebcioğlu, K. (1988). An expert system for harmonizing four-part chorales. *Computer Music Journal*, 12(3):43–51.

Eigenfeldt, A. (2014). Generative music for live performance: Experiences with real-time notation. *Organised Sound*, 10(3).

Eigenfeldt, A., Burnett, A., and Pasquier, P. (2012). Evaluating musical metacreation in a live performance context. In Maher, M. L. et al., editors, *Proceedings of ICCC 2012*.

Forth, J. C. (2012). *Cognitively-motivated geometric methods of pattern discovery and models of similarity in music*. PhD thesis, Goldsmiths, University of London.

Gärdenfors, P. (2000). *Conceptual Spaces: the geometry of thought*. MIT Press, Cambridge, MA.

Jolliffe, I. T. (2002). *Principal Component Analysis*. Springer Series in Statistics. Springer New York, second edition.

Kirnberger, J. P. (1757). Der allezeit fertige menuetten- und polonaisencomponist [the ever-ready minuet and polonaise composer].

Koestler, A. (1964). *The Act of Creation*. Hutchinson & Co., London.

Lutosławski, W. (1961). Jeux vénitiens [Venetian Games].

Magnusson, T. (2014). Herding cats: observing live coding in the wild. *Computer Music Journal*, 38(1):8–16.

McLean, A. (2011). *Artist-Programmers and Programming Languages for the Arts*. PhD thesis, Goldsmiths, University of London.

McLean, A. (2014). Making programming languages to dance to: Live coding with tidal. In *Proceedings of the 2nd ACM SIGPLAN International Workshop on Functional Art, Music, Modeling & Design (FARM '14)*, pages 63–70. ACM.

McLean, A. and Wiggins, G. (2010a). Bricolage programming in the creative arts. In *Proceedings of the 22nd Psychology of Programming Interest Group*.

McLean, A. and Wiggins, G. (2010b). Tidal - pattern language for the live coding of music. In *Proceedings of the 7th Sound and Music Computing conference*.

McLean, A. and Wiggins, G. A. (2010c). Live coding towards computational creativity. In Ventura et al., editors, *Proceedings of the First International Conference on Computational Creativity*.

Potter, K. (2000). *Four Musical Minimalists: La Monte Young, Terry Riley. Steve Reich, Philip Glass*. Cambridge University Press, Cambridge, UK.

Revill, D. (1993). *The Roaring Silence: John Cage: A Life*. Arcade Publishing.

Riley, T. (1964). In c.

Ritchie, G. (2007). Some empirical criteria for attributing creativity to a computer program. *Minds and Machines*, 17(1):67–99.

Ritchie, G. (2012). A closer look at creativity as search. In *Proceedings of the 3rd International Conference on Computational Creativity*, Dublin, Eire.

Ritchie, G. D. and Hanna, F. K. (1984). AM: A case study in AI methodology. *Artificial Intelligence*, 23:249–268.

Thompson, S. (2011). *Haskell: the craft of functional programming*. Addison-Wesley Educational Publishers Inc, 3 edition.

Turing, A. (1936). On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 2(42):230–65.

Turner, M. and Fauconnier, G. (1995). Conceptual integration and formal expression. *Metaphor and Symbolic Activity*, 10(3):183–203.

Wiggins, G. A. (2006a). A preliminary framework for description, analysis and comparison of creative systems. *Journal of Knowledge Based Systems*, 19(7):449–458.

Wiggins, G. A. (2006b). Searching for computational creativity. *New Generation Computing*, 24(3):209–222.

Wiggins, G. A. (2012a). The future of (mathematical) music theory. *Journal of Mathematics and Music*, 6(2):135–144.

Wiggins, G. A. (2012b). Music, mind and mathematics: Theory, reality and formality. *Journal of Mathematics and Music*, 6(2):111–123.

Wiggins, G. A. (2012c). On the correctness of imprecision and the existential fallacy of absolute music. *Journal of Mathematics and Music*, 6(2):93–101.

Wiggins, G. A. and Forth, J. (2015). IDyOT: A computational theory of creativity as everyday reasoning from learned information. In Besold, T. R., Schorlemmer, M., and Smaill, A., editors, *Computational Creativity Research: Towards Creative Machines*, Atlantis Thinking Machines. Atlantis Press.

Wright, M. and Freed, A. (1997). Open sound control: A new protocol for communicating with sound synthesizers. In *International Computer Music Conference*, pages 101–104, Thessaloniki, Hellas. International Computer Music Association.