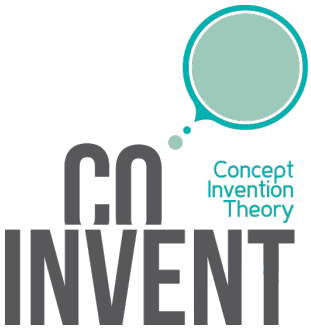


D1.2
Documentation of Examples for Working Domains

Editors	Tarek R. Besold, Danny Arlen de Jesus Gómez-Ramírez, Kai-Uwe Kühnberger, Nico Potyka
Reviewers	Marco Schorlemmer

Grant agreement no.	611553
Project acronym	COINVENT - Concept Invention Theory
Date	April 1, 2016
Distribution	PU

Disclaimer

The project COINVENT 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-Open Grant number 611553.

Abstract

This deliverable documents several rather hard examples from the mathematics and music domain with respect to concept invention by blending conceptual spaces. In total, three papers are concerned with concept invention in mathematics ranging from such diverse concepts like prime ideals and Dedekind Domains to Galois Theory and the concept of infinity in mathematics. One paper is dedicated to the music domain and discusses several hard examples of computing new chord cadences and chord progressions in harmonization contexts, like the invention of a Backdoor progression based on a perfect cadence and a Phrygian cadence as input spaces. The deliverable is based on three published articles and one paper which is currently under review. The covered concepts are rather advanced examples of concept invention by conceptual blending and are documented in a formal and detailed way.

Keyword list: **Examples of Conceptual Blending**, **Mathematics**, **Music**

Executive Summary

This deliverable covers detailed examples of conceptual blending in the domains of mathematics and music. It is based on the following four articles:

- F. Bou, M. Schorlemmer, J. Corneli, D. Gomez-Ramirez, E. Maclean, A. Smaill, and A. Pease (2015): The Role of Blending in Mathematical Invention. Proceedings of the 6th International Conference on Computational Creativity (ICCC) 2015.
- D. Gomez-Ramirez (2015): Conceptual Blending as a creative meta-generator of mathematical concepts: Prime Ideals and Dedekind Domains as Blend. Proceedings of the 4th International Workshop on Computational Creativity, Concept Invention, and General Intelligence (C3GI) 2015, Publications of the Institute of Cognitive Science (PICS), Vol. 02-2015, 2015.
- D. Gomez-Ramirez (under review): Generating Fundamental Notions of Fields and Galois Theory through Formal Conceptual Blending. Submitted to Journal of Logical and Algebraic Methods in Programming.
- M. Eppe, R. Confalonieri, E. Maclean, M. Kaliakatsos-Papakostas, E. Cambouropoulos, M. Schorlemmer, M. Codescu, and K.-U. Kühnberger: Computational Invention of Cadences and Chord Progressions by Conceptual Chord-Blending. Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI) 2015.

The first paper "The Role of Blending in Mathematical Invention" covers several examples of concept invention in mathematics. In particular, examples are considered such as the reconstruction of a partial axiomatization of the integers as a formal blend of a partial axiomatization of the natural numbers using the successor function and the notion of an inverse function, the construction of prime ideals as a blend of ideals on a commutative ring with unity and the concept of prime number, and the construction of infinity as a blend of the concept of iterative process and completed iterative process.

The second paper "Conceptual Blending as a creative meta-generator of mathematical concepts: Prime Ideals and Dedekind Domains as Blend" introduces in detail using a CASL specification[1] conceptual blending as a meta-generator. The detailed and technically spelled-out example is the characterization of prime ideals over Dedekind Domains as a blend of the space of ideals over a Noetherian ring and the space of prime numbers.

The third paper "Generating Fundamental Notions of Fields and Galois Theory through Formal Conceptual Blending" is currently under review and describes in detail an iterative blending process that eventually results in fundamental mathematical concepts like fields and Galois Theory. The iterated blending process starts with five basic structural concepts. Conceptual blending

[1]For more information concerning the specification language CASL compare Bidoit, M. and Mosses, P.D.: CASL User Manual. Lecture Note in Computer Science 2900, Springer Verlag Berlin Heidelberg New York (2004)

can be seen as a meta-generator of concept invention in mathematics from a theoretical and computational point of view.

The last paper "Computational Invention of Cadences and Chord Progressions by Conceptual Chord-blending" specifies a variety of innovative chord progressions in the domain of musical harmonization. It covers the blending of new cadences like the well-known Tritone substitution or Backdoor progression (often used in Jazz music) as well as cross-fading blends where, for example, a progression C $\rightarrow$ F and G#7 $\rightarrow$ C# is blended to a chord progression C $\rightarrow$ $C^0 7$ $\rightarrow$ C# by blending the right chord of the first input with the left chord of the second input. The result is an interesting modulation from C to C# via a diminished minor seventh chord $C^0 7$.

All documented examples covered in the four papers can be considered as hard problems of concept blending, because of the complexity of the involved conceptual spaces, the degree of abstractness of the involved concepts, and the sometimes difficult blending processes that are necessary in order to compute an interesting blend space.

Contents

1 Introduction

This deliverable comprises four research articles concerned with concept invention by conceptual blending in the domains of mathematics and music. Three articles are dedicated to the mathematics domain and one article is concerned with music, more precisely with inventing new chord progressions by blending constructions. In the project, more articles on computing inventions in the mathematics and music domain have been written that could be used as documentations of examples. Examples of further papers with detailed formal specifications are

- M. Martinez, A. Abdel-Fattah, U. Krumnack, D. Gómez-Ramírez, A. Smaill, T. Besold, A. Pease, M. Schmidt, M. Guhe, K.-U. Kühnberger (2016): Theory Blending: Extended Algorithmic Aspects and Examples. Annals of Mathematics and Artificial Intelligence 77(1–2):1–25.
- T. Besold, K.-U. Kühnberger. Applying AI for Modeling and Understanding Analogy-Based Classroom Teaching Tools & Techniques. In KI 2014: Advances in Artificial Intelligence - 37th Annual German Conference on AI, LNCS, Springer, 2014.
- E. Cambouropoulos, M. Kaliakatsos-Papakostas, K.-U. Kühnberger, O. Kutz, A. Smaill: Concept Invention and Music: Creating Novel Harmonies via Conceptual Blending, Proceedings of the 9th Conference on Interdisciplinary Musicology (CIM) 2014.

1.1 Mathematics

In deliverable D6.1 'Survey of conceptual blending in mathematics and formal reasoning' some initial examples of mathematical concepts are described that can be partially reconstructed by means of formal conceptual blending, in terms of formal analogy making (HDTP) and a categorical colimit approach (realized by Hets). First, it was shown shown how to obtain a (partial) axiomatization of the integers as a blend (i.e. a colimit construction) of a (partial) axiomatization of the natural numbers and its corresponding 'rotated' version. Second, it was shown how to obtain a very closed and, at the same time, more general theory than the one defined by the complex numbers, by blending the concepts of normed vector spaces of dimension two and fields. This was achieved by using a standard axiomatization for Abelian groups as the generic space. Third, the quaternions and octonians were described in a more informal way as instances of concepts that could be generated through a blending approach as in the case of the complex numbers. Fourth, the notion of projectile motion coming from physics was expressed as a blend of conceptual spaces. Finally, the mathematical concept of 'geometrical symmetry' in a two dimensional plane was also generated as a blending construction using reflexion and projection maps.

Now, in this deliverable we demonstrate that the particular formalizations of conceptual blending that is used in COINVENT not only have the conceptual 'power' for being considered as theoretical bricks for producing basic mathematical concepts, but can also be used to generate entire complex mathematical theories such as the Theory of Fields and Galois Theory. Both theories can be considered as very rich and important mathematical structures.

In the first article of this deliverable, entitled "The Role of Blending in Mathematical Invention" we reconstruct a partial axiomatization of the integers as a formal blend of a partial axiomatization of the natural numbers using the successor function and the notion of an inverse function.

In this example, we removed by hand one of the axioms of the first concept (pseudo-natural numbers) in order to obtain a consistent theory. Additionally, by allowing weakened blends, i.e., blends using smaller axiomatizations of the input concepts, we obtain the notion of an (actual) infinite iterative process as a blend of the concept of a (potential) infinite iterative process and completed iterative process.

A more sophisticated example emerges when we blend the notion of an ideal of a commutative ring with unity (with the collection of all possible ideals of this ring) and the notion of a prime number in (a weakened version) of the integers. By choosing a very simple generic space identifying a minimal amount of sorts we were able to co-discover a new class of commutative rings with unity, i.e., the 'containment-division rings' (CDR). These rings are defined by the condition that for any ideals I and J of the corresponding ring, I is contained in J if and only if J divides I as ideals, namely, there exists another ideal H such that I is equal to the product of the ideals H and J. With this terminology, the resulting blend is the concept of a prime ideal over a CDR (with the corresponding collection of all ideals of this ring). Now, the concept of a prime ideal is a fundamental formal 'column' of the most successful modern approach to algebraic geometry given in terms of (affine) schemes[2]. Furthermore, it is one of the key notions of commutative algebra and the corresponding dimension theory for commutative rings with unity[3]. So, the fact that we were able to generate mathematical seminal notions not only in the domain of complex analysis (e.g. the complex numbers) and basic arithmetic (e.g. the integer) but also in commutative algebra (e.g. prime ideals over a CDR) starts to show that formal conceptual blending can play a structural and 'omnipresent' role in mathematical creation, namely, it can be seen as a kind of a 'meta-generator' of mathematical concepts. The idea of meta-generator is further developed in the second article of this deliverable, entitled "Conceptual Blending as a creative meta-generator of mathematical concepts: Prime Ideals and Dedekind Domains as Blend", and the third article, entitled "Generating Fundamental Notions of Fields and Galois Theory through Formal Conceptual Blending".

In addition to these findings, it turns out that in a Noetherian setting (i.e. in a category where all the ideals are finitely generated) this new mathematical concept (CDR), being discovered with the help of this formal-cognitive tool (conceptual blending), is mathematically equivalent to a fundamental notion of algebraic number theory, namely a Dedekind domain. Effectively, let us assume an analogical formalization of conceptual blending in terms of the union of the axioms of the input spaces are given by the corresponding signature morphisms. Then, the composed notion of a prime ideal over a Dedekind domain joined with its spectrum of ideals was obtained as a blend of the concepts of an ideal over a Noetherian domain and a prime number over a weakened version of the integers (cf. Section 2 of Article 2 of this deliverable).

In one of the last sections of Article 1 of this deliverable, we suggested in a very informal way that our formal framework of conceptual blending can be used to reconstruct the most fundamental concepts of Fields and Galois theory. So, after some further work we were able to present in Article 3 a concrete formal manifestation of these previous intuitions. We start to work with five basic mathematical concepts, all of them supported by well-known examples and used very often implicitly and explicitly in modern mathematics (cf. Section 2 of Article 3): Abelian Group, Pointed Abelian Group, Space of Distribution, Action of a group over a Set, and Fixed Point Space. Secondly, we start to blend each of these concepts iteratively using very simple generic spaces in-

[2]Cf. A. Grothendieck and J. Dieudonné: Eléments de Géométric Algébrique I. Springer, 1971.

[3]Cf. D. Eisenbud: Commutative Algebra with a View Toward Algebraic Geometry. Springer, 1995.

ducing simultaneously the simplest kinds of analogical relation between the corresponding input spaces, namely, relations given by renaming sorts corresponding to sets, functions and relations. Thirdly, after doing nine times the operation of conceptual blending with a formalization of colimits of theories in many-sorted first order logic, we obtain four of the most fundamental concepts of Fields and Galois theory, i.e., the mathematical notions of Fields, Field Extension, Group of Automorphisms of a Field, and the Group of Automorphisms of a Field Extension fixing the base field (cf. Figure 1 in Article 3). This last concept coincides with the notion of Galois Group when the corresponding extension is a Galois extension. We run explicitly all the related implementation in Hets (cf. Section 3 of Article 3).

This collection of examples can be seen as foundational mathematical 'evidence' that formal conceptual blending plays a central role as a concrete 'meta-mathematical' operation allowing us to model more and more aspects of mathematical creativity. Now, we are not talking anymore about isolated mathematical concepts but about the conceptual production of entire theories with formal conceptual blending. We can make a simple analogy with the fundamental theorem of arithmetic, more precisely, the unique factorization theorem in the integers, in order to understand better the idea behind generating these fundamental concepts of Fields and Galois theory. Let us consider for a moment (mathematical) concepts as integer numbers, and formal conceptual blending as a product between numbers, i.e., if a concept B is the blending of the input concepts A and C, then C can be written as a formal product of A and C. So, one of the main claims in Article 3 of this deliverable is to find an explicit prime factorization of this particular four concepts of Fields and Galois theory.

Of course, the fact that it was possible to find such conceptual factorization for notions in Fields and Galois theory, using prime concepts coming from other areas of mathematics like topology, implicitly suggests that it should be possible to do exactly the same for mathematical concepts in every mathematical domain. On the other hand, a very interesting question emerging in this context is how can we determine when a mathematical concept can be seen as 'prime' concept in the sense that it cannot be decomposed as a non-trivial blending of two simpler concepts.

One of the issues of current research is trying to use our formalizations of conceptual blending for improving automated theorem provers and automated (reasoning) systems like for example MATHsAID[4]. The main advantage of our procedure is that with our colimit formalization to conceptual blending we have a quite concrete way of generating more efficiently possible creative definitions in comparison with approaches doing a rough combinatorial search.

In conclusion, we have shown with many examples that our formalizations of conceptual blending are a feasible conceptual tool for re-discovering and generating classical mathematical concepts, co-discovering new ones, starting to produce entire mathematical theories and originating concepts having pieces in different areas of pure mathematics. So, this work aims to serve also as a strong formal support of the soundness of our approach to formal conceptual blending from both perspectives: the theoretical one, related with the relevance of this approach for pure mathematics; and the computational one, related with the appropriateness of our models for being successfully implemented.

[4]Cf. R. McCasland, A. Bundy, and P. Smith (2006): Ascertaining mathematical theorems, ENTCS 151:21-38.

1.2 Music

In the music domain, the focus of the Coinvent project is the generation of new and innovative chord progressions for automatic harmonization systems. Historically, major developments of chord progression during centuries of music production and composition resulted in many different idioms and traditions of how musicians used harmonic spaces and sequences of chords. This deliverable contains one paper (Article 4) entitled "Computational Invention of Cadences and Chord Progressions by Conceptual Chord-Blending" where it is documented in detail how conceptual blending can be used to compute interesting and innovative chord progressions.

The algebraic basis for blending in the music domain is the coding of conceptual spaces in a variant of the *Common Algebraic Specification Language* (CASL)[5]. Chords are encoded as feature-value pairs and the degree of importance of features is represented using priority values. Generalizations are computed by weakening the input spaces, i.e. by removing axioms from the inputs, guided by priority values and heuristics. Finally blending is computed by a colimit operation.

Applying this machinery results in rather interesting examples of new chord progressions: by blending a perfect cadence and a Phrygian cadence the systems computes, dependent on the priority values, a Tritone substitution or a Backdoor progression as blended chord sequences, well-known chord progressions for Jazz musicians, but developed centuries after perfect cadences and Phrygian cadences were used by composers. Another type of blending is cross-fading: concatenating 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. As an example a progression C $\rightarrow$ F and a progression G#7 $\rightarrow$ C# is blended to a chord progression C $\rightarrow$ $C^0 7$ $\rightarrow$ C# by blending the right chord of the first input with the left chord of the second input. The result is an interesting modulation from C to C# via a diminished minor seventh chord $C^0 7$.

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

2 The Role of Blending in Mathematical Invention (Article 1)

F. Bou, M. Schorlemmer, J. Corneli, D. Gomez-Ramirez, E. Maclean, A. Smaill, and A. Pease (2015): The Role of Blending in Mathematical Invention. Proceedings of the 6th International Conference on Computational Creativity (ICCC) 2015.

Abstract:
We model the mathematical process whereby new mathematical theories are invented. Here we explain the use of conceptual blending for this purpose, and show examples to illustrate the process in action. Our longer-term goal is to support machine and human mathematical creativity.

The role of blending in mathematical invention

F. Bou
IIIA - CSIC, Barcelona

M. Schorlemmer
IIIA - CSIC, Barcelona

J. Corneli
Computing, Goldsmiths, London

D. Gómez-Ramírez
Cognitive Science, Osnabrück

E. Maclean
Informatics, Edinburgh

A. Smaill
Informatics, Edinburgh

A. Pease
Computing, Dundee

Abstract

We model the mathematical process whereby new mathematical theories are invented. Here we explain the use of conceptual blending for this purpose, and show examples to illustrate the process in action. Our longer-term goal is to support machine and human mathematical creativity.

1 Introduction

We are concerned with creativity in mathematics: creativity as evinced by human and artificial mathematicians, individually and collectively.

Work on *conceptual blending* has been much influenced by Fauconnier and Turner (1998, 2002). More recently, the centrality of conceptual blending to creativity has been stressed by Turner (2014), where he writes:

> ...the human spark comes from our advanced ability to *blend* ideas to make *new* ideas. Blending is the origin of ideas. (Turner, 2014, p 2)

The claim is that blending in this sense is a general human cognitive ability, and as such applies to mathematics, as much as to art, poetry, music and so on (see for example Turner (2005)).

The place of mathematics and the sciences among creative endeavours has been stressed by the literary critic George Steiner:

> It is in mathematics and the sciences that the concepts of creation and of invention, of intuition and of discovery, exhibit the most immediate, visible force.
>
> Steiner (2001, p 145)

Blending involves recognising features common to mathematical concepts, even when expressed in different terminology. The role of mathematical analogy in creative mathematics is well expressed by Weil (1960), and a general plea for analogical reasoning within science in (Arbib and Hesse, 1986).

