Journal of Pedagogic Development
Volume 6, Issue 1

# Teaching Rule-Based Algorithmic Composition: The PWGL Library Cluster Rules

Torsten Anders, Department of Media Arts and Production, University of Bedfordshire
**Contact: torsten.anders@beds.ac.uk**

**Abstract**
This paper presents software suitable for undergraduate students to implement computer programs that compose music. The software offers a low floor (students easily get started) but also a high ceiling (complex compositional theories can be modelled). Our students are particularly interested in tonal music: such aesthetic preferences are supported, without stylistically restricting users of the software.

We use a rule-based approach (constraint programming) to allow for great flexibility. Our software Cluster Rules implements a collection of compositional rules on rhythm, harmony, melody, and counterpoint for the new music constraint system Cluster Engine by Örjan Sandred.

The software offers a low floor by observing several guidelines. The programming environment uses visual programming (Cluster Rules and Cluster Engine extend the algorithmic composition system PWGL). Further, music theory definitions follow a template, so students can learn from examples how to create their own definitions. Finally, students are offered a collection of predefined rules, which they can freely combine in their own definitions.

Music Technology students, including students without any prior computer programming experience, have successfully used the software. Students used the musical results of their computer programs to create original compositions.

The software is also interesting for postgraduate students, composers and researchers. Complex polyphonic constraint problems are supported (high ceiling). Users can freely define their own rules and combine them with predefined rules. Also, Cluster Engine's efficient search algorithm makes advanced problems solvable in practice.

**Keywords**: research-based learning; practice-led research; algorithmic composition; constraint programming; visual programming; PWGL

## Introduction

This paper presents research directed at supporting students in learning algorithmic composition. In algorithmic composition (Nierhaus, 2009), musicians develop their own software that generates music, which can lead to new musical ideas, and to compositional techniques that are difficult to realise 'by hand'. Studying algorithmic composition is useful for Music Technology students, because it brings important areas of music technology together. Students strengthen artistic skills in composition, as well as technical skills in computer programming. Music composition is part of many Music Technology and similar courses in the UK, and so is computing (Boehm, 2007) – in particular software development using visual programming languages. The key is that students reflect composition techniques and music theory in order to computationally model them.

For this project, students are learning in research mode (research-based and research-tutored learning (Healey & Jenkins, 2009)) by conducting practice-led research (Barrett & Bolt, 2010) in an established area with a long history. Learning from first-hand research experience is highly valuable for all students (Healey & Jenkins 2009).

We want to enable students with little or no computer programming experience to computationally model music theories. It should be possible for them to model complex music theories that at the same time restrict the rhythmic, melodic, harmonic, and motivic structure of music. However, such complex theories should be reasonably easy to implement.

We propose realising this goal with a collection of ready-made compositional rules for a visual rule-based algorithmic composition system, and a prototype has been developed for this project. This collection contains various rules restricting the rhythm, melody and harmony. Students model their own music theories and that way shape their music by freely combining rules, and by customising the effect of these rules with rule parameters. The prototype rule collection is called Cluster Rules, which complements the music constraint system Cluster Engine, a successor of PWMC (Sandred, 2010); all these systems are libraries of the visual programming system PWGL (Laurson et al. 2009).[1]

The present research is conducted at a UK university with a widening participation (Hinton-Smith, 2012) agenda. It aims at presenting algorithmic composition in a way that is attractive to Music Technology students at such a university.

## Motivation and Context

### *Teaching Tonal Composition with Algorithmic Composition Tools*

For centuries, composers have aimed at balancing musical ideas and expression with a coherent organisation of musical form. Algorithmic composition (score synthesis) offers an alternative approach to that end, as it allows its practitioners to work on a higher level by focussing on a general development or process instead of composing all individual notes. Algorithmic composition also allows composers to surprise themselves; and computational models of music composition and theory can lead to a better understanding of how music 'works'.

Composition should be taught in a way that is engaging for students and which taps into their intrinsic motivation. This is especially the case at a university with a high percentage of non-traditional students. Students in our course typically want to compose music largely in a mainstream musical idiom, or the idiom of certain sub- and countercultures. For example, virtually all of our students aim for tonal music, and most often they want a clear rhythmic structure, as prevails in most popular music styles. Students are less interested in 'experimental music', e.g., traditions such as electroacoustic music (Landy, 2007) – even after they have been introduced in their course to various musical traditions including 20th/21st century music languages. Their musical interests are likely a result of both their musical upbringing, as well as their clear interest in the vocational applicability of knowledge and skills learned in their course.

The writing of tonal music is traditionally trained by studying music theory subjects such as harmony – the organisation of chords and chord progressions (Schoenberg, 1983); and counterpoint – how multiple parts can accompany each other (Piston, 1947); complemented by composition practice (Schoenberg, 1967; Russo et al. 1988; Cunningham, 2007).

However, this traditional approach is not well suited for music technology students: it is intended for several years of intensive compositional training, and is centred on music notation using pen and paper. Music technology students do not have that amount of time to study composition, many struggle with music notation, and they are not keen on using pen and paper for writing music – they prefer software. This traditional approach has therefore been adapted for music technology students: Hewitt addresses the users of modern Digital Audio Workstations when introducing music theory subjects such as harmony (Hewitt, 2011), and compositional tasks such as writing bass lines and melodies (Hewitt, 2009).

The present project tries a different approach to teaching composition, which can complement the above by Hewitt. As our students prefer using software when composing, why not try to use software for the compositional process itself, that is, use algorithmic composition techniques?

