Computational Invention of Cadences and Chord Progressions by Conceptual Chord-Blending

Manfred Eppe[16], **Roberto Confalonieri**[1], **Ewen Maclean**[2], **Maximos Kaliakatsos**[3], **Emilios Cambouropoulos**[3], **Marco Schorlemmer**[1], **Mihai Codescu**[4], **Kai-Uwe Kühnberger**[5]

[1]IIIA-CSIC, Barcelona, Spain
{meppe,confalonieri,marco}@iiia.csic.es

[2]University of Edinburgh, UK
emaclea2@inf.ed.ac.uk

[3]University of Thessaloniki, Greece
{emilios,maxk}@mus.auth.gr

[4]University of Magdeburg, Germany
codescu@iws.cs.uni-magdeburg.de

[5] University of Osnabrück, Germany
kkuehnbe@uos.de

[6] ICSI, Berkeley, USA
eppe@icsi.berkeley.edu

Abstract

We present a computational framework for chord invention based on a cognitive-theoretic perspective on *conceptual blending*. The framework builds on algebraic specifications, and solves two musicological problems. It automatically finds transitions between chord progressions of different keys or idioms, and it substitutes chords in a chord progression by other chords of a similar function, as a means to create novel variations. The approach is demonstrated with several examples where jazz cadences are invented by blending chords in cadences from earlier idioms, and where novel chord progressions are generated by inventing transition chords.

1 Introduction

Suppose we live in a early diatonic tonal world, where dissonances in chords are mostly forbidden. We assume that, in this early harmonic space, some basic cadences have been established as salient harmonic functions around which the harmonic language of the idiom(s) has been developed — for instance, the perfect cadence, the half cadence, the plagal cadence and, even, older cadences such as the Phrygian cadence. The main question to be addressed in this paper is the following: *Is it possible for a computational system to invent novel cadences and chord progressions based on blending between more basic cadences and chord progressions?*

To answer this question, we describe cadences as simple pitch classes with reference to a tonal centre of C, and combine pitches of the semi-final chords of different cadences, assuming that the final chord is a common tonic chord. Additionally, we assign *priorities* to chord notes that reflect their relative prominence. Similarly, we assign priorities to the relative extensions of a chord, e.g., having a major third or a dominant seventh, which are independent from its root note.

Let us examine more closely the perfect and Phrygian cadences (see Fig. 1). Certain notes in their prefinal chords are more important as they have specific functions: In the perfect cadence, the third of the dominant seventh is the leading and most important note in this cadence, the root is the base of the chord and moves to the tonic, and the seventh resolves downwards by stepwise motion, whereas the fifth may be omitted. In the Phrygian cadence, the bass note (third of the chord) is the most important note as it plays the role of a downward leading note, and the second most important note is the root. In such a setup, we propose two applications of chord blending, to give rise to new cadences and chord progressions.

The first application is to generate a novel cadence as a 'fusion' of existing cadences by blending chords with a similar function. For example, in case of the perfect and the Phrygian, we blend their prefinal chords. Here, we start with combinations of at least three notes with the highest priority. Many of these combinations are not triadic or very dissonant and may be filtered out using a set of constraints. However, among those blends that remain, it turns out that the highest rating accepted blend (according to the priorities described above), is the tritone substitution progression (IIb7-I) of jazz harmony.

This simple blending mechanism 'invents' a chord progression that embodies some important characteristics of the Phrygian cadence (bass downward motion by semitone to the tonic) and the perfect cadence (resolution of tritone); the blending algorithm creates a new harmonic 'concept' that was actually introduced in jazz centuries later than the original input cadences. The backdoor progression also appears in the potential blends, but it embodies less characteristics from the inputs and is therefore considered a weaker blend (Fig. 1).

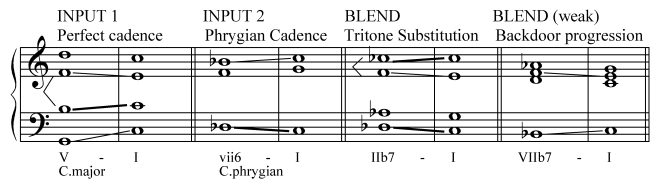

Figure 1: Conceptual blending between the perfect and Phrygian cadence gives rise to the tritone substitution progression and the backdoor progression

The second application of chord blending is to 'cross-fade' chord progressions of different keys or idioms in a smooth manner by means of a *transition chord*, which is the result of blending. Assume that the input sequences are precomposed by another system, e.g. the constrained HMM (cHMM) by [Kaliakatsos and Cambouropoulos, 2014]. Let us suppose that a chord sequence starts in C major, such as C-Dmin-G7-C-F, and after its ending an intermediate G♯7-C♯ chord progression is introduced (having thus a very remote modulation from C major to C♯ major). The cHMM system will not find any transition from the available diatonic chords in C major to the G♯7 chord, and will terminate or give a random continuation. However, if we perform blending on F ([5,9,0]) – the last chord from the first progression – with G♯7 ([8,0,3,6]) – the first chord from the last progression – then we get [0,3,6,9] which contains two notes from the first chord and three from the second.[1] This resultant chord is the diminished seventh chord that is well-known to be very versatile and useful for modulations to various keys. Hence, the blending mechanism 'invents' a new chord that bridges the two key regions.