We are investigating *computational* accounts of mathematical creativity, taking conceptual blending as a key ingredient. The work of Goguen (1999, 2005) has provided a general framework for comparison of conceptual spaces, and computation of blends. This enables the use of richer representation formalisms, and so is closer to contemporary mathematics than previous computational realisations of blending, such as in Pereira (2007).

This paper deals with the creative process in mathematics, as modelled along the lines above. We focus on the use of blending within a single process, searching for blends satisfying some evaluation criteria, from the starting point of some given conceptual spaces.

While cognitive issues are important to us, this paper is focused on issues in representation and representation change; there are however brief comments on cognition in the conclusions.

We start by providing some background, followed by an example to illustrate the components involved in our approach. A historical example based on Georg Cantor's work follows. The most extended example was carried out by a pure mathematician (D. Gómez-Ramírez), working in a domain close to his own; in this case, the blend mechanism threw up some unexpected properties, which provoked new work by the mathematician.

Subsequently we give some more speculative thoughts on where this work can go in the future, by considering Galois theory as a test-bed. Finally, we discuss the evaluation of work along these lines, and give some conclusions.

2 Background

2.1 Blending in Mathematics

Lakoff and Núñez (2000) are among the first to present a cognitive account of the origin and development of mathematical ideas,[1] arguing against the "romance of mathematics" in which mathematics is presented as an ever-increasing set of universal, absolute, certain truths which exist independently of humans. They present the thesis that human mathematics is grounded in bodily experience of a physical world, and mathematical entities inherit properties which objects in the world have, such as being consistent or stable over time. Exploring the physical world of object collection might lead to concepts like the empty collection and rules like "adding a collection of n objects to an empty collection yields a collection with n objects". People then form grounding metaphors between the physical world and an abstract mathematical world, allowing us to project from everyday experiences onto abstract concepts, thus leading to the concept of zero and the axiom that $n+0=n$. Lakoff and Núñez posit that blending different mathematical metaphors leads to more complex ideas (see also Alexander (2011)).

Alongside this account of mathematical cognition, mainstream contemporary mathematics has developed its own methodology and foundations, enjoying an exceptional place among scientific disciplines. Its methods, objects of study and sometimes astonishing results have widespread, if not universal, acceptance.

In conclusion, mathematics is a scientific discipline having not only a fundamental cognitive component, necessary in its development, but also possessing a collection of general principles and structures going beyond a particular school of thought. Among these general processes we want to highlight in this paper the importance that conceptual blending has in mathematics, incorporating both cognitive and mathematically specific aspects in order to create new mathematical concepts.

[1] This is lamented by Lakoff and Núñez, who claim that (prior to their work), "there was still no discipline of mathematical idea analysis from a cognitive perspective" (Lakoff and Núñez, 2000).

2.2 Terminology for conceptual blending

Our notion of conceptual blending is informed by Category theory, and highly influenced by Goguen's work on concepts (Goguen, 2005). In this paper we use the terminology below, and elucidate the terminology by means of a running example – discovering a version of the integers (in the sense of providing a partial approach to the genuine integers) using blending.

Conceptual spaces are partial and temporary representational structures which are constructed on the fly when talking about a particular situation, which are informed by the knowledge structures associated with a domain. These are influenced by Boden's idea of a concept space which is mapped, explored and transformed by transcending mapped boundaries (Boden, 1977), and form the input spaces to our blend.

As an example of two conceptual spaces, consider one as a theory NAT – a theory of the natural numbers, and FUNC – a theory of a total unary function with an inverse. We will refer back to these theories in this exposition.

(Many-sorted) First-order **Axioms** are the criteria which will be used here to delineate the conceptual spaces. The axiomatic method has been a fundamental aspect of mathematical research since Euclid, and various axiom changes have led to revolutions in mathematics. For instance, rejecting the parallel postulate opened up fascinating new areas of non-Euclidean geometry.

The precise formulations for NAT and FUNC can be found in Listings 1 and 2. Notice that these formulations obviously refer to partial representations of the genuine concepts employed by mathematicians. In the conceptual space with theory NAT, an example of an axiom is $\forall x.\neg 0 = s(x)$ – that is that zero is not a successor element. The conceptual space with theory FUNC has an axiom $\forall x.\, f(inv(x)) = x$.

Signature morphisms between conceptual spaces are mappings from the symbols of the source conceptual space into the symbols of the other conceptual space. For example NAT contains a function $\lambda x : Nat.\ s(x)$ that maps x to its successor, and FUNC contains a function defined over a set X that maps each element to an image $\lambda x : X.\, f(x)$. A theory G with a morphism to both NAT and FUNC might contain a function $\lambda x : N.\, func(x)$ that takes every number in some set N to its image under *func*. When we show a mapping we write this as

$$s \leftarrow_{\phi(G,\mathrm{NAT})} func \rightarrow_{\phi(G,\mathrm{FUNC})} f \quad (1)$$

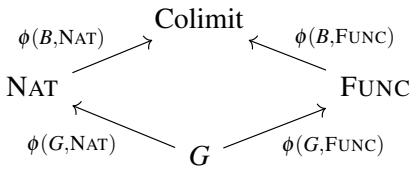
$$Nat \leftarrow_{\phi(G,\mathrm{NAT})} N \rightarrow_{\phi(G,\mathrm{FUNC})} X \quad (2)$$

The mapping $\phi(G,\mathrm{NAT})$ is a signature morphism from G to NAT. Note that associated types are also mapped.

Input Spaces refer to two or more conceptual spaces of interest.

Generic spaces are conceptual spaces that possess commonality between input spaces.

Colimits are conceptual spaces representing a blend of input spaces with respect to a given generic space and a set of signature morphisms. These are uniquely computed given a generic space and a set of morphisms. Here is a diagrammatic representation of such a computation in our example using theories NAT and FUNC :



The conceptual space represented by the Colimit is often referred to as the *blend*.

Internal Evaluation constitutes a variety of techniques to determine whether a computed colimit is viable as a conceptual space. In our example, since the conceptual spaces are mathematical theories, we can exploit the notion of consistency. This is a way of evaluating whether a blend is not only creative, but also valid. In the example of theories NAT and FUNC, the computed blend is inconsistent due to the emergent axioms in the computed colimit. The only type existing within the colimit is from now on referred to as $\mathbb{Z}$ to distinguish it from the natural numbers. Notice that in the colimit it holds that:

$$\forall x : \mathbb{Z}.\neg\ zero = s(x)$$

$$\forall x : \mathbb{Z}.\ s(sinv(x)) = x\ .$$

This is an inconsistency, as from the second axiom we see that there is an element for which 0 is the successor.

Weakening refers to the process of weakening the input theories by removing symbols or axioms. If we remove the axiom

$$\forall x : Nat.\neg\ zero = s(x)$$

then the resulting computed colimit contains a mathematical theory which is consistent.

Martinez (2014) provides an algorithm to explore the space of blends resulting from given input spaces and a given generic space, where weakening is achieved by omitting axioms. The algorithm returns the blends which are consistent, and maximally so, among those in this space of blends. This algorithm assumes that consistency of relevant theories can be checked, so is not always effective.

Running the blend refers to elaborating or completing a mathematical theory. Sometimes there are missing definitions which need to be discovered. For example in the new theory the following axiom appears

$$\forall x, y : \mathbb{Z}.\ s(x) + y \quad = \quad s(x + y)\,,$$

but we also are interested in theorems such as

$$\forall x, y : \mathbb{Z}.\ sinv(x) + y \quad = \quad sinv(x + y)\,.$$

Finding suitable theorems is an example of running the blend, and from which it is possible to discover and prove theorems such as

$$\forall x, y : \mathbb{Z}.\ sinv(x) + s(y) \quad = \quad x + y\,.$$

2.3 Technologies

The approach explained above corresponds to Goguen's proposal (Goguen, 1999) for implementing blending, but slightly simplified (as in Kutz, Neuhaus, Mossakowski, and Codescu (2014)): we use the normal colimit construction, rather than $\frac{3}{2}$-colimits (both described in Goguen (1999)).

Additionally we assume that the conceptual spaces involved are given using a CASL specification (Astesiano, 2002) and that the morphisms are theorem preserving (i.e. map theorems to theorems). The reason for these assumptions is that in these cases it is well-known how to compute colimits: the colimit specification essentially corresponds to the disjoint union of the two target

conceptual spaces except for not repeating the symbols given in the common source conceptual space. Moreover, we will be using the HETS system (Mossakowski, Maeder, and Lüttich, 2007) to compute such colimits. The code for the implemented examples in this paper is available on-line.[2]

The use of CASL specifications means that we deal with first-order logic; CASL is supported in the HETS system, and colimits here can be computed in the current implementation of HETS. Although higher-order logic (with Henkin semantics) is available in HETS (indeed in CASL) and the colimits are well-known to exist (because higher-order in this form is reducible to many-sorted first-order logic), it is worth noticing that the calculus of such colimits is not currently available in HETS. This restricts the formalisms that can be used directly for our purposes, where computation of colimits is central to our approach.

3 Blending and the infinite

3.1 Example Revisited – the Integers

As a first demonstration of the machinery involved in blending mathematical theories, we consider combining a theory of natural numbers with the concept of the inverse of a function to obtain the integers. Let us assume a simple partial axiomatisation of the natural numbers (without order axioms) as shown in Listing 1, and call this theory NAT. Now let us also define a simple theory which introduces the concept of a function with an inverse as shown in Listing 2, and call this theory FUNC.

spec NAT =
  sort *Nat*
  ops $zero : Nat$;
    $s : Nat \to Nat$;
    $\_\_+\_\_ : Nat \times Nat \to Nat$
  $\forall\, x, y : Nat$
  • $s(x) = s(y) \Rightarrow x = y$
  • $\neg\, zero = s(x)$
  • $s(x) + y = s(x + y)$
  • $zero + y = y$
end

Listing 1: A theory of the natural numbers without order

3.1.1 Identifying a Generic Space

In order to incorporate the notion of blending here we want to be able to identify a "generic" component of each theory and compute the colimit. We can use the HDTP system (Gust, Kühnberger, and Schmidt, 2006; Schmidt, 2010) to discover a common theory and signature morphism between symbols in the two theories NAT and FUNC. The Generic theory GEN contains a sort *N*

[2] See: https://github.com/ewenmaclean/ICCC2015_hetsfiles

spec FUNC =
sort X
op $f : X \to X$
op $finv : X \to X$
$\forall\, x : X$
• $f(finv(x)) = x$
• $finv(f(x)) = x$
end

Listing 2: A theory with a function and its inverse defined

and a function *func*, and the morphisms from the Generic theory to NAT and FUNC are:

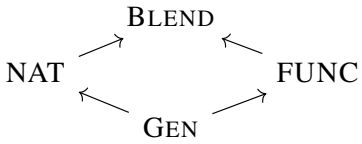
$$s \leftarrow_{\phi(G,\mathrm{NAT})} func \rightarrow_{\phi(G,\mathrm{FUNC})} f \quad (3)$$

$$Nat \leftarrow_{\phi(G,\mathrm{NAT})} N \rightarrow_{\phi(G,\mathrm{FUNC})} X \quad (4)$$

Here the successor function is identified in the mapping with the function in the theory FUNC.

3.1.2 Computing the Colimit

The HETS system (Mossakowski et al., 2007) can then be exploited to find a new theory by computing the colimit:



This generates the theory shown in Listing 3 (for the sake of understanding it is used p, for predecessor, instead of *sinv*).

3.1.3 Removal of Inconsistencies

This theory is automatically determined to be inconsistent due to the axioms

$$\forall x : \mathbb{Z}.\neg\ zero = s(x) \quad (5)$$

$$\forall x : \mathbb{Z}.\ s(p(x)) = x \quad (6)$$

Removal of the limiting axiom (5) from Listing 1 results in generating a blend theory which is very similar to what we understand to be the integers as shown in Listing 4.

3.1.4 Running the Blend

Running the blend refers to discovering definitions or adding axioms to flesh out the blend. In the example of the version in Listing 4, the definition of plus needs to be extended to understand how to calculate with the predecessor function:

$$p(x) + y = p(x + y)$$

```
spec SPEC =
    sort  N
    op    __+__ : N × N → N
    op    p : N → N
    op    s : N → N
    op    zero : N
    ∀ x, y : N • s(x) = s(y) ⇒ x = y
    ∀ x : N • ¬ zero = s(x)
    ∀ x, y : N • s(x) + y = s(x + y)
    ∀ y : N • zero + y = y
    ∀ x : N • s(p(x)) = x
    ∀ x : N • p(s(x)) = x
end
```

Listing 3: An inconsistent partial approach to the integers (without order)

```
spec SPEC =
    sort  N
    op    __+__ : N × N → N
    op    p : N → N
    op    s : N → N
    op    zero : N
    ∀ x, y : N • s(x) = s(y) ⇒ x = y
    ∀ x, y : N • s(x) + y = s(x + y)
    ∀ y : N • zero + y = y
    ∀ x : N • s(p(x)) = x
    ∀ x : N • p(s(x)) = x
end
```

Listing 4: A consistent partial approach to the integers (without order)

from which theorems such as

$$p(x) + s(y) = x + y$$

can be proved.

4 Potential and actual infinity

Some of the ideas of Lakoff and Núñez (2000) have been reworked by the authors, with increased emphasis on conceptual blending. In particular, the analysis of mathematical infinity, given in metaphorical form as the "Basic Metaphor of Infinity" (BMI) in Lakoff and Núñez (2000), is represented in blend form in Núñez (2005) as the "Basic Mapping of Infinity" (so, still "BMI").

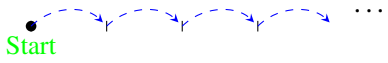
We show here how this blend works out in our setting. The BMI suggests that the notion of completed infinity, in particular the possibility of transfinite numbers in the sense of Cantor,

comes from a blend of the notion of completed, finite process with that of a potentially infinite and endless process.

Thus take two corresponding input spaces, given by CASL specifications **FinEnd** and **Inf** corresponding to the following diagrams

FinEnd:

Inf:



- **FinEnd:** Completed Iterative Processes are those that from some initial state, terminate in a final state after a finite number of state transitions. One such case is chosen.

- **Inf:** Infinite Iterative Processes are those that continue indefinitely to change state.

In both cases, the arrows indicate steps of the processes, and the process states are in a discrete linear order indicated by left-to-right order in the diagrams.

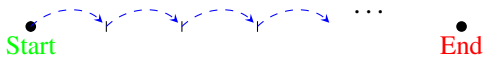
The generic space **Gen** simply identifies the start states, the notion of process step, and the linear ordering of states.

Now we can compute the blend of these spaces, which includes new features taken from both of the input spaces. This blend is *inconsistent*, for the following two reasons:

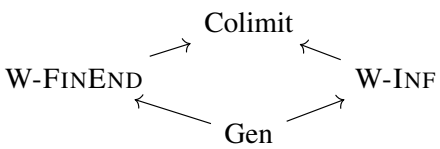
1. the number of states is finite (from **FinEnd**), and infinite (from **Inf**);

2. there both is an end state (from **FinEnd**) and is no end state (from **Inf**).

Search through the possibilities of weakening the input spaces by omitting as few axioms as possible among those involved in an inconsistency reveals the possibility of a structure with infinitely many states (from **Inf**) and an end state (from **FinEnd**). Computing the colimit from the weakened input spaces **W-FinEnd**, **W-Inf** gives a theory corresponding to this diagram:



Thus we have a blend as in the earlier examples:



5 Prime Ideals as a blend

Introduction

One of the most fundamental concepts of modern mathematics, which is the basis of commutative algebra and a seminal ingredient of the language of schemes in modern algebraic geometry, is that of *prime ideal* (Grothendieck and Dieudonné, 1971; Eisenbud, 1995).

The terminology "prime ideal" relates to the older notion of "prime number". The initial aim of this work was to look for a blend between prime numbers (from the integers) and the ideals of a commutative ring, to see what would emerge. It turned out that *the blend process*, along with providing a definition for prime ideals, also *suggested an unexpected concept in the context of rings*, namely what will be called Containment Division Rings (CDR). In turn, this prompted questions and proofs about this concept – thus running the blend (space prevents description of this step in this paper).

We present a first blend involving weakening, followed by a second blend from fuller input spaces, where the emergent concept of CDR appears.

5.1 The first conceptual space

Let $(R,+.*,0,1)$ be a commutative ring with unity (see the formal definition and examples in Eisenbud (1995)). Now, R can be understood as the sort containing the elements of the corresponding commutative ring with unity. An ideal I is a subset of R satisfying the following axiom:

$$(\forall i,j \in I)(\forall r \in R)(i+(-j) \in I \wedge r*i \in I).$$

Let us define a unary relation (predicate) *isideal* on the set (sort) of subsets of $P(R)$ corresponding to this definition. Now, we define

$$\mathrm{Id}(R) = \{A \in P(R) : \mathit{isideal}(A)\}.$$

Ideals are "multiplied" using the following definition:

$$I \cdot_\iota J = \left\{ \sum_{k=1}^{n} i_k \cdot j_k : n \in \mathbb{N} \wedge i_1,\ldots,i_n \in I \wedge j_1,\ldots,j_n \in J \right\}.$$

In other words, $I \cdot_\iota J$ is the smallest ideal extending the set $\{i \cdot j : i \in I \wedge j \in J\}$.