This research project proposes an algorithmic composition platform suitable for our undergraduate students. In particular, this platform also allows for modelling tonal music and clear rhythmic structures.

### *Rule-Based Programming (Music Constraint Programming)*

For the educational purposes of this project, we wanted an algorithmic composition technique that supports tonal music composition, and that allows our students great flexibility in shaping their generated music. It should also be reasonably easy to learn, but that aspect is discussed later in more detail.

A wide range of techniques have been proposed (Nierhaus, 2009), including various methods that stem from artificial intelligence (Fernández & Vico, 2013). Rule-based systems have often been used for modelling tonal music, and the flexibility of this approach for music composition has been demonstrated

[1] The composition environment PWGL and the libraries presented in this paper are all freely available online. PWGL: http://www2.siba.fi/PWGL/; Cluster Engine: http://sandred.com/Downloads.html; Cluster Rules https://github.com/tanders/cluster-rules.

by the wide range of existing applications (Anders & Miranda, 2011). We therefore decided for a rule-based approach. For centuries, musicians have used compositional rules for describing musical characteristics (e.g., the anonymous treatise *Musica enchiriadis* already used rules in the 9th century for describing the composition of an organum, an early polyphonic form). So, learning to think in terms of rules is useful for students beyond the field of algorithmic composition.

Generating a score from a rule-based description of the desired rhythm, harmony and so forth can be efficiently implemented by constraint programming (Apt, 2003). In this field, a music score contains *variables* (unknowns, as in algebra), e.g., note pitches and durations can be variables. Users restrict such variables by applying rules (*constraints*) to them. Such rules can implement traditional music theory rules (e.g., rules of harmony), or experimental compositional concepts, including concepts defined by users. In order to limit the size of the search space, variables are defined with a finite *domain* – possible values they can take in a solution (e.g., possible rhythmic values for note durations). A constraint *solver* then searches for one or more *solutions* that fulfil all rules applied.

Constraint programming has already been used for decades for modelling music theory and composition. Anders and Miranda (2011) provide an extensive survey of this research area, and compare in detail several music constraint systems that allow users to model their own music theories. Another review focuses on modelling harmony (Pachet & Roy, 2001). Anders (2016) surveys several compositions composed with the help of various music constraint programming systems.

### ***Visual Programming***

For this project we looked for a music constraint system that makes it easy to get started (low floor) but also allows for highly complex music theory definitions (high ceiling; Resnick et al. 2009). Users of algorithmic composition systems (including music constrain systems) develop their own computer programs, but learning to program is challenging (Jenkins, 2002). A visual programming language lowers the floor, so such a language was preferable.

Visual languages are easier to learn for new users without programming experience because users need to learn less syntax (e.g., in systems like PWGL the main syntax element is connecting boxes with patch chords, see figures below for examples) and therefore fewer errors can occur. Also, users can select boxes from menus – they do not need to remember many keywords, functions, names, etc. to get started. Such ease of use contributes greatly to the popularity of visual programming systems for music and sound.

Nevertheless, a main challenge in programming is the actual problem solving (Michaelson, 2015), and this aspect cannot really be simplified with visual programming or any other programming paradigm for that matter (van Roy & Haridi, 2004). We tried to reduce this challenge for students by providing them with templates for music theory definitions where they could then insert and remove ready-made rules of the Cluster Rules library. The main template is shown below in Figure 1, and later examples demonstrate how this template can be extended.

### **Cluster Engine**

For this project we chose the PWGL library Cluster Engine by Örjan Sandred as the foundation, because it is designed for composing polyphonic music, and users can freely constrain both pitches and rhythm. Also, users can define their own rules visually.[2] Other visual music constraint programming systems have been developed before, but these systems are far more restricted (Anders & Miranda, 2011). Cluster Engine is therefore presented in some detail in this section.

Cluster Engine provides a constraint solver designed specifically for solving complex polyphonic music definitions. The library is similar in its functionality to the earlier system PWMC (Sandred, 2010) by the same author. However, Cluster Engine solves complex polyphonic problems far more efficiently than PWMC did, in particular problems where both the temporal and pitch structure are constrained. Such improved efficiency makes many more advanced problems solvable in practice.

A simple example (Figure 1) explains common boxes of the library that are used in other examples below. The box `clusterengine` is the heart of the library: it represents the constraint solver, and it

[2] Advanced users can program rules more concisely in the textual language Common Lisp, and Cluster Rules was also defined this way.

also encapsulates the internal music representation of the library. This box expects the following inputs (among others):

- The *rhythm domain* sets possible note durations for a specific voice. In the example, quarter notes (crotchets) and eighth notes (quavers) are permitted, encoded by the fractions 1/4 and 1/8.
- The *pitch domain* sets possible pitches for a specific voice. The figures in the example (60, 62, …) represent MIDI note numbers (Rothstein, 1995), i.e., certain white keys on a music keyboard (C4, D4, E4, …).
- `clusterengine` expects an arbitrary number of rules. The box `rules->cluster` can be extended to an arbitrary number of inputs, see Figure 3.

The pitch and duration domains allow for pre-composed sequences (motifs). Such sequences are enclosed in parentheses. This is also the reason why in Figure 1 all individual values in the pitch and the duration domain are enclosed in parentheses.

For more voices `clusterengine` can be extended (up to 10 voices at maximum). `clusterengine` typically outputs the solution score (in this case following the rhythm and pitch domains, but no additional rules). A wide range of music theories can be defined by extending this template basically by adding rules.