In order to implement a computational framework that is capable of performing blends like the aforementioned examples, we build on the cognitive theory of conceptual blending by [Fauconnier and Turner, 2002]. We also take inspiration from the category-theoretical formalisation of blending by [Goguen, 1999] and use the category theoretical *colimit* operation to compute blends. Hence, we contribute to the ongoing discussion on musical computational creativity [Ramalho and Ganascia, 1994; Pachet, 2012; Wiggins *et al.*, 2009], where conceptual blending has been identified to be at the heart of music meaning and appreciation on formal, gestural, emotional and referential levels [Brandt, 2008].

A number of researchers in the field of computational creativity have recognised the value of conceptual blending for building creative systems, and particular implementations of this cognitive theory have been proposed [Veale and O'Donoghue, 2000; Pereira, 2007; Goguen and Harrell, 2010; Guhe *et al.*, 2011]. However, there is surprisingly little work on formalisations and computational systems that employ blending for music generation, and it is unclear how existing implementations of blending can deal with musicological concepts. Exceptions are [Pereira and Cardoso, 2007], who provide a systematic approach to create a novel chord by blending an existing chord with colour properties, and [Nichols *et al.*, 2009], who propose a weighted-sum combination of chord transition Markovian matrices from different musical styles to produce novel 'blended' ones. In both cases, a detailed computational implementation is not provided, and the application of chord blending to generate novel chord progressions has not been investigated. In this work, we take inspiration from [Kaliakatsos *et al.*, 2014], who use a simple informal cadence representation for blending, without providing a computational framework.

[1]Throughout this paper, we follow the usual notation of specifying notes as numbers which refer to semitones above a tonal centre. If not stated otherwise, we use an absolute tonal centre of C. However, sometimes we explicitly use the root of a chord as a relative tonal centre, as described in Sec. 2.

2 An Algebraic Model of Chords

For our blending framework, we follow Goguen's proposal to model conceptual spaces as algebraic specifications. Towards this, we use specifications defined in a variant of *Common Algebraic Specification Language* (CASL) [Mosses, 2004], which is extended with priority values associated to axioms.

Definition 1 (Prioritised CASL specification). *A prioritised CASL specification* $S = (\langle \mathcal{ST}, \lesssim_{\mathcal{ST}} \rangle, \mathcal{O}, \mathcal{P}, \langle \mathcal{A}, \leq_{\mathcal{A}} \rangle)$ *consists of a set* $\mathcal{ST}$ *of sorts along with a preorder* $\lesssim_{\mathcal{ST}}$ *that defines a sub-sort relationship, a set* $\mathcal{O}$ *of operators that map objects of argument sorts to the respective domain sort, a set* $\mathcal{P}$ *of predicates that map objects to Boolean values, and a set of axioms* $\mathcal{A}$ *with a partial priority order* $\leq_{\mathcal{A}}$.

We say that two prioritised CASL specifications are equal, if their sorts, operators, predicates, axioms, as well as subsort-relationships are equal. Note that this notion of equality does not involve priority ordering of axioms. As notational convention, we use superscript $\mathcal{A}^S$ to denote the set of axioms of a particular specification S. CASL lets us define our musical theory about chords and notes in a modular way that facilitates the definition of specifications with inheritance relations between them as follows:

Symbols is the most basic specification that contains the building blocks to describe notes and chord features. The sort $Note$ is constituted by numbers from 0 to 11, describing their position in a scale. This can be a relative or an absolute position. For example, 7 can refer to a G note in a C major tonality, or to the relative interval of a perfect fifth (7 semitones above the tonality's or chord's root).

RelChord inherits from $Symbols$ and contains the operators needed to define a chord of the sort $RelChord$, which has no absolute root. We use the predicate $relNote : RelChord \times Note$ to assign a relative note to a chord. For example $relNote(c, 7)$ means that the chord c has a note which is seven semitones above the root, i.e., a perfect fifth.

AbsRelChord extends $RelChord$ in that it provides the sort $AbsRelChord$, a subsort of chord specifications that have also absolute notes, in addition to the relative notes inherited from $RelChord$. Absolute notes are defined with the predicate $absNote : AbsRelChord \times Note$. We use the usual absolute tonal centre of C, so that e.g., a G7 chord can be specified by the absolute notes [7,11,2,5]. The $AbsRelChord$ specification also involves an operator $root : AbsRelChord \rightarrow Note$ to fix the root note of a chord, and a '+' operator that we use for arithmetics of addition in a cyclic group of 12 semitones. For example, 7+7=2 denotes that a fifth on top of a fifth is a major second. This allows us to define the relation between relative and absolute notes of a chord as follows:

$$\forall n : Note; c : AbsRelChord.\ relNote(c, n) \iff absNote(c, n + root(c)) \qquad (1)$$