The key property that we want to keep in the blend is the one saying that this operation $\cdot_\iota$ has a neutral element 1_ι, which can be seen as an additional notation for the ring. On the other hand, we want to see the containment relation $\subseteq$ as a binary relation over the sort $\mathrm{Id}(R)$.

Summarizing, our first conceptual space consists of sorts $R, \mathrm{Id}(R)$ and $P(R)$; operations $+,*,0_R,1_R,1_\iota$ and $\cdot_\iota$; and the relations $\subseteq$ and *isideal*.

Let us denote this space by $\mathbb{I}$.

5.2 The second conceptual space

Let $\mathbb{Z}$ be the set of the integer numbers. Here, we choose any partial axiomatization of them including at least the fact that $(\mathbb{Z},*,1)$ is a commutative monoid. We define also an upside-down

divisibility relation $\lfloor$ defined as $e\lfloor g := g|e$, i.e. there exists an integer c such that $e = c * g$. Let us define a unary relation *isprime* on $\mathbb{Z}$ as follows: for all $p \in \mathbb{Z}$, $isprime(p)$ holds if $p \neq 1$ and:

$$(\forall a,b \in \mathbb{Z})\ \big((ab\lfloor p) \rightarrow (a\lfloor p \vee b\lfloor p)\big).$$

Besides, we define the set (sort) of the prime numbers as

$$Prime = \{p \in \mathbb{Z} : isprime(p)\}$$

In the CASL language, we consider $\mathbb{Z}$ as the sort of the integer numbers, $*$ as a binary operation, *prime* as a predicate and $\lfloor$ as a binary relation, any of them defined over the sort $\mathbb{Z}$. We denote this conceptual space by $\mathbb{P}$.

5.3 The Generic Space

The generic space $\mathbb{G}$ consists of a set (sort) G with a binary operation $*_G$, a neutral element S and a binary relation $\leq_G$.

5.4 The Blending Morphisms

The morphism to $\mathbb{I}$ uses:
$\varphi(G) = \mathrm{Id}(R), \varphi(*_G) = *_\iota, \varphi(S) = 1_\iota$ and $\varphi(\leq_G) = \subseteq$;
the morphism to $\mathbb{G}$ uses:
$\delta(G) = \mathbb{Z}, \delta(*_G) = *, \delta(S) = 1$ and $\delta(\leq_G) = \lfloor$.

5.5 The Axiomatization of the Blending

A straightforward colimit construction based on the input and generic spaces above yields a consistent space with properties inherited both from the prime elements into the integers and from the ideals of commutative rings; one of the concepts is a notion of prime ideals, another is that of CDR.[3] Here we describe briefly a weakening of the given spaces that makes the resultant blend more generally applicable.

From the properties defining the integers we transfer into the blend only the fact that $\mathbb{Z}$ is a set with a binary operation $*$ having 1 as neutral element and $\lfloor$ as a binary relation, without taking into account its formal definition.

Now after computing the colimit, we obtain that any element $P \in G$ (i.e., an ideal of S) satisfies the predicate *isprime* if and only if

$$P \neq S \wedge (\forall X,Y \in G = \mathrm{Id}(S))(X \cdot_\iota Y \subseteq P \rightarrow (X \subseteq P \vee Y \subseteq P)).$$

Thus, the predicate *isprime* turns out to be the predicate characterizing the primality of ideals of S and the set (sort) *Prime* turns out to be the set of prime ideals of S.

Using the weakened input spaces, the blending space consists of the axioms assuring that S is a commutative ring with unity, G is the set of ideals of S, *isprime* is the predicate specifying primality for ideals of S and *Prime* is the collection of all prime ideals of S.

[3] A ring R is a Containment Division Ring (CDR) if for all ideals I and J of R, $I \subseteq J$ if and only if J divides I (i.e. there exists an ideal U such that $I = U \cdot_\iota J$).

5.6 Implementation for prime ideals over CDR-s as a blend

In this section we construct the concept of prime ideal over a CDR as a blend of the conceptual space of ideals of a commutative ring with unity and the conceptual space of the former second conceptual space where the axiom defining the upside-down divisibility relation is restored.

It is worth mentioning again that the definition of CDR-s was obtained after doing this implementation and therefore it could be seen as a form of "creative" result coming from the blending process.

After computing the corresponding colimit in HETS and interpreting "RingElt" as the sort containing the elements of the ring S, the theory defining the blend corresponds to the axioms defining a CDR (S), the set of all its ideals (*Generic*), the set all its prime ideals (*SimplePrime*) and a primality predicate (*IsPrime*). We present in Listing 5 just the theory corresponding to the colimit (omitting details of ring axioms and ideal generation).

spec SPEC =
sorts *Generic*, *RingElt*, *SimplePrime*, *SubSetOfRing*
sorts *SimplePrime* $<$ *Generic*,IdGeneric $<$ *SubSetOfRing*
ops $0, 1, S$: *RingElt*
op $\_\_*\_\_$: *RingElt* $\times$ *RingElt* $\to$ *RingElt*
op $\_\_+\_\_$: *RingElt* $\times$ *RingElt* $\to$ *RingElt*
op $\_\_x\_\_$: *Generic* $\times$ *Generic* $\to$ *Generic*
pred *IsIdeal* : *SubSetOfRing*
pred *IsPrime* : *Generic*
pred $\_\_isIn\_\_$: *RingElt* $\times$ *SubSetOfRing*
pred *gcont* : *Generic* $\times$ *Generic*
pred $\_\_generates\_\_$: *RingElt* $\times$ *Generic*
$\forall\ I$: *SubSetOfRing* $\bullet\ I \in Generic \Leftrightarrow IsIdeal(I)$
$\forall\ x$: *Generic* $\bullet\ x\ x\ S = x$
$\forall\ x$: *Generic* $\bullet\ S\ x\ x = x$
$\forall\ A, B$: *Generic*
$\bullet\ gcont(A, B) \Leftrightarrow \forall\ a$: *RingElt* $\bullet\ a\ isIn\ A \Rightarrow a\ isIn\ B$
$\forall\ x, y$: *RingElt* $\bullet\ x + y = y + x$
%% and further ring axioms ...
$\forall\ I$: *SubSetOfRing*
$\bullet\ IsIdeal(I)$
$\Leftrightarrow \forall\ a, b, c$: *RingElt*
$\bullet\ ((a\ isIn\ I \Rightarrow a\ isIn\ S) \wedge 0\ isIn\ I)$
$\wedge\ (a\ isIn\ I \wedge c\ isIn\ S \Rightarrow c * a\ isIn\ I)$
$\wedge\ (a\ isIn\ I \wedge b\ isIn\ I \wedge c\ isIn\ S \wedge b + c = 0$
$\Rightarrow a + c\ isIn\ I)$
$\forall\ a$: *RingElt*; *A* : *Generic*
%% and axioms for generates and x ...
$\forall\ x, y$: *Generic* $\bullet\ gcont(x, y) \Leftrightarrow \exists\ c$: *Generic* $\bullet\ x = y\ x\ c$
$\forall\ p$: *Generic* $\bullet\ p \in SimplePrime \Leftrightarrow IsPrime(p)$
$\forall\ p$: *Generic*
$\bullet\ IsPrime(p)$
$\Leftrightarrow (\forall\ a, b$: *Generic*
$\bullet\ gcont(a\ x\ b, p) \Rightarrow gcont(a, p) \vee gcont(b, p))$
$\wedge \neg\ p = S$
end

Listing 5: Colimit for prime ideals over CDR-s

6 A Challenge Example for Blending

6.1 Computational Creativity via Blending

The examples shown thus far in the paper have been examples of blending in mathematics whose mechanisation has helped to identify some novel and unexpected results. The blending itself was a one-stage process where human input was required to identify the input concepts. A more ambitious aim of the approach of applying blending to the problem of computational creativity in mathematics, is to allow search to be done over multiple blends and for the *process* of blending to be controlled mechanically. In this section we describe very informally a mathematical domain that seems in some ways a natural candidate for a blending approach.

6.2 Galois Theory

Galois theory develops a relationship between a polynomial $p(x)$ with coefficients in some field F, the extension of K of F (written "K/F") containing all of the roots of $p(x)$ in the algebraic closure of F, and the group $\mathbf{Gal}(K)$ of automorphisms of K/F that fix the elements of F. The fundamental theorem of Galois theory states that there is a bijection between the subfields of K/F and the subgroups of $\mathbf{Gal}(K)$; namely, subgroups correspond to their fixed fields. Using this correspondence, properties of polynomials can be derived, most famously the fact that quintic polynomials cannot be solved by algebraic operations and the extraction of roots.

We do not propose to reconstruct much of the theory here, but note that already in this basic account there are several steps that seem compellingly "blend-like."

In the first place, for field extension, E is an extension of F if F is a subfield of E. We could derive the extension relationship from the input concepts E and F by "taking everything additional from E and adding it to F." This is made specific in the process of *adjoining* elements, which simply means to augment the field with all fractions of formal finite sums and products of the adjoined elements with coefficients in the base field.

Second, the notion of the *splitting field* of a polynomial, namely the special extension K/F containing all of the roots of $p(x)$. This could be formed conceptually by combining the concept "*the roots of a polynomial $p(x)$ with coefficients in a field F*" and the concept "*a field extension E/F formed by adjoining certain elements to F*."

As above, we could then form the concept of $\mathbf{Gal}(K)$ by blending at the conceptual level. This time, there would be several constituent pieces: "*the roots of a polynomial $p(x)$ with coefficients in a field F*," "*the splitting field of $p(x)$*," "*the group of automorphisms of a field extension E*," "*the automorphisms that fix F*."

Finally, assuming that we have built $\mathbf{Gal}(K)$ in this fashion, we would like to know some of its properties. Consider the claim that *elements of* $\mathbf{Gal}(K)$ *permute the roots of* f. This time, instead of being purely conceptual, we want to work at the *process* level, and consider before-and-after descriptions of the result of applying $\varphi \in \mathbf{Gal}(K)$ to some r with the property $p(r) = 0$. This is similar in some ways to the "Riddle of the Buddhist Monk", popularised by Koestler (1964), which is cited as an example of the power of blending.[4] However, this time the generic space is not a simple geometric machine, but rather an algebraic machine with several moving parts.

[4] "A Buddhist monk begins at dawn one day walking up a mountain, reaches the top at sunset, meditates at the top for several days until one dawn when he begins to walk back to the foot of the mountain, which he reaches at sunset. Making no assumptions about his starting or stopping or about his pace during the trips, prove that there is a place on the path which he occupies at the same hour of the day on the two separate journeys."

The proof of the claim is as follows. If $p(r) = 0$, then $\varphi p(r) = \varphi 0$. Since φ is an automorphism, $\varphi 0 = 0$; and furthermore φ distributes over the sums and products that make up the polynomial $p(x)$ and fixes its coefficients, therefore $\varphi p(r) = p(\varphi r)$. Chaining the equalities together, we have $p(\varphi r) = 0$.

In short, the proof is a fairly direct result of combining the definitions. Goguen (1992) suggests that "combination is colimit." Can we realise the proof through (one or several) colimit operations? And is there anything special about this proof? Apart from these more theoretical questions, the foregoing discussion raises the following technical issues:

Field Extension When reasoning about polynomials, it is useful to distinguish the three separate types – those of E, those of F and those of E/F as a supertype. Using blending machinery removes the distinction between these types.

Splitting Field Extension Theorem A challenging but creative step is to discover the theorem that extending F *only* with the roots of $f(x)$ forms a field.

Automorphisms As mentioned in the background section, currently there is no way of computing colimits if automorphisms are characterised in higher-order logic. An alternative specification, or an implementation of colimit computation for higher-order logic is needed.

7 Evaluation and Outlook

7.1 Review of the current offering

(a) We began the paper with the reconstruction of certain mathematical objects, showing the technical feasibility of the approach.

(b) The more advanced example at the centre of the paper illustrates how this sort of reconstruction relates to mathematical practice.

(c) A future-oriented example exposes some technical challenges, while suggesting that blending could offer a novel approach to computer mathematics.

7.2 Broader issues in evaluation

In addition to motivating a further investigation of the role blending can play in proofs, Galois theory, discussed above, is paradigmatic for other reasons. This discussion draws on the early 20th Century writings of Albert Lautman on the philosophy of mathematics and the subsequent interpretation of this work by Gilles Deleuze. It uses these ideas to propose an approach to embedding evaluation within the system itself.

Concerning the common features of Galois theory, class field theory, and the development of the universal covering surface in Riemann geometry, (Lautman, 2011, p. 126) writes:

> What is characteristic of the movement of the theories that will be considered here is the existence of an end conceived in advance as a term of the ascent.

This is reminiscent of our notion of internal evaluation that apply to the blend. To illustrate, let us briefly imagine how we would use blending techniques to move from porcupine+lion to the perfected *porculione*. Here, instead of field automorphisms that preserve mathematical structure

and fix certain designated elements, we would look for mappings that preserve other properties that exist in the underlying domain. Porculiones would presumably have four feet, would be mammals, and would be omnivores; they should also be viable living creatures.

(Deleuze, 1994, pp. 227–228) follows Lautman in enthusiastically endorsing the Galoisian approach to mathematics:

> [T]he fact that an equation cannot be solved algebraically, for example, is no longer discovered as a result of empirical research or by trial and error, but as a result of the characteristics of the groups and partial resolvents which constitute the synthesis of the problem and its conditions (an equation is solveable only by algebraic means – in other words, by radicals, when the partial resolvents are binomial equations and the indices of the groups are prime numbers). The theory of problems is completely transformed and at last grounded, since we are no longer in the classic master-pupil situation where the pupil understands and follows a problem only to the extent that the master already knows the solution and provides the necessary adjunctions. For, as Georges Verriest remarks, the group of an equation does not characterise at a given moment what we know about its roots, but the objectivity of what we do not know about them. Conversely, this non-knowledge is no longer a negative or an insufficiency but a rule or *something to be learnt* which corresponds to a fundamental dimension of the object.

Although there is a commonality between blending and the Galoisian approach insofar as progressive refinement carries us toward a "perfected" conclusion, Deleuze's enthusiasm about the pedagogical situation would be significantly cooled here. It would seem, in many of our examples, that we only make progress "to the extent that the master already knows the solution and provides the necessary adjunctions."

However, this apparent infelicity may be less of a thick obstacle than it would initially appear. What seems to be most needed is a notion of a *question* inside the system. This would recover Lautman's basic thrust: "Scientific or not, every question has built in some assumptions about the form of the answer" (Larvor, 2011). In short, an experimental approach in which the system *asks* and *answers* questions would embed key aspects for evaluation in the system itself.

7.3 Future work

The idea of using blending to carry out steps in a proof would provide a useful training ground for further development. The primary problem is: If blending is the realisation of "combinatorial creativity" how will we avoid being swamped by the combinatorial explosion of possible things to combine? The first challenge is thus fitting different mathematical components together in a sensible manner. A related challenge would apply when modifying the system to selectively experiment with the rules it uses. The objective in this case would be for the system to learn to associate different (useful) techniques with different types of problems.

8 Conclusions and Remarks

The examples presented in this paper trace the development of the blending approach. The current paper begins with reconstructions, but also quickly shows how computed blends can suggest new mathematical definitions and concepts of interest to practising mathematicians. The analysis

offered here shows that this work is a building block that will be useful for future developments that are able to reason more flexibly about mathematical problems – and systematically find and propose new concepts and problems.

In future work, we will look more at the cognitive issues raised in this work. In particular, the use of *image schemas* can give a link between the computational and representational approach taken here, and the cognitive claims coming from authors such as Fauconnier and Turner, and Johnson. Here the work of Mandler and Canovás (2014, 04) and Hedblom, Kutz, and Neuhaus (2014) gives an idea of how these underlying cognitive primitives can be expressed in logical form, and can thus play an explicit role in our modelling of creativity in mathematics.

Acknowledgements

The authors are grateful to the referees for constructive comment. The project COINVENT 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-Open Grant number: 611553.

References

Alexander, J. (2011). 'Blending in mathematics'. *Semiotica*, *2011*(187), 1–48.

Arbib, M. A., and Hesse, M. B. (1986). *The construction of reality*. Cambridge, England: Cambridge University Press.

Astesiano, E. et al. (2002). 'CASL: the common algebraic specification language'. *Theoretical Computer Science*, *286*(2), 153–196.

Boden, M. A. (1977). *Artificial intelligence and natural man*. Harvester Press.

Deleuze, G. (1994). *Difference and repetition*. Translated by Paul Patton. London: Bloomsbury Academic.

Eisenbud, D. (1995). *Commutative algebra with a view toward algebraic geometry*. Graduate Texts in Mathematics. Springer.

Fauconnier, G., and Turner, M. (1998). 'Conceptual integration networks'. *Cognitive Science*, *22*(2), 133–187. Extended version 2001 on-line.

Fauconnier, G., and Turner, M. (2002). *The way we think: conceptual blending and the mind's hidden complexities*. Basic Books.

Goguen, J. A. (1992). 'Sheaf semantics for concurrent interacting objects'. *Mathematical Structures in Computer Science*, 159–191.

Goguen, J. A. (1999). 'An introduction to algebraic semiotics, with application to user interface design'. In *Computation for metaphors, analogy, and agents* (Vol. 1562, pp. 242–291). LNCS. Springer.

Goguen, J. A. (2005). 'What is a concept?' In *Conceptual structures: common semantics for sharing knowledge* (Vol. 3596, pp. 52–57). LNAI. Springer.

Grothendieck, A., and Dieudonné, J. (1971). *Eléments de géométrie algébrique I* (seconde édition). Springer.