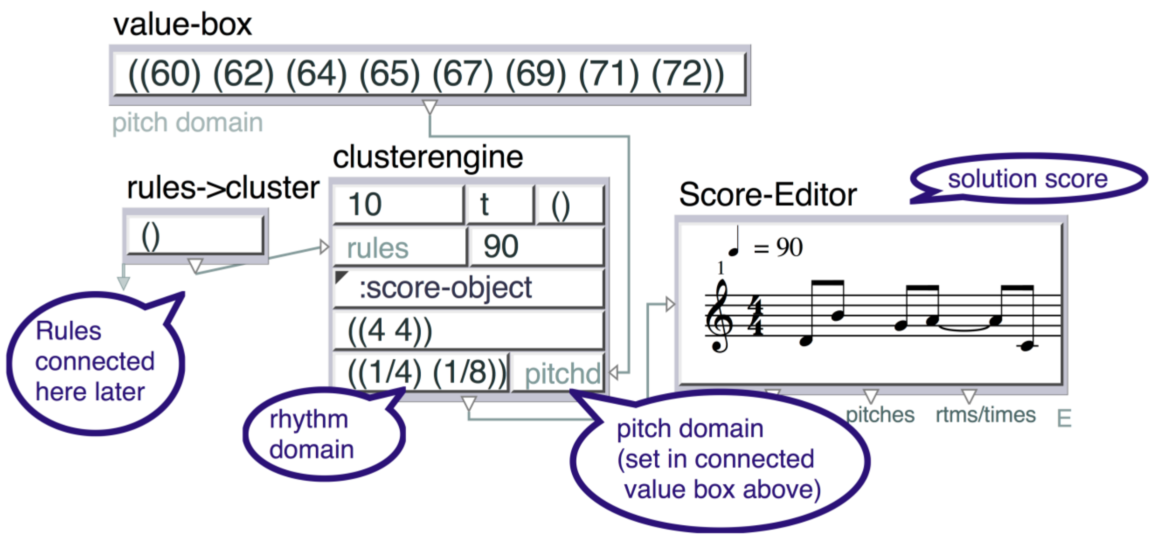

**Figure 1: Dummy example demonstrating common boxes of Cluster Engine, see text for details**

Cluster Engine predefines only few rules, but users can define their own rules. The library provides a rich set of boxes that control which score contexts (which sets of pitch, duration, and/or time signature variables) are constrained by a user-defined rule (e.g., consecutive pitches in a voice, or simultaneous pitches across multiple voices). The expressive power of these rule applicators (called access boxes by Sandred, 2010) is one of the major strengths of this system.

## Cluster Rules

This project aimed at helping students to start doing their own practice-based algorithmic composition research. Remember that we wanted to empower students with little programming experience to computationally model their own music theories, including complex theories that restrict at the same time the rhythmic, melodic, and harmonic structure of music.

For this purpose we provided students with a collection of predefined musical rules, which they could freely combine. The PWGL library Cluster Rules predefines rules for the library Cluster Engine, and that way greatly simplifies getting started to use Cluster Engine (low floor).

Nevertheless, Cluster Rules does not restrict the flexibility of Cluster Engine (high ceiling). More experienced users can freely mix the predefined rules with their own rules. Also, all advanced features of Cluster Engine like its motif domains are supported. The rest of this section gives an overview of the range of rules provided by Cluster Rules.

***Parameterised Rules***

In the simplest case, ready-made compositional rules are just selected and applied by users. However, such an approach would either restrict greatly the range of music theories that can be modelled, or alternatively require a very large number of predefined rules. Instead, users of Cluster Rules can customise the effect of each rule with various rule parameters.

Figure 2 (a) shows a rule example with three parameters. If users click on a rule name (here, `no-repetition`) then PWGL shows the parameter names (Figure 2 (b)), which are documented in the reference for each rule. This particular melodic rule disallows the repetition of notes in the given voices (here `(0 1)`, meaning the first and second voice counting from the top stave) within a window of a given number of notes (here `3`, i.e., no repetitions among three consecutive notes are allowed). At the third parameter users can select by a menu what note attributes to consider (e.g., disallowing repetitions of actual pitches or pitch classes, i.e., notes in different octaves with the same name).

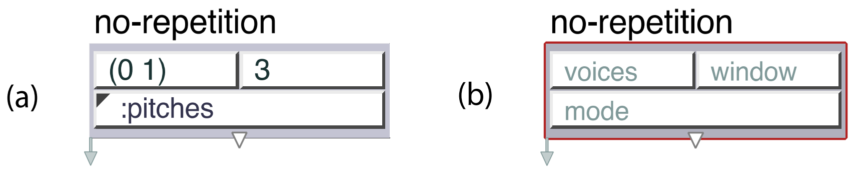

**Figure 2: (a) an example rule with three parameters of different types: a number, a sequence of numbers (enclosed in parentheses), and a value selected by a menu. (b) The parameter names are shown when the user clicks on the rule name.**

For simplicity, parameters that do not expect a number or a list of numbers can usually be selected from a menu. Again for simplicity, meaningful rule parameter defaults are chosen such that rules can usually be used directly without customising their parameters first.

To further simplify the interface of rules, some parameters are optional, and only revealed on request. For example, by default rules implement strict constraints, but an optional argument for almost all rules allows to turn them into a heuristic rule that expresses a mere preference (soft constraint) (Sandred, 2010).

***Melodic Rules***

**Local Contexts** Several rules control melodic motion between consecutive notes. Some rules enforce melodic lines to follow conventions of tonal music, which addresses aesthetic student preferences. Such rules restrict, e.g., the maximum melodic interval, or the set of melodic intervals allowed. Other rules require skips to be resolved by a step in the opposite direction. These rules are inspired by conventional counterpoint, e.g., rules by Jeppesen (1939).