For example, the relative notes [0,4,7,10] determine a relative dominant seventh chord. Setting the root to 7 makes that chord a G7, and Axiom (1) allows us to deduce the absolute notes [7,11,2,5] by adding the root to each relative note. This corresponds to the following prioritised CASL specification:

spec G7InPerfect = AbsRelChord **then**

$$\begin{array}{lll} \textit{op } c : \textit{AbsRelChord} & .root(c) = 7 & \\ .absNote(c,7) \ \text{\%p(2)\%} & .relNote(c,0) \ \text{\%p(3)\%} & \\ .absNote(c,11) \ \text{\%p(3)\%} & .relNote(c,4) \ \text{\%p(3)\%} & (2) \\ .absNote(c,2) \ \text{\%p(1)\%} & .relNote(c,7) \ \text{\%p(2)\%} & \\ .absNote(c,5) \ \text{\%p(2)\%} & .relNote(c,10) \ \text{\%p(3)\%} & \end{array}$$

end

The priorities are assigned as numbers using the CASL annotation *p*. Note that we attribute importance to the notes within a chord in two ways. Firstly, we attach priorities to notes relative to a key, and hence their function within that key, by prioritising axioms involving $absNote$ predicates. For example, given that G7 is the prefinal chord of a perfect cadence resolving in C major, then the function of the major third of this chord is vital because it provides a leading note – a semitone below the tonic. Thus, the fact $absNote(c, 11)$ is given a high priority with *%p(3)%*. Secondly, we attach priorities to notes relative to the chord root, and hence their function within the chord, by prioritising axioms involving $relNote$. For example, we usually want to emphasise the importance of the chord having a dominant seventh, denoted by $relNote(c, 10)$ *%p(3)%*, whereas the fifths has a lower importance.

A challenging problem in blending is the huge number of possible combinations of input specifications. Towards this, we constrain the search space and disallow dissonant chords by means of the following axioms:

$$\forall c : AbsRelChord.\ \neg(relNote(c,3) \wedge relNote(c,4)) \qquad (3a)$$

$$\forall c : AbsRelChord.\ \neg relNote(c,1) \qquad (3b)$$

$$\forall c : AbsRelChord.\ \neg(relNote(c,6) \wedge relNote(c,7)) \qquad (3c)$$

The axioms prohibit chords with one semitone below the major third or one semitone above the minor third (3a), one semitone above the root (3b), and one semitone below the perfect fifth or one semitone above the diminished fifth (3c). These constraints work well for our examples depicted in Sec. 4, but they can of course be extended or relaxed if desired.

3 Computational Chord Blending

A computational chord blending framework should be able to deal with three problems. First, it has to avoid *dissonances* that a naive combination of two chords can produce; second, it has to respect that some elements in the input chords are more *salient* then others; and third, it has to deal with the *huge space* of possible blends. In the implementation of this framework, we employ the core ideas of the notion of *Amalgams* from the field of case based reasoning [Ontañón and Plaza, 2010] to deal with these problems. Specifically, we employ a search process that interleaves the combination of chord specifications with a step-wise generalisation process that removes notes which cause an inconsistency.

Our framework (Fig. 2) first generalises chords by removing their least salient notes. Then, the generalised chords are combined via the colimit operation on CASL theories [Mossakowski, 1998]. After this, it completes the blends by means of the GCT algorithm [Cambouropoulos *et al.*, 2014] and a deduction step, and, finally, it performs a consistency check to ensure that the result is not too dissonant. If the produced blend is consistent, then it is evaluated by respecting (i) the total number of axioms removed from a chord specification (removing less axioms gives a better blend because more information is preserved) (ii) the importance of axioms removed from a chord specification (removing axioms of low importance gives a better blend because *salient* information is preserved), and (iii) the *balance* of the amount of generalisation of the chord specifications. The latter point (iii) is important, because we do not want blends where one chord is generalised a lot, by removing many axioms, and another chord only very little, by removing none or very few axioms. Instead, we want to keep a balanced amount of information from each input space. This behaviour refers to the *double-scope* property of blends, that is advocated in [Fauconnier and Turner, 2002] as "what we typically find in scientific, artistic and literary discoveries and inventions." Points (i) and (ii) account for many of the *optimality principles* proposed in [Fauconnier and Turner, 1998; 2002] to ensure 'good' or 'interesting' blends.

Generalisation.

Generalisation of chords is not only essential to resolve inconsistencies, but also required to identify commonalities between the input chords. In blending literature, a conceptual space that contains only commonalities between two input spaces is called *generic space* — a constitutional element of a conceptual blending process [Fauconnier and Turner, 2002]. In our framework, it is required to perform the *colimit* operation on chord specifications. Therefore, we employ a search process to find the generic space, by removing axioms (i.e., notes) from chord specifications until only the common notes between the chords are left. The formal definition of the generic space of input chord specifications is:

Definition 2 (Generic space). *Given a set* $\mathbf{S}$ *of chord specifications, we say that a specification G is a generic space of* $\mathbf{S}$*, if* $\forall S \in \mathbf{S} : G \subseteq S$*, where* $G \subseteq S$ *denotes that axioms, operators, predicates, sorts and partial orders in G are subsets of (or equal to) their respective counterparts in S.*