Gust, H., Kühnberger, K.-U., and Schmidt, U. (2006). 'Metaphors and heuristic-driven theory projection (HDTP)'. *Theoretical Computer Science*, *354*, 98–117.

Hedblom, M., Kutz, O., and Neuhaus, F. (2014). 'On the cognitive and logical role of image schemas in computational conceptual blending'. In A. Lieto et al. (Eds.), *Proceedings of the second international workshop on artificial intelligence and cognition (AIC 2014)* (Vol. 1315, pp. 110–121). CEUR Workshop Proceedings.

Koestler, A. (1964). *The act of creation*. Hutchinson.

Kutz, O., Neuhaus, F., Mossakowski, T., and Codescu, M. (2014). 'Blending in the Hub – towards a collaborative concept invention platform'. In *Proceedings of the fifth international conference on computational creativity, ICCC 2014*.

Lakoff, G., and Núñez, R. (2000). *Where mathematics comes from: how the embodied mind brings mathematics into being*. New York: Basic Books.

Larvor, B. (2011). 'Albert Lautman: Dialectics in Mathematics'. *Foundations of the Formal Sciences VII*.

Lautman, A. (2011). *Mathematics, ideas and the physical real*. Translated by Simon Duffy. A&C Black.

Mandler, J. M., and Canovás, C. P. (2014). 'On defining image schemas'. *Language and Cognition*, *6*, 510–532.

Martinez, M. et al. (2014). 'Algorithmic aspects of theory blending'. In *12th international conference on Artificial Intelligence and Symbolic Computation*. Sevilla, Spain.

Mossakowski, T., Maeder, C., and Lüttich, K. (2007). 'The Heterogeneous Tool Set'. In O. Grumberg, and M. Huth (Eds.), *Tacas 2007* (Vol. 4424, pp. 519–522). Lecture Notes in Computer Science. Springer.

Núñez, R. (2005). 'Creating mathematical infinities: the beauty of transfinite cardinals'. *Journal of Pragmatics*, *37*(10), 1717–1741.

Pereira, F. C. (2007). *Creativity and artificial intelligence: a conceptual blending approach*. Applications of Cognitive Linguistics. Mouton de Gruyter.

Schmidt, M. (2010). *Restricted higher-order anti-unification for heuristic-driven theory projection* (PICS-Report No. 31-2010). Univ. Osnabrück. Germany.

Steiner, G. (2001). *Grammars of creation*. London: Faber and Faber.

Turner, M. (2005). 'Mathematics and narrative'. In *International conference on mathematics and narrative*. Mykonos, Greece.

Turner, M. (2014). *The origin of ideas: blending, creativity and the human spark*. Oxford: OUP.

Weil, A. (1960). 'De la métaphysique aux mathématiques'. *Sciences*. in (Weil, 1979, pp 408–412).

Weil, A. (1979). *Œuvres scientifiques/collected papers*. Corrected second printing. New York: Springer.

3 Conceptual Blending as a creative meta-generator of mathematical concepts: Prime Ideals and Dedekind Domains as Blend (Article 2)

D. Gomez-Ramirez (2015): Conceptual Blending as a creative meta-generator of mathematical concepts: Prime Ideals and Dedekind Domains as Blend. Proceedings of the 4th International Workshop on Computational Creativity, Concept Invention, and General Intelligence (C3GI) 2015, Publications of the Institute of Cognitive Science (PICS), Vol. 02-2015, 2015.

Abstract:
Conceptual blending is presented as a meta-generator of mathematical concepts, by means of showing, among others, the following examples: The conceptual space of prime ideals over containment-division rings is explicitly presented, by means of an implementation in HETS, as a blend (colimit) of the conceptual space of ideals over a commutative ring with unity and the one of prime numbers over a set with a ”product” operation with neutral element. Besides, a new equivalent form of being a Dedekind domain is presented allowing to express the conceptual space of prime ideals over a Dedekind domain as a blend of the space of ideals over a Noetherian ring and the one of prime numbers as before.

Conceptual Blending as a creative meta-generator of mathematical concepts: Prime Ideals and Dedekind Domains as a Blend

Danny Gomez-Ramirez
Institute of Cognitive Sciences
University of Osnabrück

Abstract

Conceptual blending is presented as a meta-generator of mathematical concepts, by means of showing, among others, the following examples: The conceptual space of prime ideals over containment-division rings is explicitly presented, by means of an implementation in HETS, as a blend (colimit) of the conceptual space of ideals over a commutative ring with unity and the one of prime numbers over a set with a "product" operation with neutral element. Besides, a new equivalent form of being a Dedekind domain is presented allowing to express the conceptual space of prime ideals over a Dedekind domain as a blend of the space of ideals over a Noetherian ring and the one of prime numbers as before.

Introduction

In past years, conceptual blending (see Fauconnier and Turner [6]) has grown in importance in mathematical and logical domains. This cognitive process can be understood as kind of mind's natural way of combining two concepts identifying certain commonalities between them and finally, "blending" them in such a way that a new emergent meaning appears.

The following fundamental notions (blends) are examples of the growing prominence of blending: the integer, rational and real numbers; the Grothendieck group (in general, the algebraic as well as the topological K-theory) and the notions of generic points and motives in algebraic geometry, among others [1].

A more formal approach to concept blending given in terms of colimits in the theory of categories ([8], [9] and [10]), has allowed for the reconstruction of the set of the complex numbers as a blend [12]. Furthermore, by identifying the notion of colimit with the one of pushout, one can reconstruct typical notions of modern algebra like finite dimensional vector spaces or tensor products of algebras as blends.

Moreover, the concepts of prime ideals and containment-division rings (CDR-s) can be also seen as blends [7]. In this paper, we present a complete implementation in HETS [15] of the notion of a prime ideal over a CDR as a blend of the notion of an ideal over a commutative ring with unity and the one of a prime number over a more general algebraic structure as the one of the integers with the product as roughly indicated in [7].

In [7] we present the concept of a prime ideal of a commutative ring with unity ([11] and [5]) as a sort of partial (or weaken) colimit (i.e. a colimit considering some axioms of the input theories) between the concepts of an ideal of a commutative ring with unity (enriched with the collection of all the ideals of the corresponding ring) and the concept of a prime number of the integers.

In order to obtain the desired conceptual space the authors in [7] consider a more general version of the prime numbers, namely, a monoid $(Z,*,1)$ with an "special" divisibility relation $\lfloor$. Besides, the generic space would capture just the syntactic correspondences to be identified in the blending space. This is because, the blend, as a colimit, is essentially the union of the collections of axioms given on each space, but doing at the same time the corresponding syntactic identifications.

By slightly modifying the input conceptual spaces, we obtain, in this paper, one of the most fundamental concepts of algebraic number theory, i.e., the one of Dedekind domain [4, Theorem 37.1], (together with the collection of ideals and prime ideals) as a blend of the concepts of noetherian domains (with the set of ideals and prime ideals) and again a version of the prime numbers in a very elementary form of the integers, but adding the explicit axiom defining the upside-down divisibility relation (see the implementation in section 1). In fact, we present a new equivalent form of Dedekind domains (see §2) based on a containment-division condition, which suggests a new class of commutative rings called containment-division rings (CDR) which are shortly mentioned in [7].

It is worthwhile to mention that the concept of an ideal was discovered by Dededind, after studing the work of Kummer on "ideal numbers" on cyclotomic fields, in order to find a more general ("ideal") entity, generalizing the notion of a number, such that the unique factorization theorem could hold over this new entities on a suitable commutative ring.

1 Implementation for prime ideals over CDR-s as a blend

Firstly, let us define the following class of commutative rings with unity.

Definition 1. *A commutative ring with unity R is a containment-division ring (CDR) if for any two ideals $I,J \subseteq$, it holds that $I \subseteq J$ if and only if J divides I as ideals, i.e., there exists an ideal D such that $I = D**J$ (for the formal definitions of ideals and products of ideals see the corresponding axioms in the implementation below).*

On this section, we reconstruct the conceptual space of prime ideals of a CDR as a blend (colimit in HETS) of the conceptual space of ideals of a commutative ring with unity and the conceptual space of prime elements of a very general version of the integers.

It is also important to note that for this implementation we looked for a minimal set of axioms such that the semantic interpretation can be uniquely determined. It is always possible to construct an implementation with additional axioms given by properties that could be logically derived from the main axioms, (e.g. the set theoretical properties of the containment relation for subsets of a set) but these properties are secondary ones. Meanwhile, the essential ones are those that define the arithmetic of the ring, of an ideal and of the set of ideals of the ring.

```
logic CASL
%%PRIME IDEAS OVER CDR-s AS A BLEND


spec IdealsOfRing =
sort RingElt                %% sort of Ring Elements
```

```
sort SubSetOfRing           %% sort of parts of this ring
pred IsIdeal : SubSetOfRing %% when a subset is an ideal
op  0 : RingElt
op  1 : RingElt
op  __*__ : RingElt * RingElt -> RingElt
op  __+__ : RingElt * RingElt -> RingElt

pred __isIn__ : RingElt * SubSetOfRing

sort Ideal = { I : SubSetOfRing . IsIdeal(I) }
op  R : Ideal        %% the Ring as an ideal
op    __ ** __: Ideal * Ideal -> Ideal, unit R
%%Definition of the predicate of containment

pred __issubsetOf__ : Ideal * Ideal

forall A,B : Ideal
. A issubsetOf B <=> forall a: RingElt. a isIn A => a isIn B

%% axiomatization of a commutative Ring with unity

forall x : RingElt; y : RingElt . x + y = y + x
forall x : RingElt; y : RingElt; z : RingElt
. (x + y) + z = x + (y + z)
forall x : RingElt . x + 0 = x /\ 0 + x = x
forall x : RingElt . exists x' : RingElt . x' + x = 0
forall x : RingElt; y : RingElt . x * y = y * x
forall x : RingElt; y : RingElt; z : RingElt
. (x * y) * z = x * (y * z)
forall x : RingElt . x * 1 = x /\ 1 * x = x
forall x, y, z : RingElt . (x + y) * z = (x * z) + (y * z)
forall x, y, z : RingElt . z * (x + y) = (z * x) + (z * y)

%%axioms for Ideal

forall I: SubSetOfRing. IsIdeal(I) <=>
( forall a,b,c : RingElt
.( (a isIn I => a isIn R)
/\  0 isIn I)
/\ (a isIn I /\ c isIn R => (c * a) isIn I)
/\ (a isIn I /\ b isIn I /\ c isIn R
 /\ b + c = 0 => a + c isIn I ))


%% Definition of the product of ideals without subindexes
```

```
forall A,B: Ideal
. forall a,b: RingElt. (a isIn A /\ b isIn B) => a*b isIn A**B
. forall D: Ideal. (forall a,b: RingElt
. (a isIn A /\ b isIn B) => a*b isIn D)
=> A**B issubsetOf D
end

%%%axioms defining a very simple version of the integers,
%%%considered with an operation * with neutral element,
%%% a binary relation || (upside-down divisibility relation)
%%% and a primality axiom.

spec SimpleInt=
sort SimpleElem
ops 1:  SimpleElem
__ x __: SimpleElem * SimpleElem -> SimpleElem, unit 1
preds __ || __: SimpleElem * SimpleElem
IsPrime : SimpleElem
%Def_upsidedownDivisilityRelation%
forall x,y: SimpleElem
. x || y <=> (exists c: SimpleElem. x = y x c)


%% subsort of primes
sort SimplePrime = { p : SimpleElem . IsPrime(p) }

forall p:SimpleElem .
IsPrime(p) <=>
(forall a,b: SimpleElem
. a x b || p => a || p \/ b || p      %Def_primality%
/\ not (p = 1))
end

%%% Generic space

spec Gen=
sort Generic
ops S: Generic
__ gpr __: Generic * Generic -> Generic, unit S
pred gcont: Generic * Generic
end

view I1: Gen to IdealsOfRing =
Generic |-> Ideal, S |-> R,
__ gpr __ |-> __ ** __, gcont |-> __issubsetOf__
```

```
view I2: Gen to SimpleInt =
Generic |-> SimpleElem, S |-> 1,
__ gpr __ |-> __ x __, gcont |-> __ || __

spec Colimit = combine I1, I2
```

Now, seeing "RingElt" as the sort containing the elements of the ring S, one can obtain, by computing the blend in HETS (as a colimit), the (blend) theory corresponding to the axioms defining a CDR, denoted by S; the set of all its ideals, denoted by Generic; the set of all its prime ideals, denoted by SimplePrime; and a primality predicate, denoted by IsPrime:

```
logic CASL.SulFOL=

sorts Generic, RingElt, SimplePrime, SubSetOfRing
sorts SimplePrime < Generic; Generic < SubSetOfRing
op 0 : RingElt
op 1 : RingElt
op S : Generic
op __*__ : RingElt * RingElt -> RingElt
op __+__ : RingElt * RingElt -> RingElt
op __x__ : Generic * Generic -> Generic
pred IsIdeal : SubSetOfRing
pred IsPrime : Generic
pred __isIn__ : RingElt * SubSetOfRing
pred gcont : Generic * Generic

forall I : SubSetOfRing . I in Generic <=> IsIdeal(I)
%(Ax1)%

forall x : Generic . x x S = x %(ga_right_unit___**__)%

forall x : Generic . S x x = x %(ga_left_unit___**__)%

forall A, B : Generic
. gcont(A, B) <=> forall a : RingElt . a isIn A => a isIn B

%(Ax4)%

forall x, y : RingElt . x + y = y + x %(Ax5)%

forall x, y, z : RingElt . (x + y) + z = x + (y + z)%(Ax6)%

forall x : RingElt . x + 0 = x /\ 0 + x = x %(Ax7)%

forall x : RingElt . exists x' : RingElt . x' + x = 0%(Ax8)%

forall x, y : RingElt . x * y = y * x %(Ax9)%

forall x, y, z : RingElt . (x * y) * z = x * (y * z)%(Ax10)%
```

```
forall x : RingElt . x * 1 = x /\ 1 * x = x %(Ax11)%

forall x, y, z : RingElt
. (x + y) * z = (x * z) + (y * z) %(Ax12)%

forall x, y, z : RingElt
. z * (x + y) = (z * x) + (z * y) %(Ax13)%

forall I : SubSetOfRing
. IsIdeal(I)
 <=> forall a, b, c : RingElt
. ((a isIn I => a isIn S) /\ 0 isIn I)
/\ (a isIn I /\ c isIn S => c * a isIn I)
/\ (a isIn I /\ b isIn I /\
c isIn S /\ b + c = 0 => a + c isIn I)
                                              %(Ax14)%
forall a : RingElt; A : Generic
. a generates A
  <=> forall c : RingElt
. c isIn A => exists d : RingElt . c = a * d
                                                  %(Ax16)%

forall A, B : Generic; a, b : RingElt
. a isIn A /\ b isIn B => a * b isIn A x B        %(Ax17)%

forall A, B, D : Generic
. (forall a, b : RingElt
   . a isIn A /\ b isIn B => a * b isIn D)
  => gcont(A x B, D)                              %(Ax18)%

forall x, y : Generic
. gcont(x, y) <=> exists c : Generic . x = y x c  %(Ax3)%

forall p : Generic . p in SimplePrime <=> IsPrime(p)
%(Ax4_19)%

forall p : Generic
. IsPrime(p)
  <=> (forall a, b : Generic
       . gcont(a x b, p) => gcont(a, p) \/ gcont(b, p))
      /\ not p = S                                %(Ax5_20)%
```

It is worthwhile to mention that the definition of CDR-s was obtained after computing this implementation and observing that the condition given by *(Ax3)* of the former implementation express, after computing the blending, a new non-expected condition, which is exactly the one used in the definition of a CDR. Therefore, it could be seen as a form of "creative" result coming from the blending process.

2 Prime ideals over Dedekind noetherian domains as a blend

The containment-division condition is very close related to the one defining a Dedekind domain, i.e., an integral domain such that every proper ideal can be written as a finite product of ideals [4, Theorem 37.1 and 37.8]. Effectively, if we add the property of being Noetherian [5], then both notions are equivalents:

Theorem 2. *Let R be an integral domain, i.e., a commutative ring with unity without zero divisors. Then the following two conditions are equivalent:*

1. *R is a dedekind domain.*
2. *R is a noetherian CDR.*

Proof. $1 \Rightarrow 2$.

Every Dedekind domain is Noetherian [4, Theorem 37.1]. Besides, the CDR condition is a well-known property of Dedekind domains (see for example [16, Fundamental Theorem of OAK-s]. In fact, for any ideals $I, J \in \mathrm{Id}(R)$, if $I \subseteq J$, then by the unique factorization theorem for ideals in R [4, Theorem 37.11] and by considering the localizations on the prime ideals appearing on their factorizations one sees immediately that J divides I.

$2 \Rightarrow 1$.