Further rules are less conventional, but offer interesting ways to shape results. For example, it is possible to enforce a minimum melodic interval, or how many consecutive intervals can be ascending or descending.

Again, all these rules are parameterised. For example, users can specify the minimum skip size that should be resolved, and the maximum interval allowed as resolution. Tweaking rule parameters can enforce more or less conventional results.

**Profile Rules: Large-Scale Contexts** Profile rules (Schilingi, 2009) allow users to outline how a melody should roughly develop over time. Figure 3 shows a simple example. The melodic curve is constrained by a profile – specified graphically as break-point function by a PWGL 2D-Editor – to move first up and then down. This profile is given to the actual profile rule (`follow-profile-hr`). Rule-related boxes have usually a darker border for clarity in this paper.

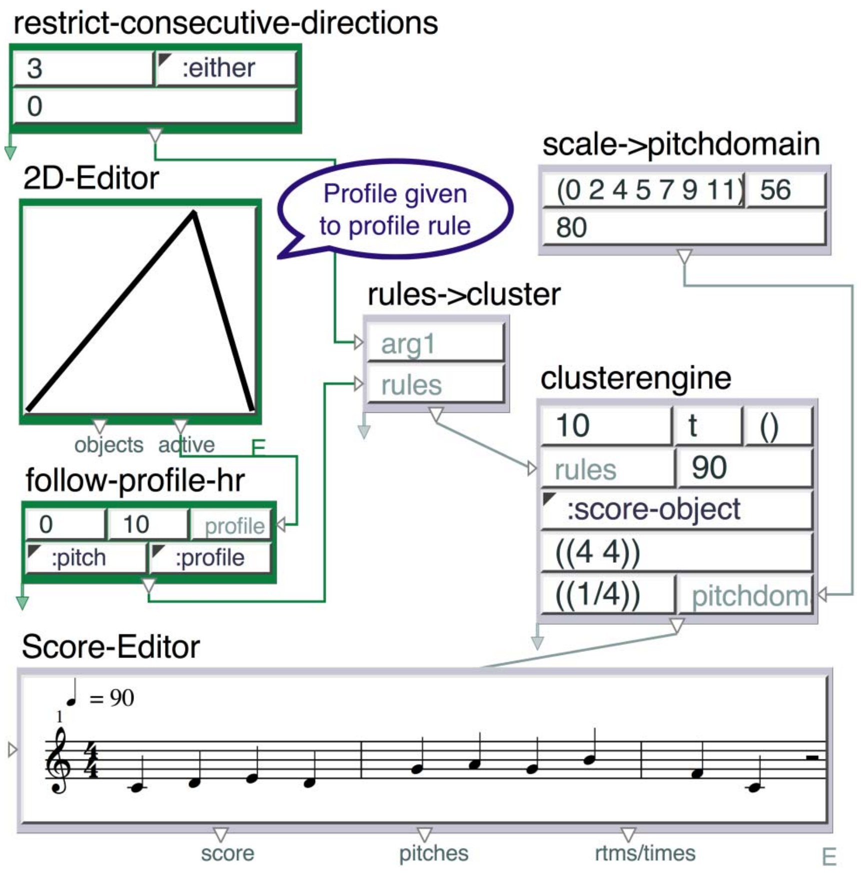

**Figure 3: Patch constraining a melody to follow a pitch profile, see text for details**

The domain of note durations of the only voice here is fixed to `1/4` (only crotchets are allowed). The pitch domain is specified with the convenience box `scale->pitchdomain`, which generates a domain specification from a list of pitch classes (here the C-major scale) and MIDI note numbers for the lower and upper boundary.

Profile rules are heuristic rules. The resulting music tries to follows the profile closely, but if that causes conflicts with other rules then the resulting music deviates from the profile as necessary. In the example, a second rule is applied as well: only three consecutive notes can progress in the same direction (`restrict-consecutive-directions`). The resulting melody 'wraps around' the given profile.

***Rhythm Rules***

Cluster Rules predefines a variety of rules that restrict the rhythm. Traditionally, rhythm has been somewhat neglected by music theory, but controlling the rhythm has great impact on the musical result. Therefore, a number of predefined rules in this category have no equivalent in traditional music theory.

The rhythmic complexity can be restricted with several rules, answering student preferences. For example, some rules control syncopations over beats or bar lines. Other rules restrict the position of tuplet notes to more simple cases.

Some higher-level rules clarify the perceived rhythmic structure. A rule proposed by Sandred (2003) enforces an alignment between simultaneous parts that could be called quasi-homophony, or a rhythmic hierarchy. Figure 4 shows an example, where all the rhythmic onsets of the upper voice are shared by the lower voice. However, the lower voice can have additional note onsets in between (e.g., the penultimate note in the first bar). The shown rhythmic solution is syncopated, but the rule enforces both voices to agree on this syncopation.