As an example consider the prefinal chords of the perfect and the Phrygian cadence (G7 and B♭min). They have one absolute note (5) and two relative notes (root and fifth) in common, which are represented as the following generic space:

$$absNote(\text{c},5) \quad relNote(\text{c},0) \quad relNote(\text{c},7) \qquad (4)$$

We realise the generalisation of chords by successive application of *generalisation operators*, which remove axioms from a chord specification. For each axiom $ax \in \mathcal{A}^S$ of a chord specification S there exists one generalisation operator $\sigma(ax)$ that removes the axiom. The formal definition is:

Definition 3 (Generalisation operator). *A generalisation operator* $\sigma(ax)$ *is a partial function that maps a chord specification to a chord specification. The application of a generalisation operator* $\sigma(ax)$ *on a specification* S *is defined as*

$$\sigma(ax)(S) = \begin{cases} S \setminus \{ax\} & \textit{if } \exists S' : ax \notin S' \\ \textit{undefined} & \textit{otherwise} \end{cases} \qquad (5)$$

where $S \setminus ax$ *denotes the removal of an axiom* ax *from the poset of axioms* $\mathcal{A}^S$ *of* S*, and* S' *denotes another chord specification that is input to the blending process.*

The conditional statement in Eq. (5) only allows the removal of an axiom if there exists a chord specification S' that does not involve the axiom, i.e., if the axiom is not common among

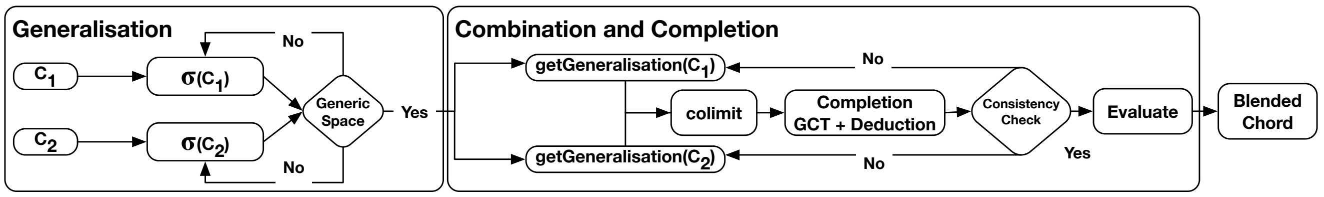

Figure 2: Blending as interleaved generalisation, combination and completion process

all input chord specification. This assures that the resulting generic space is a *least general generalisation*, where all commonalities between the input specifications are kept.

The successive application of generalisation operators on a chord specification forms a *generalisation path* from the input specification to the generic space. Hence, in order to find a generic space between several input chords, we search for one generalisation path for each input specification. A generalisation path is defined as follows:

Definition 4 (Generalisation path). *Let S be a prioritised CASL specification and $\sigma_1, \dots, \sigma_n$ be generalisation operators. We denote a generalisation path as $p = \sigma_1; \dots; \sigma_n$ and write $p(S) = \sigma_n(\cdots \sigma_2(\sigma_1(S)) \cdots)$ to denote the successive application of the generalisation operators in p on S.*

As an example, consider the path (6) which leads from the G7 specification (2) to the generic space (4).

$$\begin{aligned}&\sigma(\mathit{absNote}(c, \mathit{2})); \sigma(\mathit{absNote}(c, \mathit{7})); \sigma(\mathit{relNote}(c, \mathit{10}));\\ &\sigma(\mathit{relNote}(c, \mathit{4})); \sigma(\mathit{absNote}(c, \mathit{11}));\end{aligned} \tag{6}$$

A general problem is the huge search space of possible generalisation paths. To avoid this, we exploit axiom priorities, and only allow a limited number k of operators that violate the priority order of axioms within a path, i.e., that remove a high-priority axiom before a lower-priority axiom is removed. For example, path (6) does not violate the priority order, because axioms with a low priority are always removed before axioms with a higher priority. However, in the case of the backdoor blend described in Sec.1, the early removal of axiom $\mathit{absNote}(c, \mathit{11})$ violates the priority order. For most of our examples, a value of $k = 2$ turned out to be useful.

For the generalisation process (Fig. 2), two chords C_1 and C_2 are given as input to the system, and successively generalised until the generalisation operators form a generalisation path to the generic space. Note that it is possible to have generalisation paths of different length. For example, three generalisation operators may have to be applied on one input space, while only two generalisations are required for the other input space. Once a pair of generalisation paths is found, they are handed over to a combination and completion process.

Combination and completion.

While a full generalisation path generalises an input chord specification towards the generic space, we want to keep those specifics of each input specification that do not cause inconsistencies in the blend. For example, blending the prefinal G7 chord of the perfect cadence with the prefinal B♭min chord of the Phrygian cadence results in a D♭7 chord, which is the prefinal of the tritone substitution progression. However, this requires to generalise the G7 chord by removing its absolute 2 note, because otherwise one would end up with too much dissonance that arises in combination with the root note 1 of the resulting D♭7. However, we must not generalise all the way down to the generic space, because, for example, the absolute 11 in the G7 is very salient and should be kept.