Let I be an proper ideal of R. If I is prime, then clearly we can express I as the product of one prime ideal. If not, let P_1 be a prime ideal of R such that $I \subseteq P_1$. Then, due to the fact that R is a CDR, a proper ideal Q_1 such that $I = Q_1 P_1$ exists. Now, if Q_1 is a prime ideal, then we can clearly express I as a product of two ideals. Otherwise, let us choose again a prime ideal P_2 containing Q_1. So, analogously there is another proper ideal Q_2 such that $Q_1 = Q_2 P_2$. If Q_2 is prime, then we can express $I = Q_2 P_2 P_1$ as a finite product of prime ideals. Otherwise, we continue inductively in the same fashion. If after finitely many steps some Q_r is prime, then we can write I as a finite product of prime ideals. In another case, we obtain an ascending chain of ideals

$$Q_1 \subseteq Q_2 \subseteq Q_3 \subseteq \cdots \subseteq Q_n \subseteq \cdots$$

Now, since R is Noetherian, this sequence is stationary (i.e., there exists some $m \in \mathbb{N}$ such that for all $i \geq m$, $Q_i = Q_m$ [2, Proposition 6.2]). Furthermore, $Q_m = Q_{m+1} P_{m+1} = Q_m P_{m+1}$ and so $Q_m = Q_m^i P_{m+1} \subseteq Q_m^i$, for all $i \in \mathbb{N}$. Therefore, I and Q_m are contained in the intersection of all the powers of Q_m, $\cap_{i \geq 1} Q_m^i$, which is the zero ideal due to Krull's Intersection Theorem [5, Corollary 5.4]. In conclusion, $I = (0)$, which is in our case a prime ideal. So, R is a Dedekind domain.

□

As an immediate corollary of this theorem, we see that in the setting of noetherian domains the concept of a CDR is equivalent to the one of a Dedekind domain.

Remark 3. *On the other hand, if R is not a Dedekind domain, but for example a unique factorization domain (UFD), then R is not, in general, a CDR. For example, when $R = \mathbb{Z}[T]$, one can check that the ideals $X = (2)$ and $Y = (2, T)$ gives a counterexample.*

It suggests that in the setting of commutative rings with unity, the class of CDR-s is an intermediate new class of rings.

Now, let us consider as our first conceptual space the space of the implementation called "IdealOfRing" with the extra condition of being a notherian domain, i.e.,

$$(\forall a,b \in R)(ab = 0 \rightarrow a = 0 \vee b = 0),$$

and the noetherian property:

Definition 4. *A commutative ring with unity R is noetherian, if for any ideal A in R, there exists $a_1, \cdots a_n \in A$ such that every element a can be written as a linear combination of these elements: there exists $b_1, \cdots, b_n \in R$ such that $a = b_1 a_1 + \cdots + b_n a_n$.*

On the other hand, let us consider as second conceptual space, the space of the implementation called "SimpleInt". In particular, we include the axiom defining the upside-down divisibility relation:

$$(\forall a,b \in Z)(a \lfloor b \leftrightarrow (\exists c \in Z)(a = cb)).$$

Furthermore, let us choose the same generic space and blend morphisms as in the example of "Prime ideals as blends" presented in [7].

Then, if we do the (total) blend of the corresponding spaces, we obtain the former blend space of the implementation (commutative ring with unity, its set of ideals and prime ideals and a predicate for the prime ideals) plus the stronger condition for the ring S of being a Noetherian domain. Now, when we translate the corresponding version of the former (upside-down divisibility) axiom we obtain

$$(\forall a,b \in G)(a \subseteq b) \leftrightarrow (\exists c \in G)(a = c \cdot_\iota b),$$

where G denotes the set (sort) of ideals of R (see the sort "Generic" in the implementation).

Now, this condition means exactly being a CDR, as in the axiom 3 of the blend space in the implementation.

Therefore, by Theorem 2, we obtain as blend the conceptual space of a Dedekind domain with its collection of ideals and prime ideals and a primality predicate.

It is worthwhile to mention that the concept of an ideal was firstly discovered by Dededind, after studying the work of E. Kummer about "ideal numbers", in order to find a more general ("ideal") entity, generalizing the notion of an integer number, such that the unique factorization theorem on the integers (i.e., the fundamental theorem of arithmetic) could hold over this new "ideal" objects within a suitable number field [3] and [14]. Subsequently, Dedekind domains were defined as the canonical "suitable" rings allowing an unique factorization theorem for ideals.

Finally, one could say that the original invention of Dedekind domains had as main motivating source a metaphorical quest for extending the unique factorization properties coming from the integers.

On the other hand, we recover here again (an equivalent version of) the notion of Dedekind domain (equipped with ideals and prime ideals) just using collections of axioms defining quite more general arithmetical standard properties, not only in the case of the definition of ideals (whose properties are basically the properties defining an abelian group adding an absorption property for the product), but also a very more general concept than the one given by the monoid $(\mathbb{Z}, \cdot, 1)$ of the integers with the product operation,

such that one cannot even derived from these axioms (see the concept "SimpleInt" in the former implementation) the existence of a finite factorization of elements of this structure in terms of the corresponding "prime" elements (see "SimplePrime").

So, the way of re-discovering the notion of a Dedekind domain presented here seems to offer new conceptual sources.

3 Conclusions

Mathematics had evolved as one of the fundamental structural languages use to describe the laws of our physical world. In fact, one could say that almost any modern technological device or tool was born and is based on an specific mathematical framework (e.g. computers, GPS technology, robots, etc...). It implies that the way in which we develop and discover mathematics has necessarily a cognitive component. This perspective was developed in a monist way by Lakoff and Núñes [13].

On the other hand, the presented examples suggest that (the "colimit" approach to) conceptual blending can be seen as a cognitive and universal meta-mathematical procedure, which can be seen as a "meta-generator" of mathematical definitions, which is, omnipresent among most important mathematical fields.

Finally, this contribution is aimed to give new insights in order to understand better how human-level creativity works in the specific field of pure mathematics.

Acknowledgments.

The author would like to thank Alan Smaill and Felix Bou for the help in order to codifying the former implementation. Also, he wishes to thank Ulrich Von der Ohe, Terese Rutkowski and all the members of the COINVENT project.

References

[1] J. Alexander, "Blending in mathematics", *Semiotica*, Volume 187, pp: 1-48, 2003.

[2] M. F. Atiyah and I. G. Macdonald, *Introduction to Commutative Algebra*, Addison-Wesley, Reading, MA, 1969.

[3] N. Bourbaki *Commutative Algebra 1-7*, Springer Verlag, 1989.

[4] Eds. A. Coleman and P. Ribenboim. "Multiplicative Ideal Theory", *Quenn's Papers in Pure and Applied Mathematics*, Ontario, Canada, 1992.

[5] D. Eisenbud, *Commutative Algebra with a View Toward Algebraic Geometry*, Springer-Verlag, 1995.

[6] G. Fauconnier and M. Turner *The Way We Think*, Basic Books, New York, 2003.

[7] F. Bou, J. Corneli, D. A. J. Gomez-Ramirez, E. Maclean, A. Pease, M. Schorlemmer and A. Smaill, "The role of blending in mathematical invention", *accepted in ICCC'15*, Ohio, USA, 2015.

[8] J. Gougen, "An introduction to algebraic semiotic with application to user interface design. In *Computation for methaphors, analogy and agents*, edited by C. L. Nehaniv. volume 1562, pp- 242-291. 1999.

[9] J. Gougen. "Towards a design theory for virtual worlds: algebraic semiotics and scientific visualization as a case study". In *Proceedings Conference on Virtual Worlds and Simulation (Phoenix AZ, 7-11 January 2001)*, edited by C. Landauer and K. Bellman. pp: 298-303, Society for Modelling and Simulation, 2001.

[10] J. Gougen. "Steps towards a design theory for virtual worlds" In *Developing future interactive systems*, edited by M. Sanchez-Segura, Idea Group publishing, pp: 116-128, 2003.

[11] A. Grothendieck and J. Dieudonné. *Eléments de Géométrie Algébrique* I. Springer, 1971.

[12] J. Fleuriot, E. Maclean, and D. Winterstein, “Reinventing the complex numbers”, in *Workshop at ECAE*, edited by T. Besold, A. Smaill others, volume 1 of PICS, Prague, 2014.

[13] G. Lakoff and R. Núñes, *Where Mathematics Comes From*, Basic Books, 2000.

[14] F. Lemmermeyer, "Jakobi and Kummer’s ideal numbers", in Arxiv: http://arxiv.org/pdf/1108.6066.pdf.

[15] C. Maeder, T. Mossakowski and M. Codescu. "Hets user guide version 0.99." *http://www.informatik.uni-bremen.de/agbkb/forschung/formal methods/CoFI/hets/UserGuide.pdf*, 2014.

[16] E. Weiss *Algebraic Number Theory*, McGraw-Hill, 1963.

4 Generating Fundamental Notions of Fields and Galois Theory through Formal Conceptual Blending (Article 3)

D. Gomez-Ramirez (under review): Generating Fundamental Notions of Fields and Galois Theory through Formal Conceptual Blending. Submitted to Journal of Logical and Algebraic Methods in Programming.

Abstract:
Using the formalization of conceptual blending given in terms of colimits of manysorted first order theories due to J. Gougen, we show explicitly how to generate recursively in HETS fundamental concepts of Fields and Galois theory like the ones of field, field extension, group of a automorphisms of a field and Aut(E/F), as blends starting from five basic structural concepts. This implies that formal conceptual blending is able to play the role of a meta-generator of mathematical concepts from a theoretical and computational point of view; specifically, in (automatic) theorem proving and in meta-mathematics, respectively.

Generating Fundamental Notions of Fields and Galois Theory through Formal Conceptual Blending

Danny Gomez-Ramirez
Institute of Cognitive Sciences
University of Osnabrück

Abstract

Using the formalization of conceptual blending given in terms of colimits of many-sorted first order theories due to J. Gougen, we show explicitly how to generate recursively in HETS fundamental concepts of Fields and Galois theory like the ones of field, field extension, group of a automorphisms of a field and $Aut(E/F)$, as blends starting from five basic structural concepts. This implies that formal conceptual blending is able to play the role of a meta-generator of mathematical concepts from a theoretical and computational point of view; specifically, in (automatic) theorem proving and in meta-mathematics, respectively.

Introduction

Over the last twenty years conceptual blending has gained more and more attention and recognition as a fundamental human mind's ability for merge concepts and ideas in a wide range of disciplines like linguistics, computer sciences, arts, literature, physics and mathematics, among others [7], [5], [6]. In fact, there is a current interdisciplinary research project called COINVENT (Concept Invention Theory), supported by the European Commission pursuing to give a concrete formal (theoretical and computational) model of human's concept creation, based on Fauconnier and Turner's theory of conceptual blending [21].

Particularly, Alexander has stressed indirectly a sort of 'omnipresence' of blending in pure mathematics. For example, he indicates a fundamental role of blending for obtaining so basic notions of modern mathematics starting from the integer, rational, irrational and imaginary numbers; and finishing with highly abstract concepts like generic points, motives and k-theory [1].

Furthermore, modern formalizations of conceptual blending coming from the computer siences ([9], [10] and [11]) has allowed to model in a more accurate way how conceptual blending can 'generate' concepts like the complex numbers [13], simple versions of the integers and the notion of prime ideals over commutative ring with unity [3]. Even, by computing the specification of the prime ideal example as a blend, a new kind of commutative rings was suggested indirectly by the system HETS [18] (i.e. the class of containment-division rings) which turns out to be equivalent to the notion of Dedekind domain in the finitely generated setting [8].

Besides, in [8] was suggested explicitly that conceptual blending can be seen as a 'meta-generator' of mathematical concepts. More precisely, that means that starting from some seminal and elementary notions one could basically obtained any mathematical concept as a finite iteration of 'suitable' blends of such seminal concepts. For example, one of the most fundamental notion of algebraic number theory, i.e., Dedekind domain joint

with its prime spectra is obtained as a blend of the notions of ideal over a Noetherian ring and the one of prime element on a quasi-monoid [8, §2].

Here, we are going to consider as a study case, as suggested in [3], Fields and Galois theory [14] to show explicitly how to generate fundamental concepts of these theories as blends of few elementary mathematical concepts as the ones of (abelian) group, pointed (abelian) group, action of a group on a set, space of fixed points, algebraic substructure and distributive space. Some of these concepts are old and very well-known in the mathematical community and other ones are more implicitly used by the working mathematician and they emerge in a natural way on the process of generating such a basic concepts of Fields and Galois theory through conceptual integration.

This work has as one of its objects start to enlighten and to offer evidence concerning the following fundamental questions in modern meta-mathematics: is it possible to meta-model and to meta-generate a considerably 'huge' collection of mathematical theories (e.g. mathematical concepts and theorems) by modeling the most important cognitive processes discovered in cognitive sciences within the last decades, as for example: analogical and case base reasoning, metaphor, particularization, generalization and conceptual blending? And, what kind of concrete computational limitations could the incompleteness theorems offer to the former meta-research project?

Now, we can see the examples developed in this work as seminal formal evidence that some of the current formalization of one of these primary cognitive processes (i.e. conceptual blending) is, in fact, being able to formally generate high abstract mathematical concepts starting from very basic ones. Similarly, as the natural numbers can be generated from the prime numbers by means of combining them through a formal process called 'product'.

1 Basic Terminology

We will specify our concepts in many-sorted first order logic [16]. Here, a concept is given by a signature $\Sigma = (S, F, R)$, where S is a set of sorts, F is a set of functional symbols, each of them carrying a finite set of symbols of S specifying the n-tuple sort of the domain and the sort of the codomain (constants are just functional symbols with empty domain), and R is a set of symbols for relations with the corresponding m-tuple sort.

Besides, a concept would have a finite set of sentences A in many-sorted first order logic, called the axioms of the concept.

Finally, an interpretation M of a concept is just a collection of sets M_S, functions M_F and relations M_R, with elements indexed by the corresponding sets S, F and R, respectively; such that if M_g is an *interpretation* of $g : s_1 \times s_2 \times \cdots \times s_n \to s_m$, then $M_g : M_{s_1} \times M_{s_2} \cdots \times M_{s_n} \to M_{s_m}$.

Similarly for the interpretations of the symbols in R.

Now, for simplicity we define the class of models of a concept as big as possible, i.e., given the signature Σ and the finite set of axioms A, we define the class of models of the concept defined by (Σ, A) as the class $\mathbb{M}$ of all interpretations M, such that the interpretation of any axiom of A is, in fact, true over Σ, i.e. $\mathbb{M} \models_\Sigma A$. Here, we adopt the standard definition of satisfaction in many-sorted first order logic. Furthermore, another way to express that is saying that the class of models of a concept is just the dual of its set of

axioms A into the class of all possible interpretations. This formalization allows us to say, by definition, that two concepts C_1 and C_2 are equivalent if their corresponding class of models coincide, i.e., if $\mathbb{M}_1 = \mathbb{M}_2$.

In conclusion, a concept for us consists of a triple $C = (\Sigma, A, \mathbb{M})$.

Let us define the notion of a morphism of concepts: if $C_1 = (\Sigma_1, A_1, \mathbb{M}_1)$ and $C_2 = (\Sigma_2, A_2, \mathbb{M}_2)$ are concepts then a morphism $\phi : C_1 \to C_2$ is just a triple

$$\phi = (\phi_{\Sigma_1} : \Sigma_1 \to \Sigma_2, \phi_{F_1} : F_1 \to F_2, \phi_{R_1} : R_1 \to R_2),$$

such that the translation of the axioms of C_1 into C_2 induced by ϕ, i.e., $\phi(A)$, are deducible from the axioms A_2, that means $A_2 \models_{\Sigma_2} \phi(A_1)$.

Now, it is a well-known fact that the collection of concepts with their morphisms forms a category $\mathbb{C}oncepts$. Moreover, in this category any V-shaped diagram, $\alpha : G \to C_1$ and $\beta : G \to C_2$ has a colimit [17].

Now, for such a V-shaped diagram, we will say following the formalization of Gougen ([9], [10] and [11]) that its colimit B is the *blending* of the concepts C_1 and C_2 regarding to (the identifications codified by) the concept G (through α and β).

Now, let us use the notation $B = C_1 \vee_G C_2$, coming from the wedge sum of topological spaces (which is a particular case of a colimit construction), but adding the G as sub-index in order to point out that the identifications of the two spaces is done among the generic space [20].

2 Structural Concepts

In order to obtain the fundamental concepts of Fields and Galois theory such as the ones of field, field extension, group of automorphisms of a field and $\mathrm{Aut}_F E$ (where E/F is a field extension) as a blend, we present the basic 'structural' concepts required as conceptual 'bricks' for constructing the whole 'building' of Fields and Galois Theory.

We give the explicit definitions of each of the structural concepts, which are used, at least, implicitly, in several areas of modern mathematics. More precisely, some of them are explicitly very well-known concepts by the working mathematician, e.g. (abelian) group, action of a group on a set and fixed point space; whereas other ones are often implicitly used in modern mathematics without receiving until now any kind of special name, e.g. bigroup, pointed (abelian) group and algebraic (bi)substructure.

Definition 1. An abelian group is a set A with a binary operation $+$ and a special (neutral) element $0 \in A$ such that the following axioms hold:

1. $(\forall a \in A)(a + 0 = 0 + a = a)$.
2. $(\forall a \in A)(\exists b \in A)(a + b = b + a = 0)$.
3. $(\forall a, b, c \in A)((a + b) + c = a + (b + c)))$.
4. $(\forall a, b \in A)(a + b = b + a)$.

A is a group if it fulfills conditions 1.-3.

Definition 2. A pointed (abelian) group is a set B with an binary operation $*$ and a distinguished element $b \in B$ such that $(B \setminus \{b\}, *_{|B\setminus\{b\}\times B\setminus\{b\}})$ is an (abelian) group and, $b*c = c*b = b$ for all $c \in B$.

Well-known examples of pointed abelian groups are the integer, rational, real and complex numbers with the zero element and the product operation, respectively. Moreover, for any non-empty set with a distinguished element there exists at least one structure of pointed group for it. In fact, it could be shown that this statement is equivalent to the axiom of choice [12].