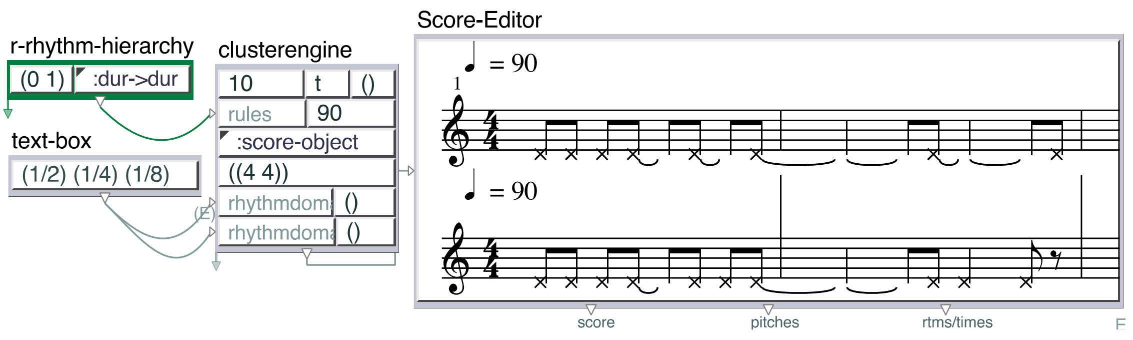

**Figure 4: Quasi-homophonic texture enforced by a rhythmic rule**

Other rules control the position of accented notes in the music (Anders, 2014). For example, longer notes following rather short notes are often perceived as carrying an accent. Some rules control the metric position of such durational accents (Lester, 1986).

***Rules on Simultaneous Pitches***

As mentioned above, our students want to control the harmony of their music. In general, harmony can be controlled by restricting the intervals between simultaneous pitches, which is basically the approach of Renaissance music (Jeppesen, 1939). Alternatively, music can be based on an underlying harmony (e.g., represented by chord symbols), which is the common approach for tonal music since the Baroque (Rameau, 1984), including today's popular music. Cluster Rules supports both approaches with predefined rules.

**Constraining Harmonic Intervals** A number of rules restrict intervals between simultaneous pitches (or pitch classes). Users can control which intervals are allowed or forbidden between certain voices. A classical rule restricts the interval of a fourth between the bass and any higher voice, effectively restricting six-four chords. Other rules restrict the maximum or minimum size of harmonic intervals (e.g. certain voices should not be too far apart from each other), or the number of different simultaneous pitch classes (e.g. it is possible to require always at least three different pitch classes for a full sound).

**Constraining the Harmony** Several rules control how the music follows an underlying harmony. The harmony is represented by an extra voice with chords (and possibly even a further voice for the underlying scale to represent how scales change in modulations). That way, the harmony itself can be constrained simply like actual notes. For example, a rule can require common pitches between consecutive chords (Schoenberg, 1983). When exporting, such analysis stave can be deleted, so that only the actual music is left.

In Figure 5 the music is constrained to follow a simple C-major cadence, shown in the second stave. For simplicity, these chords are defined manually: the pitch domain of the second voice is a chord progression specified in a text box.

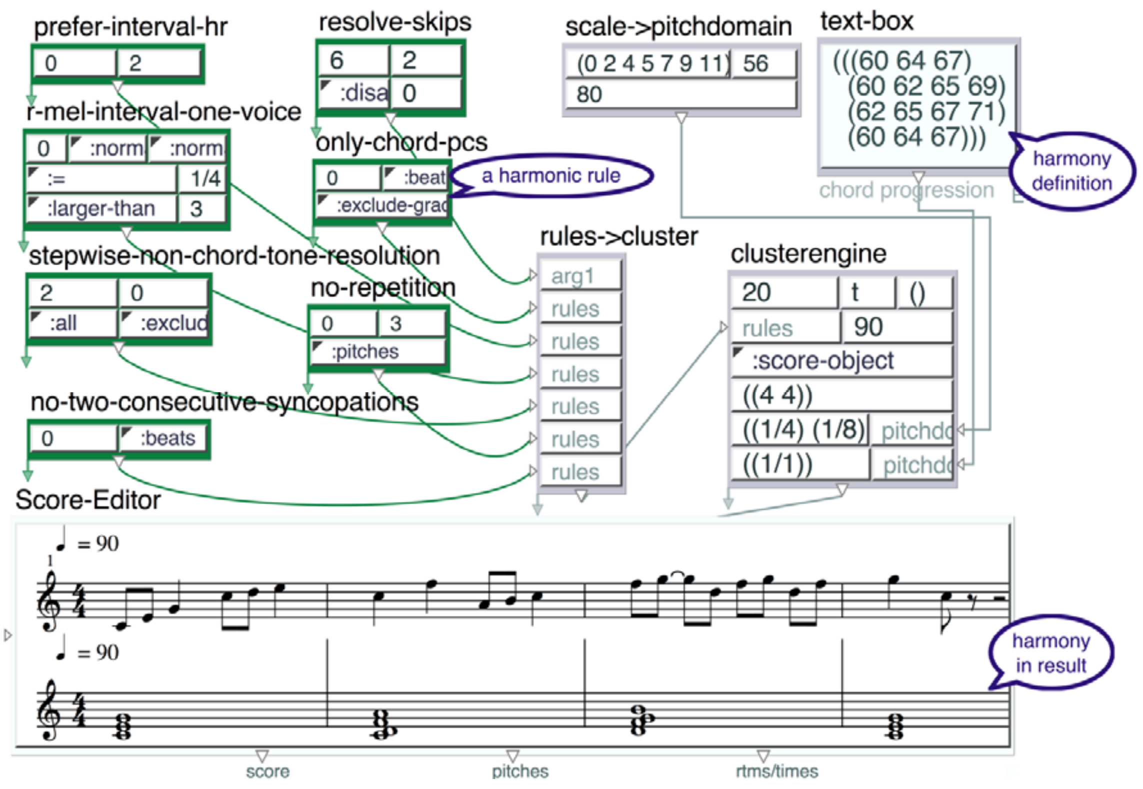

**Figure 5: Patch constraining a melody to follow an underlying harmony (C-major cadence)**

This patch is slightly more complex than the previous examples: there are seven rules applied in total. The rule boxes are arranged to use rather little room, and not in the order in which they are presented in this text.