Towards this, we introduce the notion of a *prefix* of a generalisation path, which generalises an input chord only as much as necessary to avoid inconsistencies, thereby keeping as many salient notes as possible. Formally, a prefix is the subsequence of the first m generalisation operators of a generalisation path.

Definition 5 (Generalisation path prefix). *Given a generalisation path $p = \sigma_1; \cdots; \sigma_n$, then $p^{pre} = \sigma_1; \cdots; \sigma_m$ is a prefix of p iff $m \leq n$.*

For example, it turns out that for blending G7 and B♭min, the generalisation path prefix of G7, which is required to remove all dissonant notes, is $\sigma(\mathit{absNote}(c, \mathit{2})); \sigma(\mathit{absNote}(c, \mathit{7}))$.

The combination and completion process depicted in Fig. 2 selects generalisation path prefixes for each input via *getGeneralisation*, and applies them to the input chords. The result is a generalised version of each input chord, used to compute candidate blends. The process starts with empty generalisation path prefixes, and increases the amount of generalisation in each iteration, until a consistent blend is found. The amount of generalisation is measured as a *generalisation cost*, that considers (i) the total amount of generalisation of each input space, (ii) an additional penalty for paths where the priority order among axioms is not preserved, and (iii) the balance between the amount of generalisation for each input specification (recall the *double-scope* property mentioned in [Fauconnier and Turner, 2002]). This is determined by the following functions:

$$\begin{aligned}&\mathit{cost}(p) = |p| + |\{\sigma \in p \mid\\ &\qquad\qquad\sigma \text{ violates the priority order among axioms}\}|\\ &\mathit{totalCost}(p_1, p_2) = \max(\mathit{cost}(p_1), \mathit{cost}(p_2))^2 +\\ &\qquad\qquad\min(\mathit{cost}(p_1), \mathit{cost}(p_2))\end{aligned} \tag{7}$$

First, $\mathit{cost}(p)$ determines the generalisation cost of one generalisation path prefix. This is realised as the sum of the length $|p|$ of the path, and the number of generalisation operators σ within p that violate the priority ordering among axioms. The priority order is violated if an axiom with a higher priority value is removed before an axiom with a lower priority in the same path. Second, $\mathit{totalCost}$ determines the total cost of both generalisation path prefixes together. This is defined as the *square* of the higher generalisation cost of the two paths, plus the lower generalisation cost. Using the square causes a lower total cost for a pair of paths which have a similar generalisation cost, compared to a pair of paths where the amount of generalisation is unbalanced. It therefore promotes blends with the desired *double-scope* property.

Having selected generalisation prefixes p_1^{pre}, p_2^{pre} with a certain total generalisation cost for two input chords C_1, C_2, we obtain two generalised chord specifications by applying the prefixes to the chords as described in Def. 4. The generalised chord specifications and the generic space are then input to the colimit. Indeed, the colimit operation coincides with what [Fauconnier and Turner, 2002] refer to as the *composition* step of blending, i.e., a 'raw' candidate combination of information from the input spaces. According to Fauconnier and Turner, the composition is then subject to a *completion* and an *elaboration* step that enrich the composition with background knowledge. In our framework, we complete the blend as follows: First, we analyse the set of absolute notes to determine the root of a chord using the GCT algorithm [Cambouropoulos *et al.*, 2014]. Second, we deduce additional information about absolute and relative notes via axiom (1). For example, in case of blending G7 with B♭min, GCT analyses the absolute notes [1,5,11] of the colimit, and infers that 1 is the root of the resulting D♭7 chord. With the information about the root, additional information about the relative notes is used to deduce additional absolute notes and vice-versa, e.g., we deduce that the D♭7 chord should also have a relative fifth.

Once the completion step is done, we check consistency of the blend. If the blend is consistent, then we evaluate it by considering the generalisation cost. The lower the total generalisation cost of the path prefixes according to $totalCost(p_1^{pre}, p_2^{pre})$ (7), the better the blend. After evaluation, the blend is output as a potential solution.

Implementation.

The described blending system is implemented using the Stable Model Semantics of Answer Set Programming (ASP) [Gelfond and Lifschitz, 1988], a well-known declarative programming paradigm to solve non-monotonic search problems. In particular, ASP facilitates the implementation of the unique nondeterministic choice of generalisation operators in the generalisation part, and the unique nondeterministic selection of prefixes in the combination part of our system, by using so-called choice rules (see e.g. [Gebser *et al.*, 2012]). We use the ASP solver *clingo v4* [Gebser *et al.*, 2014] as main reasoning engine, which allows us not only to implement the search in an incremental manner, but also to use external programs via a Python interface. In our case, we need Python to call HETS [Mossakowski, 1998] as an external tool for computing the colimits for CASL specifications, and to invoke the theorem provers *darwin* [Baumgartner *et al.*, 2007] and *eprover* [Schulz, 2013] for the consistency check.