Definition 3. A distributive space consists of two sets D y K with two operations $\oplus : D \times D \to D$ and $\otimes : K \times D \to D$ such that

$$\forall x \in K \forall y,z \in D(x \otimes (y \oplus z) = (x \otimes y) \oplus (x \otimes z)).$$

Instances of spaces of distribution are boolean algebras, the space of square matrices with entries over a field (e.g. the real or complex numbers) and the standard sum and product operations; and clearly the natural, integer, rational, real and complex numbers with addition and multiplication, respectively. In all these cases, $D = K$.

On the other hand, we obtain also an example of a distributive space if $(D, \oplus)$ is a vector space over a field K and $\otimes$ denotes the corresponding scalar product. Now, if $\dim D > 1$, then clearly $D \neq K$.

Definition 4. An action of a group $(G, +, 0)$ on a set X is simply a function $* : G \times X \to X$ such that the following two conditions holds:

1. $(\forall a,b \in G)(\forall x \in X)((a+b)*x = a*(b*x))$.
2. $\forall x \in X(0*x = x)$.

Definition 5. An algebraic substructure $\mathbb{S} = ((A, +_A, 0_A), (B, +_B, 0_B), i : A \to B)$, consists with two sets with binary operations defined over each of them and special constants, such that i fulfills the following properties:

1. i is an homomorphism: $i(0_A) = 0_B$ and $\forall x,y \in A(i(x +_A y) = i(x) +_B i(y))$.
2. i is injective: $\forall x,y \in A(i(x) = i(y) \Rightarrow x = y)$.
3. $\forall x \in B \forall y \in A((x +_B i(y) = 0_B) \Rightarrow \exists z \in A(i(z) = x))$.

The last condition can be rephrase as follows: the 'potential inverses' of elements of A in B are, in fact, again in A.

Usual examples of algebraic substructures are given by the natural injections $i_1 : \mathbb{Z} \to \mathbb{Q}$, $i_2 : \mathbb{Q} \to \mathbb{R}$ and $i_3 : \mathbb{R} \to \mathbb{C}$ (as well as the remaining meaningful combinations) with the addition operation and the zero element, respectively.

The main intuition with this definition is that when $(A, +_A, 0_A)$ has additionally an algebraic stucture, as the one of a monoid, a semi-group or a group; then $(B, +_B, 0_B)$ would inherits automatically exactly the same structure. This definition is a stronger notion as the one of embedding (i.e. an injective morphism) commonly used in the mathematical literature, since, in principle, the sets A and B have a very basic algebraic structure, e.g., we

do not even require associativity for the corresponding operations. However, we impose the typical conditions for an embedding in 1) and 2) and additionally, we request condition 3) for including potential inverses of the smaller structure into itself. Now, if we restrict ourselves to the category of monoids, semi-group and groups, then, these two notions coincide, because we can prove that under these hypothesis 3) would follow from 1) and 2).

Definition 6. if X denotes a set and F is a collection of functions from X to X, then a subset Y of X is called the space of fixed points of F, if

$$\forall x \in X((\forall f \in F)(f(x) = x) \leftrightarrow x \in Y).$$

Typical examples of fixed point spaces appear in topology when one should verify that a particular topological space is a fixed-point space and in the setting of retractions between topological spaces [19].

Finally, the concepts described on this section were obtained by trying to decompose the four former ones, coming from Fields and Galois theory, into their 'minimal' conceptual building blocks. In particular, the concept of algebraic substructure appears as a simple natural solution to the question of decomposing the concept of field extension as a blend of the notion of field and a coherent additional concept. In fact, as it is shown later, this particular concept of algebraic substructure is a more general version of the notion of embedding, which seems to be category independent, since there is no specific intrinsic condition imposed to the corresponding binary operations, not even associativity. So, this process of formally decomposing (mathematical) concepts can also lead to the discovery of new seminal concepts, which allows to understand in a more general as well as practical way some of the classical concepts used by the working mathematician (e.g. embedding).

3 Generating the Fundamental Concepts of Field and Galois Theory

3.1 Fields

On this section we use the system HETS [18] in the language of CASL [2] in order to compute the colimits of the former concepts.

Definition 7. A bigroup is a set Q with two binary operations $+$ and $*$ such that $(Q,+,0)$ is an abelian group and $(Q,*)$ is a pointed abelian group with distinguished element 0.

Examples of bigroups are the rational, real and complex numbers with the standard operations and the zero element as distinguished constant in any case. In fact, let us show a concrete example of a bigroup which is not a field. Let us define on the group $(R = \mathbb{Z}/4\mathbb{Z},*)$, the following second binary operation $*$: $a*b=0$, if either $a=0$ or $b=0$. For the subset $R' = R\setminus\{0\}$, we define $*$, in terms of the following bijection $\phi : \mathbb{Z}/3\mathbb{Z} \to R'$, defined by $\phi(0)=3, \phi(1)=1$ and $\phi(2)=2$. It is straightforward to show that $(R,+,*)$ is a bygroup. However, $1*(1+2)=1*3=3$ and $1*1+1*2=2+3=1$, thus $1*(1+2)\neq 1*1+1*2$. So, R is not a field. We use here this concept of bigroup as an intermediate concept in order to obtain the notion of a field as a 'fusion' of some of the former structural concepts.

Here it is relevant to give an intuitive account of the concrete form of a blending within this category of concepts in many-sorted first order logic. Typically, one starts with two input concepts, defined by a finite collection of symbols for sorts, functions, relations; and a finite collection of axioms. Secondly, one identifies the symbols to be identified (blended) into the colimit, and then, one 'codifies' by hand this identifications by defining a suitable generic space and two morphisms from this spaces into the input spaces. These morphisms describe the way in which one wants to do the identifications. At the end, one obtains as colimit space the collection of all the axioms from the two input spaces written using the 'new' symbols induced by the generic space via the two corresponding morphisms. That is basically what HETS' command *combine* does.

Now, we show firstly how to obtain the concept of bigroup as a blend of the concepts of an abelian group and a pointed abelian group, i.e., $BiGroup = AbGroup \vee_G PoinAbGroup$. Let us show how to specify the input spaces, the generic space and the blending morphisms:

```
logic CASL

spec GROUP=
     sort Element
     ops 0,f: Element;
         __ + __: Element * Element -> Element
     sort Elem2= { x: Element. not x = 0 }
%% Axioms of an Abelian Group with at least two elements.
        forall x : Element; y : Element . x + y = y + x
        forall x : Element; y : Element; z : Element
. (x + y) + z = x + (y + z)
        forall x : Element . x + 0 = x /\ 0 + x = x
        forall x : Element . exists x' : Element . x' + x = 0
        . not (f = 0)

end
%% Axioms of a pointed abelian group
spec PUNGROUP=
     sort ElemPlus
     op   zero: ElemPlus;
     sort Elem3={ x: ElemPlus. not x = zero }
     ops 1: Elem3;
         __ * __: ElemPlus * ElemPlus -> ElemPlus

       forall x : Elem3; y : Elem3 . x * y = y * x
       forall x : Elem3; y : Elem3; z :Elem3
. (x * y) * z = x * (y * z)
       forall x : Elem3 . x * 1 = x /\ 1 * x = x
       forall x : Elem3 . exists x' : Elem3 . x' * x = 1
       forall x : ElemPlus . x * zero = zero
/\ zero * x = zero
end

spec GENERIC =
     sort Elem
     op   elt: Elem
```

```
     sort SubElem={ x : Elem . not x = elt }
end

view I1:
     GENERIC to GROUP =
     Elem |-> Element, elt |-> 0, SubElem |-> Elem2
end

view I2:
     GENERIC to PUNGROUP =
     Elem |-> ElemPlus, elt |-> zero, SubElem |-> Elem3
end

spec Colimit = combine I1, I2
```

After computing the colimit we obtain exactly the concept of a bigroup:

```
logic CASL.SulFOAlg=

sorts Elem, SubElem
sorts SubElem < Elem
op 1 : SubElem
op __*__ : Elem * Elem -> Elem
op __+__ : Elem * Elem -> Elem
op f : Elem
op zero : Elem
forall x : Elem . x in SubElem <=> not x = zero %(Ax1)%
forall x, y : Elem . x + y = y + x %(Ax2)%
forall x, y, z : Elem . (x + y) + z = x + (y + z) %(Ax3)%
forall x : Elem . x + zero = x /\ zero + x = x %(Ax4)%
forall x : Elem . exists x' : Elem . x' + x = zero %(Ax5)%
. not f = zero %(Ax6)%
forall x, y : SubElem . x * y = y * x %(Ax2_8)%
forall x, y, z : SubElem . (x * y) * z = x * (y * z) %(Ax3_9)%
forall x : SubElem . x * 1 = x /\ 1 * x = x %(Ax4_10)%
forall x : SubElem . exists x' : SubElem . x' * x = 1 %(Ax5_11)%
forall x : Elem . x * zero = zero /\ zero * x = zero %(Ax6_12)%
```

Second, we show how to obtain the concept of field as a blend of the concepts of bigroup and a distributive space, i.e.,

$$Field = BiGroup \vee_{G_1} DistSpace = (AbGroup \vee_G PoinAbGroup) \vee_{G_1} DistSpace.$$

The specification of the concepts and morphisms is the following:

```
logic CASL

spec BIGROUP=
     sort Element
     ops 0,1: Element;
         __ + __: Element * Element -> Element
         __ * __: Element * Element -> Element
```

```
     sort Elem2= { x: Element. not x = 0 }
        forall x : Element; y : Element . x + y = y + x
        forall x : Element; y : Element; z : Element
. (x + y) + z = x + (y + z)
        forall x : Element . x + 0 = x /\ 0 + x = x
        forall x : Element . exists x' : Element . x' + x = 0
        . not (1 = 0)
       forall x : Element; y : Element . x * y = y * x
       forall x : Element; y : Element; z :Element
. (x * y) * z = x * (y * z)
       forall x : Element . x * 1 = x /\ 1 * x = x
       forall x : Elem2 . exists x' : Elem2 . x' * x = 1
end

spec DISTRISPACE=
     sort Elemdistr
     ops __ oo __: Elemdistr * Elemdistr -> Elemdistr
         __ && __: Elemdistr * Elemdistr -> Elemdistr
         forall p : Elemdistr; q : Elemdistr; r : Elemdistr
. p && (q oo r) = (p && q) oo (p && r)

end

spec GEN=
     sort Elem
     ops __ + __: Elem * Elem -> Elem
         __ * __: Elem * Elem -> Elem

end

view I1:
     GEN to BIGROUP =
     Elem  |-> Element, __ + __ |-> __ + __,
__ * __ |-> __ * __
end

view I2:
     GEN to DISTRISPACE =
     Elem  |-> Elemdistr, __ + __ |-> __ oo __,
__ * __ |-> __ && __
end

spec Colimit = combine I1, I2
```

After doing the computation of the colimit we obtain the classic concept of a field:

```
logic CASL.SulFOAlg=

sorts Elem, Elem2
sorts Elem2 < Elem
op 0 : Elem
```

```
op 1 : Elem
op __*__ : Elem * Elem -> Elem
op __+__ : Elem * Elem -> Elem
forall x : Elem . x in Elem2 <=> not x = 0 %(Ax1)%
forall x, y : Elem . x + y = y + x %(Ax2)%
forall x, y, z : Elem . (x + y) + z = x + (y + z) %(Ax3)%
forall x : Elem . x + 0 = x /\ 0 + x = x %(Ax4)%
forall x : Elem . exists x' : Elem . x' + x = 0 %(Ax5)%
. not 1 = 0 %(Ax6)%
forall x, y : Elem . x * y = y * x %(Ax7)%
forall x, y, z : Elem . (x * y) * z = x * (y * z) %(Ax8)%
forall x : Elem . x * 1 = x /\ 1 * x = x %(Ax9)%
forall x : Elem2 . exists x' : Elem2 . x' * x = 1 %(Ax10)%
forall p, q, r : Elem
. p * (q + r) = (p * q) + (p * r)      %(Ax1_11)%
```

3.2 Field Extensions

Now, let us blend two times the concept of an algebraic substructure, identifying a minimal amount of sorts, in order to obtain a concept which we call *algebraic bisubtructure*, necessary for getting the concept of a field extension in the next step. The concrete implementation of $AlgBiSubstruc = AlgSubStruc \vee_{G_2} AlgSubStruc$ is the following:

```
logic CASL
%% the concept of a algebraic bisubstructure as a trivial blend
%% of two times the concept of an algebraic substructure
spec SUBALGSTRUC =
     sort SubAlgStruc
     sort AlgStruc
     op 0 : AlgStruc
     op zero : SubAlgStruc
     op __+__ : AlgStruc * AlgStruc -> AlgStruc
     op __++__ : SubAlgStruc * SubAlgStruc -> SubAlgStruc
     ops inj : SubAlgStruc -> AlgStruc

forall x, y : SubAlgStruc . inj(x) = inj(y) => x = y
. inj(zero)=0
forall x, y : SubAlgStruc . inj(x ++ y) = inj(x) + inj(y)
forall a : AlgStruc; b : SubAlgStruc
. a + inj(b) = 0 => exists c : SubAlgStruc . a = inj(c)
end

spec SUBALGSTRUC2 =
     sort SubAlgStruc2
     sort AlgStruc2
     op 1 : AlgStruc2
     op one : SubAlgStruc2
     op __*__ : AlgStruc2 * AlgStruc2 -> AlgStruc2
     op __**__ : SubAlgStruc2 * SubAlgStruc2 -> SubAlgStruc2
     ops inj2 : SubAlgStruc2 -> AlgStruc2

 forall x, y : SubAlgStruc2 . inj2(x) = inj2(y) => x = y
```

```
     . inj2(one)=1
     forall x, y : SubAlgStruc2
. inj2(x ** y) = inj2(x) * inj2(y)
     forall a : AlgStruc2; b : SubAlgStruc2
 . a * inj2(b) = 1 =>
exists c : SubAlgStruc2 . a = inj2(c)
end

spec GENERIC =
     sort GSubAlgStruc
     sort GAlgStruc
     op Ginj: GSubAlgStruc -> GAlgStruc
end

view I1:
     GENERIC to SUBALGSTRUC =
     GSubAlgStruc |-> SubAlgStruc, GAlgStruc |-> AlgStruc,
Ginj |-> inj
end

view I2:
     GENERIC to SUBALGSTRUC2 =
     GSubAlgStruc |-> SubAlgStruc2, GAlgStruc |-> AlgStruc2,
Ginj |-> inj2
end

spec Colimit = combine I1, I2
```

Now, the blending space is basically a subset (GSUBALGSTRUC) of the base space (GALGSTRUC) having simultaneously two algebraic substructures given by two binary operations. The specific list of axioms of this space is found in the next implementation under the name ALGBISUBSTRUC. In fact, the notation '$Ax*\_*$' in preserved in such a list exactly as HETS displays it when one asks for the theory of the colimit node.

So, we obtain now a particular list of axioms defining the concept of field extension as a blend of the concepts of field and algebraic bisubstructure, that is, $FieldExt = Field \vee_{G_3} BiSubAlgStruc$:

```
logic CASL
%% Generating the concept of a field extension as a blend of the
%% concepts of field and bialgebraic sub-structure.

spec FIELD =
     sort Elem
     op 0 : Elem
     op 1 : Elem
     op __+__ : Elem * Elem -> Elem
     op __*__ : Elem * Elem -> Elem

%%Axioms of a field

 forall x : Elem; y : Elem . x + y = y + x %%Commutativity with +
 forall x : Elem; y : Elem; z : Elem
```

```
 . (x + y) + z = x + (y + z) %%Associativity with +
 forall x : Elem . x + 0 = x /\ 0 + x = x%%unit with +
 forall x : Elem . exists x' : Elem . x' + x = 0 %%Inverse with +
 forall x : Elem; y : Elem . x * y = y * x%%Commutativity with *
 forall x : Elem; y : Elem; z : Elem
 . (x * y) * z = x * (y * z) %%Associativity with *
 forall x : Elem . x * 1 = x /\ 1 * x = x%%unit with *
 forall x : Elem .
 not (x = 0) => exists w : Elem . x * w = 1 %%Inverse with *
 forall x, y, z : Elem
. (x + y) * z = (x * z) + (y * z) %%Distributivity

end
%% Axioms of a algebraic bisubstructure

spec ALGBISUBSTRUC =
sorts GAlgStruc, GSubAlgStruc
op 0 : GAlgStruc
op 1 : GAlgStruc
op __*__ : GAlgStruc * GAlgStruc -> GAlgStruc
op __**__ : GSubAlgStruc * GSubAlgStruc -> GSubAlgStruc
op __+__ : GAlgStruc * GAlgStruc -> GAlgStruc
op __++__ : GSubAlgStruc * GSubAlgStruc -> GSubAlgStruc
op inj2 : GSubAlgStruc -> GAlgStruc
op one : GSubAlgStruc
op zero : GSubAlgStruc

forall x, y : GSubAlgStruc . inj2(x) = inj2(y) => x = y
%%(Ax1)%
. inj2(zero) = 0 %%(Ax2)%
forall x, y : GSubAlgStruc . inj2(x ++ y) = inj2(x) + inj2(y)
%%(Ax3)%
forall a : GAlgStruc; b : GSubAlgStruc
. a + inj2(b) = 0 => exists c : GSubAlgStruc . a = inj2(c)
%%(Ax4)%
. inj2(one) = 1 %(Ax2_6)%
forall x, y : GSubAlgStruc . inj2(x ** y) = inj2(x) * inj2(y)
%%(Ax3_7)%
forall a : GAlgStruc; b : GSubAlgStruc
. a * inj2(b) = 1 => exists c : GSubAlgStruc . a = inj2(c)
%%(Ax4_8)%
end

spec GENERIC =
        sort Gen
        op 0 : Gen
        op 1 : Gen
        op __+__ : Gen * Gen -> Gen
        op __*__ : Gen * Gen -> Gen
end

view I1:
```

```
     GENERIC to FIELD =
     Gen |-> Elem, __ + __ |-> __ + __ ,
__ * __ |-> __ * __ ,0 |-> 0, 1 |-> 1
end

view I2:
     GENERIC to ALGBISUBSTRUC =
     Gen |-> GAlgStruc, __ + __ |-> __ + __,
__ * __ |-> __ * __, 0 |-> 0, 1 |-> 1
end

spec Colimit = combine I1, I2
```