Two rules restrict the relation between the underlying harmony (second stave) and the melody (first stave) in a traditional way. The rule `only-chord-pcs` restricts all notes starting on a beat (second rule parameter) to a chord tone, but the eighth notes between beats can be non-harmonic tones. Nevertheless, `stepwise-non-chord-tone-resolution` constrains these tones to be reached and left by an interval no greater than 2 semitones (first rule parameter), i.e. a step, which is a standard dissonance treatment.

Some conventional melodic rules further shape the melody. The heuristic rule `prefer-interval-hr` expresses that steps (second parameter: interval size 2) are preferred; `resolve-skips` constrains skips exceeding a tritone (interval size 6) to be resolved by a step (maximal size 2) in the opposite direction; `no-repetition` disallows pitch repetitions within 3 tones.

An unusual rule adds melodic interest: `r-mel-interval-one-voice` requires all quarter notes to be left by an interval larger than 3, i.e. a skip. Finally, a rhythmic rule simplifies the result: `no-two-consecutive-syncopations` forbids syncopations across beats (second parameter).

Note that in Cluster Rules the harmony is not restricted to chords and scales from common practice. Any pitch set can be used as a chord or scale, including microtonal pitches (specified as 'MIDI-floats').

**Voice-Leading Constraints**
Cluster Rules also predefines some common voice-leading constraints. For example, users can prohibit open or hidden parallels, and can restrict voice crossing.

## Evaluation

***Usage***
Cluster Rules has been introduced at a UK university to level 6 (3rd year) undergraduate students: over a single term, five students practised algorithmic composition in an optional unit (module). All students had composed before during their studies. However, many students had little music theory background,

and struggled with music notation. Students could therefore use 'piano roll notation' instead (see Figure 6), supported by the PWGL 2D Editor.

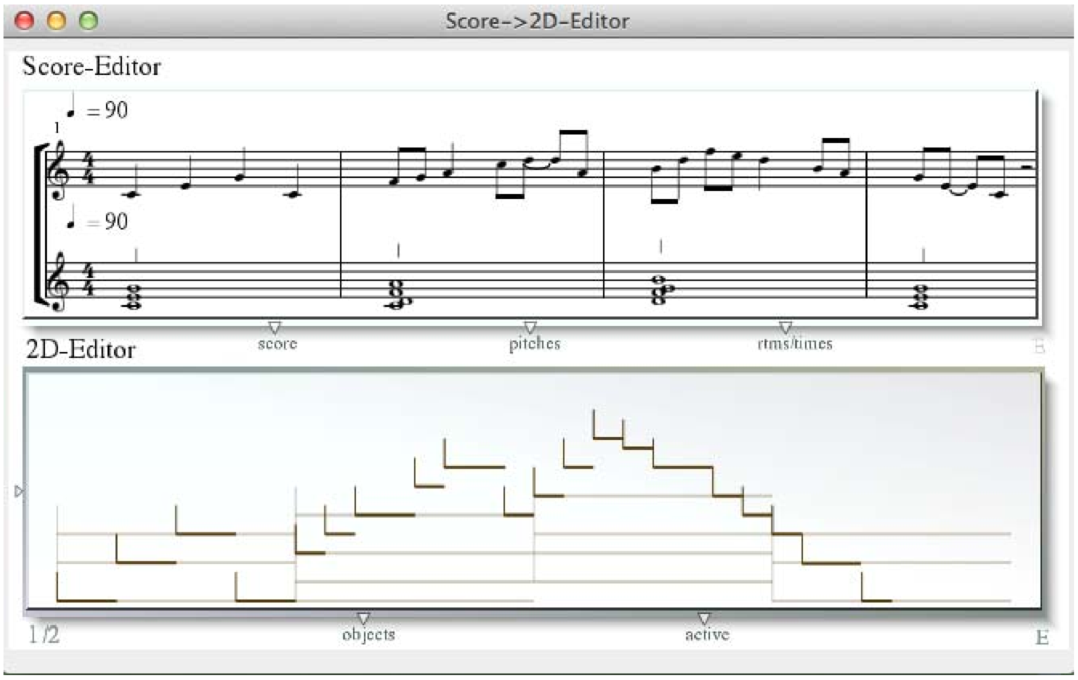

**Figure 6: A score of two parts automatically translated into 'piano roll notation', where these parts are superimposed**

Two students came from a vocational course (two years foundation degree). They did not have any programming experience, and also did little academic work in general before. The other students had used other music programming systems before, namely Pure Data (Kreidler, 2009) and SuperCollider (Wilson et al., 2011). Nevertheless, no student had used PWGL.

Students were introduced to a number of classical algorithmic composition techniques in the unit. For an assessment, students were asked to program an algorithmic composition application with PWGL using a technique of their choice, and later they composed pieces with their patches.

All students chose using Cluster Rules and Cluster Engine (one student combined it with another technique – using the result as a pitch profile rule). This unanimous choice for a rule-based approach was contrary to the tutor's initial anticipation. The reasons are possibly aesthetic preferences. In particular, the underlying harmony is more easily controllable with a rule-based approach. Anyway, such choice indicates that the combination of Cluster Rules and Cluster Engine is suitable for undergraduate students – even for students without any prior programming experience.

## *Discussion*

As anticipated and therefore supported by the design of Cluster Rules, all students wanted to program a patch that controlled the underlying harmony of the resulting music. The unit actually encouraged them by musical examples (e.g., Xenakis and Murail) to go beyond traditional tonal harmony, but that had little observable impact on their compositional intentions.