4 Chord Blending at Work

To validate our approach, we present various examples of our system at work. This is summarised in Tables 1 and 2, where we provide the input chord progressions, the chords that are blended, the prefixes,[2] the colimit, the completion, and the resulting blend with its respective total generalisation cost. The two tables refer to the two different applications that we envisage for chord blending. We refer to these applications as *cadence fusion* and *cross-fading*.

[2]Recall that prefixes denote the notes that are *removed* from the input chords. 'Prefix1' refers to the blended chord in the upper line of the 'Inputs' column, and 'Prefix2' to the chord in the lower line.

Cadence fusion. This application takes two chord sequences as input and blends chords of a similar function, which results in a 'fusion' of both input chord sequences. In this paper we investigate the special case of cadences and present five corresponding examples. For brevity we generalise the final chords in the cadences to C chords. As discussed in Sec.1, we assign a high priority to the absolute 11 note of the G7 of the perfect cadence, and to the absolute 1 note of the prefinal B♭min in the Phrygian cadence. For the plagal cadence, we put a high priority on the absolute 5 and 9 notes of its prefinal chord, since these cause its characteristic suspended feel.

▷ *Tritone progression.* This refers to the running example described throughout the paper. We blend the G7 of the perfect and the B♭min of the Phrygian cadence to obtain a D♭7 as prefinal chord of the tritone substitution progression.

▷ *Backdoor cadence.* Like the tritone, this result is also obtained by blending the prefinal chord of the perfect and the Phrygian cadence. However, this is a weaker blend with a higher generalisation cost, because the generalisation prefixes violate the priority order of axioms. For example, $absNote(c, 11)$ is removed from the G7 of the perfect cadence, even though it has a high priority.

▷ *Diatonic extensions.* These are variations of generating jazz-type chords that are obtained by blending the prefinal chords of the plagal and the perfect cadence. The notes from the plagal cadence are used as 9th and 11th extensions to the prefinal G7 of the perfect cadence.

▷ *Modified Phrygian.* Here we blend the prefinal chords of the plagal and the Phrygian cadence. The result is an interesting modification of the Phrygian, with a prefinal B♭minmaj9.

Cross-fading. The second application of chord blending concatenates two chord sequences to a single chord sequence by blending the last chord of the first sequence with the first chord of the last sequence to obtain a transition chord. The blended chord then serves as a transition chord that is used to 'cross-fade' the two chord progressions in a smooth manner. Examples for this application are depicted in Table 2. We used the first example for the development of our system. The last four examples are taken from the *Real Book* of jazz [Leonard, 2004]. They are of particular interest because they allow us to evaluate our system. Towards this, we take chord sequences with a key transition from the book, and blend the last chord from the first transition with the first chord from the last transition. Then we compare the resulting blended chord with the actual chord that is found in the book. If the chord is the same or similar, we consider the approach to be successful.

As far as the priorities in the examples are concerned, we give those absolute notes with a specific function in the key a high priority. In particular, the roots, and thirds within a key are usually causing certain characteristics that are important. It is of similar importance when a chord has a characteristic extension, such as a dom7. Hence, such relative notes (10 in the case of dom7) are also given a high priority.

▷ *Development example.* This refers to the example described in Sec. 1, where the F chord is blended with G♯7 chord to obtain the C°7 chord as transition between the keys.

▷ *All the things you are.* This has a chord progression F♯min7 - B7 - Emaj7 - C+7 - Fmin7 - B♭ - E♭ and hence a key transition from E major to E♭ major with C+7 as transi-

	Inputs	Prefix1	Prefix2	Colimit	Completion	Blend
Perfect/Phrygian **Tritone**	G7 ⟶ C B♭min → C	$\sigma(abs(2))$; $\sigma(abs(7))$	$\sigma(abs(10))$; $\sigma(rel(3))$	D♭7 without perf.5th	root D♭ A♭ as perf.5th	D♭7 → C (total cost = 6)
Perfect/Phrygian **Backdoor**	G7 ⟶ C B♭min → C	$\sigma(abs(7))$; $\sigma(abs(11))$	$\sigma(abs(1))$; $\sigma(rel(3))$	B♭7	root B♭ A♭ as dom.7th	B♭7 → C (total cost = 19)
Perfect/Plagal **Diatonic extension**	G7 ⟶ C F ⟶ C	$\emptyset$	$\emptyset$	G11	root G	G11 → C (total cost = 0)
Perfect/Plagal **Diatonic extension**	G7 ⟶ C F ⟶ C	$\emptyset$	$\sigma(abs(0))$; $\sigma(rel(7))$	G9	root G	G9 → C (total cost = 4)
Phrygian/Plagal **Modified Phrygian**	B♭min ⟶ C F ⟶ C	$\emptyset$	$\sigma(rel(4))$	B♭min	root B♭ A as maj.7th C as 9th	B♭minmaj9 → C (total cost = 4)