After computing the colimit we obtain an equivalent version of the concept of field extension, with very few axioms:

```
logic CASL.FOAlg=

sorts Gen, SubAlgStruc
op 0 : Gen
op 1 : Gen
op __*__ : Gen * Gen -> Gen
op __**__ : SubAlgStruc * SubAlgStruc -> SubAlgStruc
op __+__ : Gen * Gen -> Gen
op __++__ : SubAlgStruc * SubAlgStruc -> SubAlgStruc
op inj : SubAlgStruc -> Gen
op one : SubAlgStruc
op zero : SubAlgStruc
forall x, y : Gen . x + y = y + x %(Ax1)%
forall x, y, z : Gen . (x + y) + z = x + (y + z) %(Ax2)%
forall x : Gen . x + 0 = x /\ 0 + x = x %(Ax3)%
forall x : Gen . exists x' : Gen . x' + x = 0 %(Ax4)%
forall x, y : Gen . x * y = y * x %(Ax5)%
forall x, y, z : Gen . (x * y) * z = x * (y * z) %(Ax6)%
forall x : Gen . x * 1 = x /\ 1 * x = x %(Ax7)%
forall x : Gen . not x = 0 => exists w : Gen . x * w = 1
%(Ax8)%
forall x, y, z : Gen . (x + y) * z = (x * z) + (y * z)
%(Ax9)%
forall x, y : SubAlgStruc . inj(x) = inj(y) => x = y
%(Ax1_10)%
. inj(zero) = 0 %(Ax2_11)%
. inj(one) = 1 %(Ax3_12)%
forall x, y : SubAlgStruc
. inj(x ++ y) = inj(x) + inj(y)
/\ inj(x ** y) = inj(x) * inj(y)
%(Ax4_13)%
forall a : Gen; b : SubAlgStruc
. a + inj(b) = 0 => exists c : SubAlgStruc . a = inj(c)
%(Ax5_14)%
forall a : Gen; b : SubAlgStruc
. a * inj(b) = 1 => exists c : SubAlgStruc . a = inj(c)
%(Ax6_15)%
```

3.3 Group of Automorphisms of a Field

We want to specify spaces of functions from a set X into itself in order to be able to manage concepts like the group of automorphisms of a field extension E/F, fixing the base field F, i.e. $\mathrm{Aut}_F E$. For this purpose we need to define a new sort $Func$, carrying implicitly such a space of functions: It is because HETS works with a language of many-sorted first order logic. Besides, we are able to specify notions like the image of a function f on a point $x \in X$ just by defining an operation $eval : Func \times X \to X$.

In this way we are going to express this kind of spaces of functions. Of course, when we talk about the collection of models for these spaces, then spaces of functions (in Zermelo-Fraenkel Set Theory) would be a particular prototype of models, but, in principle, there could be additional kinds of model as well.

However, from any kind of model fulfilling a concept C involving an evaluation function $eval : Func \times X \to X$, we can always recover an interpretation of any element $f \in Func$ as a function $f : X \to X$ in the classical sense, by means of defining $f(x) := eval(f,x)$, for any $x \in X$. So, in this sense we are going to interpret this kind of concepts.

Our first goal is to generate the space $\mathrm{Aut}L$ of automorphisms of a field L as a blending of structural concepts. Thus, let us give an explicit definition of this concept.

Definition 8. Let $(L,+,*,0,1)$ be a non-trivial field. A function $f : L \to L$ is an automorphism of L if the following conditions hold:

1. $\forall a,b \in L f(a+b) = f(a)+f(b) \wedge f(a*b) = f(a)*f(b)$.
2. f is a bijection.

The set of automorphisms of L is denoted by $\mathrm{Aut}L$ and it has the structure of a group with the operation of composition of functions $\circ$ and with the identity on L as neutral element.

It is straightforward to verify that in the case of non-trivial fields (i.e., $0 \neq 1$), an automorphisms f, so defined, fulfills besides $f(1) = 1$ (that means in conclusion that f is an homomorphism of commutative rings with unity) and its inverse function is also an automorphism. Besides, it is also clear that one can express the two former conditions of automorphisms in terms of the evaluation function $eval : \mathrm{Aut}L \times L \to L$. Explicitly, let us write down a concrete specification of this space in the language of CASL:

```
sorts Elems, Automor
op Id : Automor
op __eval__ : Automor * Elems -> Elems
op __o__ : Automor * Automor -> Automor
op __plus__ : Elems * Elems -> Elems
op __times__ : Elems * Elems -> Elems
op one : Elems
op zero : Elems

forall x, y, z : Automor . (x o y) o z = x o (y o z)
forall x : Automor . x o Id = x /\ Id o x = x
forall x : Automor . exists x' : Automor . x' o x = Id
forall f, g : Automor; a : Elems
. (f o g) eval a = f eval (g eval a)
forall a : Elems . Id eval a = a
```

```
forall n : Automor; a, b : Elems
. n eval (a plus b) = (n eval a) plus (n eval b)
forall x, y : Elems . x plus y = y plus x
forall x, y, z : Elems . (x plus y) plus z = x plus (y plus z)
forall x : Elems . x plus zero = x /\ zero plus x = x
forall x : Elems . exists x' : Elems . x' plus x = zero
forall x, y : Elems . x times y = y times x
forall x, y, z : Elems
. (x times y) times z = x times (y times z)
forall x : Elems . x times one = x /\ one times x = x
forall x : Elems
. not x = zero => exists w : Elems . x times w = one
forall x, y, z : Elems
. (x plus y) times z = (x times z) plus (y times z)
. not zero = one
forall n : Automor; a, b : Elems
. n eval (a times b) = (n eval a) times (n eval b)
```

Now, let us start combining the notions of action and distributive space to obtain what we call here a *distributive action*. In our notation $DistrivAct = Action \vee_{G_4} DisSpace$:

```
logic CASL

%% the concept of a distributive action of a group over
%% an algebraic struture as a blend between the concepts
%%of the action of a group on a set and the concept
%% of a distributive space

spec ACTIONGROUP=

     sort Groupelem
     sort Setelem
     op Id : Groupelem
     op __ o __: Groupelem * Groupelem -> Groupelem
     op __ act __: Groupelem * Setelem -> Setelem

%% Axioms of a group for Groupelem
forall x : Groupelem; y : Groupelem; z : Groupelem
. (x o y) o z = x o (y o z) %%Associativity with o
forall x : Groupelem . x o Id = x /\ Id o x = x
%%unit with o
forall x : Groupelem . exists x' : Groupelem
. x' o x = Id %Inverse with o

%% Axiom for the action
  forall f : Groupelem; g : Groupelem; a : Setelem
. (f o g) act a = f act (g act a) %% compatibility condition
  forall a : Setelem . Id act a = a %% identity condition

end

%% a distributive space
```

```
spec DISTRSPACE=
     sort Elem
     sort Gelem
     op __ * __: Gelem * Elem -> Elem
     op __ plus __: Elem * Elem -> Elem

     forall n : Gelem, a : Elem; b : Elem
. n * (a plus b) = (n * a) plus (n * b)

end

spec GENERIC =
     sort GGroupelem
     sort GSetelem
     op __ action __: GGroupelem * GSetelem -> GSetelem

end

view I1:
      GENERIC to ACTIONGROUP =
     GGroupelem |-> Groupelem, GSetelem |-> Setelem,
__ action __ |-> __ act __
end

view I2:
      GENERIC to DISTRSPACE =
     GGroupelem |-> Gelem, GSetelem |-> Elem,
__ action __ |-> __ * __

spec Colimit= combine I1, I2
```

The blending space is shown in the next implementation under the name DISTRIBUTIVEACTION. In fact, by combining this concept with the one of a field we get as blend the concept of bijections of a field which are group homomorphisms with the sum, i.e., $BiyGHomField = DistrivAct \vee_{G_5} Field$:

```
logic CASL

%% the concept of space of bijections of a field which are
%%homomorphisms with the addition operation as a blend
%% of the concepts of a distributive action on a set and the
%% one of a field

spec DISTRIBUTIVEACTION=
    sorts GGroupelem, GSetelem
    op Id : GGroupelem
    op __o__ : GGroupelem * GGroupelem -> GGroupelem
    op __action__ : GGroupelem * GSetelem -> GSetelem
    op __plus__ : GSetelem * GSetelem -> GSetelem

forall x, y, z : GGroupelem . (x o y) o z = x o (y o z)
%%(Ax1)%
forall x : GGroupelem . x o Id = x /\ Id o x = x
```

```
%%(Ax2)%
forall x : GGroupelem . exists x' : GGroupelem
. x' o x = Id   %%(Ax3)%
forall f, g : GGroupelem; a : GSetelem
. (f o g) action a = f action (g action a) %%(Ax4)%
forall a : GSetelem . Id action a = a %%(Ax5)%
forall n : GGroupelem; a, b : GSetelem
. n action (a plus b) = (n action a) plus (n action b)
%%(Ax1_6)%
end

spec FIELD=
    %%Axioms of a field for ElemfieldE
sort ElemfieldE
ops zero, one : ElemfieldE
ops __ add __ : ElemfieldE * ElemfieldE -> ElemfieldE
ops __ mul __ : ElemfieldE * ElemfieldE -> ElemfieldE
forall x : ElemfieldE; y : ElemfieldE . x add y = y add x
forall x : ElemfieldE; y : ElemfieldE; z : ElemfieldE
. (x add y) add z = x add (y add z)
forall x : ElemfieldE . x add zero = x /\ zero add x = x
forall x : ElemfieldE . exists x' : ElemfieldE
 . x' add x = zero
forall x : ElemfieldE; y : ElemfieldE . x mul y = y mul x
forall x : ElemfieldE; y : ElemfieldE; z : ElemfieldE
. (x mul y) mul z = x mul (y mul z)
forall x : ElemfieldE . x mul one = x /\ one mul x = x
forall x : ElemfieldE
. not (x = zero) => exists w : ElemfieldE
. x mul w = one
forall x, y, z : ElemfieldE
. (x add y) mul z = (x mul z) add (y mul z)
end

spec GENERIC=
      sort Elem
      op __ + __: Elem * Elem -> Elem
end

view I1:
      GENERIC to DISTRIBUTIVEACTION =
      Elem |-> GSetelem, __ + __ |-> __ plus __
end

view I2:
      GENERIC to FIELD =
      Elem |-> ElemfieldE, __ + __ |-> __ add __
end

spec Colimit= combine I1, I2
```

Summarizing, the colimit of the last implementation corresponds to the concept of

bijections of a field which are at the same time group homomorphisms regarding to the addition of the corresponding field (for an explicit description of this blending see the space BIJECTIONADDFIELD of the next implementation).

So, we use again the concept of distributive space combining it with the concept obtained before in order to obtain the notion of group of automorphisms of a field, i.e., $AutField = BiyGHomField \vee_{G_6} DistSpace$:

```
logic CASL

%% the group of automorphisms of a field as a blend of
%%the concepts of the bijections of a field
%% which are group homomorphisms with the addition and
%%the concept of distributive space

spec BIJECTIONADDFIELD=

sorts Elem, GGroupelem
op Id : GGroupelem
op __o__ : GGroupelem * GGroupelem -> GGroupelem
op __action__ : GGroupelem * Elem -> Elem
op __mul__ : Elem * Elem -> Elem
op __plus__ : Elem * Elem -> Elem
op one : Elem
op zero : Elem

forall x, y, z : GGroupelem . (x o y) o z = x o (y o z)
%%(Ax1)%
forall x : GGroupelem . x o Id = x /\ Id o x = x
%%(Ax2)%
forall x : GGroupelem . exists x' : GGroupelem . x' o x = Id
%%(Ax3)%
forall f, g : GGroupelem; a : Elem
. (f o g) action a = f action (g action a)  %%(Ax4)%
forall a : Elem . Id action a = a %(Ax5)%
forall n : GGroupelem; a, b : Elem
. n action (a plus b) = (n action a) plus (n action b)
%%(Ax6)%
forall x, y : Elem . x plus y = y plus x %(Ax1_7)%
forall x, y, z : Elem . (x plus y) plus z = x plus (y plus z)
%%(Ax2_8)%
forall x : Elem . x plus zero = x /\ zero plus x = x
%%(Ax3_9)%
forall x : Elem . exists x' : Elem . x' plus x = zero
%%(Ax4_10)%
forall x, y : Elem . x mul y = y mul x %%(Ax5_11)%
forall x, y, z : Elem . (x mul y) mul z = x mul (y mul z)
%%(Ax6_12)%
forall x : Elem . x mul one = x /\ one mul x = x %%(Ax7)%
forall x : Elem
. not x = zero => exists w : Elem . x mul w = one
%%(Ax8)%
forall x, y, z : Elem
```

```
. (x plus y) mul z = (x mul z) plus (y mul z)
%%(Ax9)%
. not (zero = one) %% non trivial fields
end

spec DISTRSPACE=
     sort Elemnew
     sort Gelem
     op __ * __: Gelem * Elemnew -> Elemnew
     op __ oper __: Elemnew * Elemnew -> Elemnew

     forall n : Gelem, a : Elemnew; b : Elemnew
. n * (a oper b) = (n * a) oper (n * b)

end

spec GENERIC=
     sort Elems, Grelems
     op __ times __ : Elems * Elems -> Elems
     op __ act __ : Grelems * Elems -> Elems
end

view I1:
     GENERIC to BIJECTIONADDFIELD =
     Elems |-> Elem, Grelems |-> GGroupelem,
__ times __ |-> __ mul __, __ act __ |-> __ action __
end

view I2:
     GENERIC to DISTRSPACE =
     Elems |-> Elemnew, Grelems |-> Gelem,
__ times __ |-> __ oper __, __ act __ |-> __ * __
end

spec Colimit = combine I1, I2
```

Now, the colimit obtained here is exactly equivalent to the specification (in CASL) of Aut$L$ given before. In fact, for the exact specification see the concept AUTOMORPHSPACE of the next implementation.

3.4 Aut(E/F)

In Galois theory one of the most fundamental notions to consider is the one of the subgroup of the group of automorphisms of a field E, where E/F is an extension of fields, fixing any element of F. This notion corresponds to the concept of Galois group of the extension E/F, when E/F is a Galois extension [14].

Definition 9. Let E/F be an extension of fields. Then $\mathrm{Aut}_F E$ (or $\mathrm{Aut}(E/F)$) is the group of automorphisms of the extension E/F, i.e., the collection of $\phi \in \mathrm{Aut}E$ such that $\phi(x) = x$, for all $x \in F$.