All students successfully constrained the underlying harmony using Cluster Rules (though some needed more support than others). Nevertheless, the complexity and the character of the harmony of the student patches clearly differed. For example, one student played with a simple yet effective unconventional chord progression, where only a single pitch class changed between consecutive chords (*A, C, E♭ → A, C, E → A, C, F*). Another student created an interface for selecting a scale and a key, and used that interface for composing music that progressed through a variety of scales in various keys.

Students were usually happy (and said so) if the resulting music merely played the harmony tones in figurations, even though Cluster Rules supports more complex musical situations, e.g., control of non-harmonic tones (e.g., passing tones).

Many students struggled to differentiate between an underlying harmony as an analytical concept, and actual chords played, such as in an accompaniment. As previously discussed (p. 9), Cluster Rules represents analytical harmonic information simply as chords in an extra voice. Students were presented example patches where this analysis voice was muted,[3] but when explaining this concept we un-muted the analysis voice. Interestingly, no student later considered to mute such analytical information again, and many students did not even remove this additional homophonic 'accompaniment layer' when later manually revising their compositions using music notation software (Sibelius), or a digital audio workstation (Logic).

In future, Cluster Rules and Cluster Engine could be adapted to instead represent harmonic information differently (e.g., notated by chord symbols). Nevertheless, the flexibility of the current representation would be nice to keep. For example, constraining harmonic relations between chords would be much more difficult in a representation based on chord symbols. We will consider other ways to mark voices as analytical information.

Cluster Rules and Cluster Engine make it easy for students to start defining their own music theories (low floor), but these libraries also allow for highly complex definitions (high ceiling). We got all students started, but students used only a small subset of the available functionality. For example, few students controlled the rhythm beyond specifying duration domains for their voices. Also, students used only a small set of rules, usually those demonstrated and discussed in example patches in class. Most students did not explore further rules on their own, even though documentation for each rule is easily available from within the system. It might be that students lacked confidence to go beyond the material already discussed in class. Moore (2014) found that the confidence of music technology students increased in group work, which we should also explore in future.

Instead, we practised reading the reference documentation of rules; Figure 7 shows an example. Students struggled to comprehend the somewhat formal language and conciseness of such a reference – to the surprise of the tutor, revealing an expert blind spot (Ambrose et al*.,* 2010). We discussed the sentence that summarised the rule `resolve-skip` for a while, but only after students were shown musical examples and a visualisation of the rule (drawn ad hoc at the white board) did they signal understanding.

```
RESOLVE-SKIPS   (SKIP-SIZE RESOLUTION-SIZE REPETITION?)

Resolve any skip larger than skip-size by an interval in the
opposite direction.

Args:
  skip-size: The minimum interval size (in semitones) that
triggers this rule.
  resolution-size: The maximum interval size that is allowed as a
resolution.
  repetition?: Whether or not tone repetitions are allowed as
resolution.
```

**Figure 7: Documentation of a rule (excerpt) that we practised in class**

Future versions of Cluster Rules should therefore simplify the exploration of the system for the benefit of undergraduates. For example, it would be highly useful to complement the comprehensive textual reference documentation with more example patches, which demonstrate the various rules. The default parameters of some rules should be revised to make it easier to start using them. Further, unifying the order of rule parameters would help students understanding and using further rules. For example, most rules support specifying which voices they constrain, but this `voices` parameter can be at different positions.

[3] Voices can be muted in PWGL by an ENP-script (Kuuskankare & Laurson, 2006).

No student defined their own rules, even though we practised the definition of custom rules in class. There are multiple possible reasons for this. While music technology students are used to composing, most had little formal composition training, and they are therefore not used to thinking in terms of compositional rules (in contrast to, say, composition students). Because they know few standard rules, they probably did not miss any rules in system. While composers may like to develop their own rules as part of the composition process (practice-based research), the music technology students were possibly already overwhelmed by the range of rules provided. Lastly, using predefined rules is easier than defining your own.

More generally, many students seemingly had difficulties or did not feel any need to review the musical output of their patches. The tutor and the assessment brief explicitly encouraged students to manually revise the generated music in software like Logic or Sibelius, but most students used their patch outputs as ready-made musical snippets (like loops in Logic) without further editing. In future, we plan to introduce formative peer feedback as a way to stronger encourage revisions, because peer feedback can be more effective than feedback by the tutor (Strijbos et al., 2010).

## Conclusions

This project aimed at enabling Music Technology students with little or no programming experience to computationally model music theories such that students could control at the same time the rhythm, melody, harmony, counterpoint, and motivic structure of music.

The proposed approach to computational music theory modelling offers a low floor. Visual programming paved the way. The rule collection of Cluster Rules enabled students to model their own music theories as a combination of predefined rules. All students controlled the harmony and melody, and some students additionally restricted the rhythm, counterpoint, or specified motifs to use. Students were particularly interested in the control over the harmony offered by the PWGL libraries, and they used the musical results of their patches in original compositions.

This approach also offers a high ceiling. For example, complex polyphonic music can be modelled by students who want to continue on their own after the unit, by postgraduate students, or by researchers. Cluster Engine's efficient search algorithm solves advanced problems in a reasonable amount of time. Due to the integration in an established computer-aided composition ecosystem (PWGL) powerful editors are available (e.g., a score editor); constraint programming can be combined with other algorithmic composition approaches and PWGL libraries; and results can be output into multiple formats (MIDI, MusicXML) for further editing in industry-standard software such as Logic or Sibelius. Perhaps most importantly, users can define their own rules on various score contexts.