Table 1: Cadence fusion results generated by our system

	Inputs	Prefix1	Prefix2	Colimit	Completion	Blend
Dev. Example	C ⟶ F G♯7 → C♯	$\sigma(abs(5))$; $\sigma(rel(7))$	$\sigma(abs(8))$; $\sigma(rel(10))$; $\sigma(rel(4))$; $\sigma(rel(7))$	single C note	root C E♭ as min.3rd G♭ dim.5h A as dim.7th	C → C^{o}7 → C♯ (total cost = 19)
All the things	B7 ⟶ Emaj7 Fmin7 ⟶ B♭	$\sigma(rel(11))$	$\sigma(abs(5))$; $\sigma(rel(3))$; $\sigma(rel(7))$	C7♯5♯9 without 7th	root C B♭ as 7th	Emaj7 → C7♯5♯9 → Fmin7 (total cost = 26)
Blue Bossa	Cmin E♭min	$\sigma(abs(7))$; $\sigma(rel(7))$	$\sigma(abs(1))$	Cmin without 5th	root C F♯ as dim.5th B♭ as min.7th	Cmin → C$^{\emptyset 7}$ → E♭min (total cost = 5)
Con alma	C♯min ⟶ B7 E♭maj7 → E♭min7	$\sigma(abs(6))$; $\sigma(abs(9))$; $\sigma(abs(11))$	$\sigma(abs(2))$; $\sigma(rel(11))$; $\sigma(abs(7))$	B♭11 without maj.3rd without perf.5th without 7th	root B♭ D as maj.3rd F as perf.5th A♭ as 7th	B7 → B♭11 → E♭maj7 (total cost = 20)
Days Nights	F7 ⟶ B♭maj7 F♯min7 → Bmin7	$\sigma(abs(5))$; $\sigma(rel(11))$; $\sigma(abs(10))$; $\sigma(abs(2))$	$\sigma(abs(6))$; $\sigma(rel(3))$	A	root A G as min.7th	B♭maj7 → A7 → F♯min7 (total cost = 39)

Table 2: Cross-fading results generated by our system

tion chord. To evaluate our system, we try if our system would generate the C+7 transition chord automatically by blending Emaj7 with Fmin7. Our result is C7♯5♯9, which in fact is quite similar to the original C+7.

▷ *Blue Bossa.* It contains a progression Cmin-X-E♭min, moving from C minor to E♭ minor key without explicitly giving a transition chord (denoted by 'X'). Our resulting blend is a C$^{\emptyset 7}$, which one can safely assume as a natural transition chord chosen by an accompanist.

▷ *Con Alma.* This contains a progression C♯min - B7 - B♭7 - E♭7 and is (arguably) moving from a C♯min to an E♭ major key via the B♭7. Our system generates a B♭11 as transition chord, which is very close to the original B♭7.

▷ *Days and Nights Waiting.* This contains a progression B♭maj7 - A7 - F♯min7, i.e. it is moving from B♭ major to a D major key via the A7 as transition. Blending B♭maj7 and F♯min7 indeed gives us an A7.

5 Conclusion

The paper presents a blending-based approach to generate novel chord progressions and cadences. Though other blending frameworks, such as [Goguen and Harrell, 2010; Pereira, 2007; Guhe *et al.*, 2011; Veale and O'Donoghue, 2000] are in principle expressive enough to deal with basic chord specifications, they do not provide a formal model for this, and it is also unclear how their implementations would resolve inconsistencies. Our evaluation shows that the results of our framework are musicologically useful, in terms of inventing jazz cadences from earlier ones, and in terms of finding transition chords to 'cross-fade' chord sequences. We are not aware of any other approach that provides a full computational framework for this. Our work is based on the cognitive theory of conceptual blending [Fauconnier and Turner, 2002], and the category theoretical formalisation by [Goguen, 1999], in that we use algebraic specifications and combine chords via the colimit. Though one could think of simpler methods than the colimit for the naive combination of chords, we appreciate the generality of our approach: Firstly, it allows us to extend our system in future work, such that blending can happen directly on the level of cadences and chord progressions, or specifications of other musical entities, instead of blending only chords. Secondly, we can use it for the blending of input specifications with different algebraic signatures, which makes it possible to blend non-musicological and musicological concepts (e.g. [Zbikowski, 2002; Antović, 2011]). Such applications would involve a prioritisation for operators and predicates of the algebraic input language, and the introduction of generalisation and renaming operators for operators and predicates, so that the full potential of [Goguen, 1999]'s ideas of blending general sign-systems can be explored.

Acknowledgments

This work is supported by the 7th Framework Programme for Research of the European Commission funded COINVENT project (FET-Open grant number: 611553). The authors thank Enric Plaza for his inputs to the Amalgams part.

References

[Antović, 2011] M. Antović. Musical metaphor revisited: Primitives, universals and conceptual blending. *Stockholm Metaphor Festival*, 2011.

[Baumgartner *et al.*, 2007] P. Baumgartner, A. Fuchs, H. de Nivelle, and C. Tinelli. Computing finite models by reduction to function-free clause logic. *Journal of Applied Logic*, 2007.

[Brandt, 2008] P. Brandt. Music and the abstract mind. *Journal of Music and Meaning*, 7:1–15, 2008.

[Cambouropoulos *et al.*, 2014] E. Cambouropoulos, M. Kaliakatsos, and C. Tsougras. An idiom-independent representation of chords for computational music analysis and generation. In *Proceeding of the International Computer Music Conference*, 2014.