Students developed skills and knowledge in multiple areas in this algorithmic composition unit. Students engaged in practice-based research. By developing computational models of music, they practised computer programming and problem solving skills, as well as reflecting on how music is created, and judging the aesthetic quality of their musical results.

## Acknowledgments

Thanks to Örjan Sandred (University of Manitoba, Canada) for helpful feedback on a draft of this text. This research has been supported by the Rising Stars Programme research grant of the University of Bedfordshire.

## Bibliography

Ambrose, S.A., Bridges, M.W., DiPietro, M., Lovett, M.C., Norman, M.K. (2010) *How Learning Works: Seven Research-Based Principles for Smart Teaching*, San Francisco: Jossey Bass

Anders, T. (2014) 'Modelling Durational Accents for Computer-Aided Composition', in *Proceedings of the 9th Conference on Interdisciplinary Musicology – CIM14*, Berlin, Germany.

Anders, T. (2016) 'Compositions Created with Constraint Programming', in McLean, A. and Dean, R.T., eds., *The Oxford Handbook of Algorithmic Music*, Oxford: Oxford University Press.

Anders, T., Miranda, E.R. (2011) 'Constraint Programming Systems for Modeling Music Theories and Composition', *ACM Computing Surveys*, 43(4), 30:1–30:38.

Apt, K.R. (2003) *Principles of Constraint Programming*, Cambridge: Cambridge University Press.

Barrett, E., Bolt, B. (Eds.) (2010) *Practice as Research: Approaches to Creative Arts Enquiry*, London: I. B. Tauris.

Boehm, C. (2007) 'The discipline that never was: current developments in music technology in higher education in Britain', *Journal of Music, Technology and Education*, 1(1), 7–21.

Cunningham, M.G. (2007) *Technique for Composers*, Milton Keynes: AuthorHouse.

Fernández, J.D., Vico, F. (2013) 'AI Methods in Algorithmic Composition: A Comprehensive Survey', *Journal of Artificial Intelligence Research*, 48, 513–582.

Healey, M., Jenkins, A. (2009) *Developing Undergraduate Research and Inquiry*, Higher Education Academy.

Hewitt, M. (2009) *Composition for Computer Musicians*, Boston: Course Technology.

Hewitt, M. (2011) *Harmony for Computer Musicians*, Boston: Course Technology.

Hinton-Smith, T. (Ed.) (2012) *Widening Participation in Higher Education: Casting the Net Wide?*, Baskingstoke: Palgrave Macmillan.

Jenkins, T. (2002) 'On the difficulty of learning to program', in *Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences*, 53–58.

Jeppesen, K. (1939) *Counterpoint: The Polyphonic Vocal Style of the Sixteenth Century*, New York: Prentice-Hall.

Kreidler, J. (2009) *Loadbang: Programming Electronic Music in Pd*, Wolke Publishing House.

Kuuskankare, M., Laurson, M. (2006) 'Expressive notation package', *Computer Music Journal*, 30(4), 67–79.

Landy, L. (2007) *Understanding the Art of Sound Organization*, Cambridge, USA: MIT Press.

Laurson, M., Kuuskankare, M., Norilo, V. (2009) 'An Overview of PWGL, a Visual Programming Environment for Music', *Computer Music Journal*, 33(1), 19–31.

Lester, J. (1986) *The Rhythms of Tonal Music*, Carbondale: Southern Illinois University Press.

Michaelson, G. (2015) 'Teaching Programming with Computational and Informational Thinking', *Journal of Pedagogic Development*, 5(1), 51–66.

Moore, D. (2014) 'Supporting students in music technology higher education to learn computer programming', *Journal of Music, Technology and Education*, 7(1), 75–92.

Nierhaus, G. (2009) *Algorithmic Composition: Paradigms of Automated Music Generation*, New York: Springer.

Pachet, F., Roy, P. (2001) 'Musical Harmonization with Constraints: A Survey', *Constraints Journal*, 6(1), 7–19.

Piston, W. (1947) *Counterpoint*, New York: W. W. Norton.

Rameau, J.P. (1984) *Treatise on Harmony*, New York: Dover.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B. (2009) 'Scratch: programming for all', *Communications of the ACM*, 52(11), 60–67.

Rothstein, J. (1995) *MIDI: A Comprehensive Introduction*, 2nd ed, Madison, WI: A-R Editions, Inc.

van Roy, P., Haridi, S. (2004) *Concepts, Techniques, and Models of Computer Programming*, Cambridge, USA: MIT Press.

Russo, W., Ainis, J., Stevenson, D. (1988) *Composing Music: A New Approach*, University of Chicago Press.

Sandred, Ö. (2003) 'Searching for a Rhythmical Language', in *PRISMA 01*, Milano: EuresisEdizioni.

Sandred, Ö. (2010) 'PWMC, a Constraint-Solving System for Generating Music Scores', *Computer Music Journal*, 34(2), 8–24.

Schilingi, J.B. (2009) 'Local and Global Control in Computer-Aided Composition', *Contemporary Music Review*, 28(2), 181–191.

Schoenberg, A. (1967) *Fundamentals of Musical Composition*, London: Faber and Faber.

Schoenberg, A. (1983) *Theory of Harmony*, Berkeley: University of California Press.

Strijbos, J.-W., Narciss, S., Dünnebier, K. (2010) 'Peer feedback content and sender's competence level in academic writing revision tasks: Are they critical for feedback perceptions and efficiency?', *Learning and instruction*, 20(4), 291–303.

Wilson, S., Cottle, D., Collins, N. (2011) *The SuperCollider Book*, Cambridge, USA: MIT Press.