[Fauconnier and Turner, 1998] G. Fauconnier and M. Turner. Principles of conceptual integration. In J. P. Koenig, editor, *Discourse and Cognition: Bridging the Gap*, pages 269–283. Center for the Study of Language and Information, 1998.

[Fauconnier and Turner, 2002] G. Fauconnier and M. Turner. *The Way We Think: Conceptual Blending And The Mind's Hidden Complexities*. Basic Books, 2002.

[Gebser *et al.*, 2012] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub. *Answer Set Solving in Practice*. Morgan and Claypool, 2012.

[Gebser *et al.*, 2014] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub. Clingo = ASP + control: Preliminary report. *CoRR*, abs/1405.3694, 2014.

[Gelfond and Lifschitz, 1988] M. Gelfond and V. Lifschitz. The Stable Model Semantics for Logic Programming. In *Proceedings of the International Conference on Logic Programming*, 1988.

[Goguen and Harrell, 2010] J. A. Goguen and D. F. Harrell. Style: A computational and conceptual blending-based approach. In S. Argamon, K. Burns, and S. Dubnov, editors, *The Structure of Style: Algorithmic Approaches to Understanding Manner and Meaning*, pages 291–316. Springer, 2010.

[Goguen, 1999] J. A. Goguen. An introduction to algebraic semiotics, with application to user interface design. In C. L. Nehaniv, editor, *Computation for Metaphors, Analogy, and Agents*, volume 1562 of *Lecture Notes in Computer Science*, pages 242–291. 1999.

[Guhe *et al.*, 2011] M. Guhe, A. Pease, A. Smaill, M. Martínez, M. Schmidt, H. Gust, K. U. Kühnberger, and U.Krumnack. A computational account of conceptual blending in basic mathematics. *Cognitive Systems Research*, 12(3–4):249–265, 2011.

[Kaliakatsos and Cambouropoulos, 2014] M. Kaliakatsos and E. Cambouropoulos. Probabilistic harmonisation with fixed intermediate chord constraints. In *Proceeding of the International Computer Music Conference*, 2014.

[Kaliakatsos *et al.*, 2014] M. Kaliakatsos, E. Cambouropoulos, K. U. Kühnberger, O. Kutz, and A. Smaill. Concept invention and music: Creating novel harmonies via conceptual blending. *Proceedings of the Conference on Interdisciplinary Musicology*, 2014.

[Leonard, 2004] H. Leonard. *The Real Book*. Hal Leonard Corporation, 2004.

[Mossakowski, 1998] T. Mossakowski. Colimits of order-sorted specifications. In *Recent trends in algebraic development techniques*, volume 1376 of *Lecture Notes in Computer Science*, pages 316–332. Springer, Berlin, 1998.

[Mosses, 2004] P. D. Mosses. *CASL Reference Manual – The Complete Documentation of the Common Algebraic Specification Language*, volume 2960 of *Lecture Notes in Computer Science*. Springer, 2004.

[Nichols *et al.*, 2009] E. Nichols, D. Morris, and S. Basu. Data-driven exploration of musical chord sequences. In *Proceedings of the International Conference on Intelligent User Interfaces*, pages 227–236, 2009.

[Ontañón and Plaza, 2010] S. Ontañón and E. Plaza. Amalgams: A formal approach for combining multiple case solutions. In I. Bichindaritz and S. Montani, editors, *Proceedings of the International Conference on Case Base Reasoning*, volume 6176 of *Lecture Notes in Computer Science*, pages 257–271. Springer, 2010.

[Pachet, 2012] F. Pachet. Musical virtuosity and creativity. In *Computers and Creativity*, pages 115–146. Springer, 2012.

[Pereira and Cardoso, 2007] F. Pereira and F. Cardoso. Knowledge Integration with Conceptual Blending. In *Proceedings of the Irish Conference on Artificial Intelligence & Cognitive Science*, 2007.

[Pereira, 2007] F. Pereira. *Creativity and Artificial Intelligence: A Conceptual Blending Approach*, volume 4 of *Applications of Cognitive Linguistics*. Mouton de Bruyter, 2007.

[Ramalho and Ganascia, 1994] G. Ramalho and J. G. Ganascia. Simulating creativity in jazz performance. In *Prodeedings of AAAI*, volume 94, pages 108–113, 1994.

[Schulz, 2013] S. Schulz. System Description: E 1.8. In Ken McMillan, Aart Middeldorp, and Andrei Voronkov, editors, *Proceedings of LPAR*, volume 8312 of *LNCS*. Springer, 2013.

[Veale and O'Donoghue, 2000] T. Veale and D. O'Donoghue. Computation and blending. *Cognitive Linguistics*, 11(3/4):253–281, 2000.

[Wiggins *et al.*, 2009] G. Wiggins, M. Pearce, and D. Müllensiefen. Computational modeling of music cognition and musical creativity. In *The Oxford Handbook of Computer Music*. Oxford University Press, 2009.

[Zbikowski, 2002] L. Zbikowski. *Conceptualizing Music: Cognitive Structure, Theory, and Analysis*. Oxford University Press, 2002.