We generate the concept $\mathrm{Aut}_F E$ just by blending two more concepts to the notion of Aut$E$ as follows. In the next implementation we generate the concept of group of homomorphisms of a field, which is the extension of another field as a blend of the concepts of group of automorphisms of a field and the concept of algebraic bisubstructure, with our notation

$$AutFieldExt = AutField \vee_{G7} AlgBiSubStruc.$$

```
logic CASL

spec AUTOMORPHSPACE =
sorts Elems, Grelems
op Id : Grelems
op __action__ : Grelems * Elems -> Elems
op __o__ : Grelems * Grelems -> Grelems
op __plus__ : Elems * Elems -> Elems
op __times__ : Elems * Elems -> Elems
op one : Elems
op zero : Elems

forall x, y, z : Grelems . (x o y) o z = x o (y o z)
%(Ax1)%
forall x : Grelems . x o Id = x /\ Id o x = x %(Ax2)%
forall x : Grelems . exists x' : Grelems . x' o x = Id
%(Ax3)%
forall f, g : Grelems; a : Elems
. (f o g) action a = f action (g action a) %(Ax4)%
forall a : Elems . Id action a = a %(Ax5)%
forall n : Grelems; a, b : Elems
. n action (a plus b) = (n action a) plus (n action b)
%(Ax6)%
forall x, y : Elems . x plus y = y plus x %(Ax1_7)%
forall x, y, z : Elems
. (x plus y) plus z = x plus (y plus z) %(Ax8)%
forall x : Elems . x plus zero = x /\ zero plus x = x
%(Ax9)%
forall x : Elems . exists x' : Elems . x' plus x = zero
%(Ax10)%
forall x, y : Elems . x times y = y times x %(Ax11)%
forall x, y, z : Elems
. (x times y) times z = x times (y times z)%(Ax12)%
forall x : Elems . x times one = x /\ one times x = x
%(Ax13)%
forall x : Elems
. not x = zero => exists w : Elems . x times w = one
%(Ax14)%
forall x, y, z : Elems
. (x plus y) times z = (x times z) plus (y times z)
%(Ax15)%
. not zero = one %(Ax16)%
forall n : Grelems; a, b : Elems
. n action (a times b) = (n action a) times (n action b)
```

```
%(Ax1_17)%
end

%% Axioms of a algebraic bisubstructure

spec ALGBISUBSTRUC =
sorts GAlgStruc, GSubAlgStruc
op 0 : GAlgStruc
op 1 : GAlgStruc
op __*__ : GAlgStruc * GAlgStruc -> GAlgStruc
op __**__ : GSubAlgStruc * GSubAlgStruc -> GSubAlgStruc
op __+__ : GAlgStruc * GAlgStruc -> GAlgStruc
op __++__ : GSubAlgStruc * GSubAlgStruc -> GSubAlgStruc
op inj2 : GSubAlgStruc -> GAlgStruc
op one : GSubAlgStruc
op zero : GSubAlgStruc

forall x, y : GSubAlgStruc . inj2(x) = inj2(y) => x = y
%%(Ax1)%
. inj2(zero) = 0 %%(Ax2)%
forall x, y : GSubAlgStruc . inj2(x ++ y) = inj2(x) + inj2(y)
%%(Ax3)%
forall a : GAlgStruc; b : GSubAlgStruc
. a + inj2(b) = 0 => exists c : GSubAlgStruc . a = inj2(c)
%%(Ax4)%
. inj2(one) = 1 %(Ax2_6)%
forall x, y : GSubAlgStruc . inj2(x ** y) = inj2(x) * inj2(y)
%%(Ax3_7)%
forall a : GAlgStruc; b : GSubAlgStruc
. a * inj2(b) = 1 => exists c : GSubAlgStruc . a = inj2(c)
%%(Ax4_8)%
end

spec GENERIC =
     sort Fieldelem
     ops 0,1 : Fieldelem
     op __ add __ : Fieldelem * Fieldelem -> Fieldelem
     op __ mul __ : Fieldelem * Fieldelem -> Fieldelem
end

view I1: GENERIC to AUTOMORPHSPACE =
         Fieldelem |-> Elems, 0 |-> zero, 1 |-> one,
__ add __ |-> __ plus __, __ mul __ |-> __ times __
end

view I2: GENERIC to ALGBISUBSTRUC =
         Fieldelem |-> GAlgStruc, 0 |-> 0, 1 |-> 1,
__ add __ |-> __ + __, __ mul __ |-> __ * __
end

spec Colimit = combine I1, I2
```

Finally, we blend the last colimit with the concept of a space of fixed points in order to obtain the notion of $\mathrm{Aut}_F E$. So,

$$\mathrm{Aut}(F/E) = AutFieldExt \vee_{G_8} FixPointSpc.$$

```
logic CASL
%% The concept of Aut(E/F) as a blend of the concepts
%% of the group of automorphisms of a field extension
%%and the concept of an space of fixed points of a
%%collection of endomorphisms Space of Automorphisms
%%of a field extension
spec AUTOFIELDEXT =
sorts Fieldelem, GSubAlgStruc, Grelems
op 0 : Fieldelem
op 1 : Fieldelem
op Id : Grelems
op __**__ : GSubAlgStruc * GSubAlgStruc -> GSubAlgStruc
op __++__ : GSubAlgStruc * GSubAlgStruc -> GSubAlgStruc
op __action__ : Grelems * Fieldelem -> Fieldelem
op __o__ : Grelems * Grelems -> Grelems
op __plus__ : Fieldelem * Fieldelem -> Fieldelem
op __times__ : Fieldelem * Fieldelem -> Fieldelem
op inj2 : GSubAlgStruc -> Fieldelem
op one : GSubAlgStruc
op zero : GSubAlgStruc

forall x, y, z : Grelems . (x o y) o z = x o (y o z)
%(Ax1)%
forall x : Grelems . x o Id = x /\ Id o x = x
%(Ax2)%
forall x : Grelems . exists x' : Grelems . x' o x = Id
%(Ax3)%
forall f, g : Grelems; a : Fieldelem
. (f o g) action a = f action (g action a)%(Ax4)%
forall a : Fieldelem . Id action a = a %(Ax5)%
forall n : Grelems; a, b : Fieldelem
. n action (a plus b) = (n action a) plus (n action b)
%(Ax6)%
forall x, y : Fieldelem . x plus y = y plus x %(Ax1_7)%
forall x, y, z : Fieldelem
. (x plus y) plus z = x plus (y plus z)%(Ax8)%
forall x : Fieldelem . x plus 0 = x /\ 0 plus x = x
%(Ax9)%
forall x : Fieldelem . exists x' : Fieldelem
. x' plus x = 0 %(Ax10)%
forall x, y : Fieldelem . x times y = y times x
%(Ax11)%
forall x, y, z : Fieldelem
. (x times y) times z = x times (y times z) %(Ax12)%
forall x : Fieldelem . x times 1 = x /\ 1 times x = x
%(Ax13)%
```

```
forall x : Fieldelem
. not x = 0 => exists w : Fieldelem . x times w = 1
%(Ax14)%
forall x, y, z : Fieldelem
. (x plus y) times z = (x times z) plus (y times z)
%(Ax15)%
. not 0 = 1 %(Ax16)%
forall n : Grelems; a, b : Fieldelem
. n action (a times b) = (n action a) times (n action b)
%(Ax1_17)%
forall x, y : GSubAlgStruc
 . inj2(x) = inj2(y) => x = y %(Ax1_18)%
. inj2(zero) = 0 %(Ax2_19)%
forall x, y : GSubAlgStruc
. inj2(x ++ y) = inj2(x) plus inj2(y)%(Ax3_20)%
forall a : Fieldelem; b : GSubAlgStruc
. a plus inj2(b) = 0 => exists c : GSubAlgStruc
 . a = inj2(c)%(Ax4_21)%
. inj2(one) = 1 %(Ax2_6)%
forall x, y : GSubAlgStruc
. inj2(x ** y) = inj2(x) times inj2(y)%(Ax6_22)%
forall a : Fieldelem; b : GSubAlgStruc
. a times inj2(b) = 1 => exists c : GSubAlgStruc
 . a = inj2(c)%(Ax7)%
end

spec FIXEDPOINTSPACE =
     sort Fixedelems, Elems, Functions
     op __ eval __ : Functions * Elems -> Elems
     op immer: Fixedelems -> Elems
%% condition guaranteeing that immer(Fixedelems) is
%% the fixed point set of Functions
forall x : Elems . (forall f : Functions
 . f eval x = x <=> exists y : Fixedelems . immer(y) = x)
end
spec GENERIC =
     sorts SubStruc, Struc, Automorphs
     op __ evalu __ : Automorphs * Struc -> Struc
     op inclu : SubStruc -> Struc
end

view I1: GENERIC to AUTOFIELDEXT =
        SubStruc |-> GSubAlgStruc, Struc |-> Fieldelem,
__ evalu __ |-> __ action __, inclu |-> inj2
end

view I2: GENERIC to FIXEDPOINTSPACE =
SubStruc |-> Fixedelems, Struc |-> Elems,
Automorphs |-> Functions, __ evalu __ |-> __ eval __,
inclu |-> immer
end
```

```
spec Colimit = combine I1, I2
```

So, for the sake of completeness we present the explicit specification of $\mathrm{Aut}_F E$ obtained computing the former colimit:

```
sorts Automorphs, Struc, SubStruc
op 0 : Struc
op 1 : Struc
op Id : Automorphs
op __**__ : SubStruc * SubStruc -> SubStruc
op __++__ : SubStruc * SubStruc -> SubStruc
op __evalu__ : Automorphs * Struc -> Struc
op __o__ : Automorphs * Automorphs -> Automorphs
op __plus__ : Struc * Struc -> Struc
op __times__ : Struc * Struc -> Struc
op inj2 : SubStruc -> Struc
op one : SubStruc
op zero : SubStruc

forall x, y, z : Automorphs . (x o y) o z = x o (y o z)
%(Ax1)%
forall x : Automorphs . x o Id = x /\ Id o x = x %(Ax2)%
forall x : Automorphs . exists x' : Automorphs . x' o x = Id
%(Ax3)%
forall f, g : Automorphs; a : Struc
. (f o g) evalu a = f evalu (g evalu a)%(Ax4)%
forall a : Struc . Id evalu a = a %(Ax5)%
forall n : Automorphs; a, b : Struc
. n evalu (a plus b) = (n evalu a) plus (n evalu b)
%(Ax6)%
forall x, y : Struc . x plus y = y plus x %(Ax1_7)%
forall x, y, z : Struc
. (x plus y) plus z = x plus (y plus z) %(Ax8)%
forall x : Struc . x plus 0 = x /\ 0 plus x = x %(Ax9)%
forall x : Struc . exists x' : Struc . x' plus x = 0
%(Ax10)%
forall x, y : Struc . x times y = y times x %(Ax11)%
forall x, y, z : Struc
. (x times y) times z = x times (y times z) %(Ax12)%
forall x : Struc . x times 1 = x /\ 1 times x = x %(Ax13)%
forall x : Struc
. not x = 0 => exists w : Struc . x times w = 1 %(Ax14)%
forall x, y, z : Struc
. (x plus y) times z = (x times z) plus (y times z)%(Ax15)%
. not 0 = 1 %(Ax16)%
forall n : Automorphs; a, b : Struc
. n evalu (a times b) = (n evalu a) times (n evalu b)
%(Ax1_17)%
forall x, y : SubStruc . inj2(x) = inj2(y) => x = y %(Ax1_18)%
. inj2(zero) = 0 %(Ax2_19)%
forall x, y : SubStruc . inj2(x ++ y) = inj2(x) plus inj2(y)
%(Ax3_20)%
forall a : Struc; b : SubStruc
```

```
. a plus inj2(b) = 0 => exists c : SubStruc . a = inj2(c)
%(Ax4_21)%
. inj2(one) = 1 %(Ax2_6)%
forall x, y : SubStruc . inj2(x ** y) = inj2(x) times inj2(y)
%(Ax6_22)%
forall a : Struc; b : SubStruc
. a times inj2(b) = 1 => exists c : SubStruc . a = inj2(c)
%(Ax7)%
forall x : Struc; f : Automorphs
. f evalu x = x <=> exists y : SubStruc . inj2(y) = x
%(Ax1_25)%
```

The diagrammatic representation in the last graphic (Figure 1) displays the blending interconnections between the several concepts. The way to read the graphic is the following: Going upwards, the blending concept is where two lines (coming from two concepts) meet. For example, the concepts *Punc. Abelian Group* and *Abelian Group* meet in their blend *bigroup*. Besides, two versions of the concept *Alg. Substructure* blend into the concept *Alg. Bisubstructure*.

Conclusions

When we understand computational mathematics also in the sense of developing computationally feasible tools for *creating-discovering* abstract mathematical concepts and theories, then the implementations developed here start to offer initial basic features playing a central role in the formation of mathematical concepts, e.g. basic formal analogy making. More concretely, in all the generated mathematical definitions, we got the blended concepts as combinations of two input conceptual spaces by doing basic syntactic identifications between some sort, function and/or predicate symbols. So, this identifications can be seen as very elementary instances of formal analogies (see for example [22]).

Besides, the collection of former examples supports the meta-mathematical soundness of the formalization of blending through a colimit categorical approach, since it suggests a very concrete way for modeling our mind's formal ability to create new mathematical definitions.

Furthermore, from a computational creativity perspective, these explicit implementations also suggest that formal conceptual blending (as colimits) routines could be subsequently implemented in automatic theorem provers and automated (mathematical) reasoning systems in order to make such a systems more efficient and to improve the quality of their models of creative reasoning.

Finally, going forward towards a more practical usage of automated theorem provers by working mathematicians, as suggested by A. Bundy in [4], one can also integrate 'blending' routines within an automated theorem synthesis [4, §6], i.e., processes of generating from a particular set of axioms 'interesting' collections of theorems (mathematically speaking) by means of some fixed inference rules. Specifically, this could be implemented into the MATHsAID system [15].

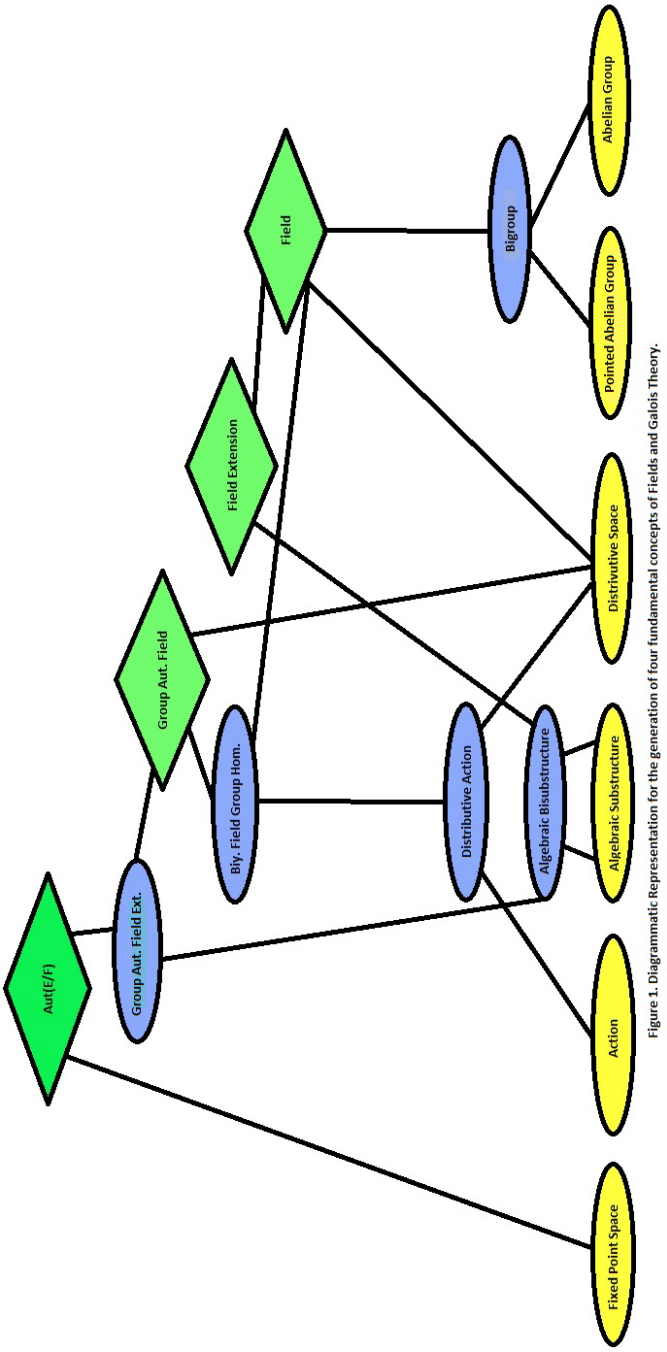


Figure 1: Diagrammatic Representation for the generation of four fundamental concepts of Fields and Galois Theory.

Acknowledgements

The author wishes to thank all other members of the project COINVENT supported by the European Commission (SP7-ICT-2013-10), under FET-Open Grant number: 611553. Specially, he thanks Alan Smaill for valuable and enlightening discussions.

References

[1] Alexander, J.: Blending in mathematics. Semiotica **187**, 1–48 (2011)

[2] Bidoit, M., Mosses, P.D.: CASL User Manual. Lecture Note in Computer Science 2900, Springer Verlag Berlin Heidelberg New York (2004)

[3] Bou, F., Corneli, J., Gomez-Ramirez, D., Maclean, E., Peace, A., Schorlemmer, M., Smaill, A.: The role of blending in mathematical invention. Proceedings of the Sixth International Conference on Computational Creativity (ICCC). S. Colton et. al., eds. Park City, Utah, June 29-July 2, 2015. Publisher: Brigham Young University, Provo, Utah. pp. 55–62 (2015)

[4] Bundy, A.: Automated theorem provers: a practical tool for the working mathematician? Ann. Math. Artif. Intell. **61**, 3–14 (2011)

[5] Cambouropoulos, E., Kaliakatsos-Papakostas, M., Kuehnberger, K.U., Kutz, O., Smaill, A.: Concept invention and music: Creating novel harmonies via conceptual blending. In: Proceedings of the 9th Conference on Interdisciplinary Musicology (CIM) (2014)

[6] Cambouropoulos, E., Kuehnberger, K.U., Kutz, O., Smaill, A., Schorlemmer, M., Colton, S., Pease, A.: Coinvent: Towards a computational concept invention theory. In: Proceedings of the 5th International Conference on Computational Creativity (ICCC) (2014)

[7] Fauconnier, G., Turner, M.: The Way We Think. Basic Books (2003)

[8] Gomez-Ramirez, D.: Conceptual blending as a creative meta-generator of mathematical concepts: Prime ideals and dedekind domains as a blend. C3GI at UNILOG 2015, Workshop on Computational Creativity, Concept Invention, and General Intelligence. Tarek B. Besold et. al., eds. Publications of the Institute of Cognitive Sciences, PICS series, University of Osnabrück Vol. 2 (2015)

[9] Gougen, J.: An introduction to algebraic semiotic with application to user interface design. *In* Computation for methaphors, analogy and agents. C. L. Nehaniv, Ed. Vol. 1562 pp. 242–291 (1999)

[10] Gougen, J.: Towards a design theory for virtual worlds: algebraic semiotics and scientific visualization as a case study. *In* Proceedings Conference on Virtual Worlds and Simulation (Phoenix AZ, 7-11 January 2001). C. Landauer and K. Bellman, Eds. Society for Modelling and Simulation pp. 298–303 (2001)

[11] Gougen, J.: Steps towards a design theory for virtual worlds. *In* Developing future interactive systems. M. Sanchez-Segura, ed. Idea Group publishing, 116-152. pp. 116–128 (2003)

[12] Horward, P., Rubin, J.E.: Consecuences of the Axiom of Choice, vol. 59. Mathematical surveys and monographs, American Mathematical Society, USA (1998)

[13] J. Fleuriot E. Maclean, A.S., Winterstein, D.: Reinventing the complex numbers (2014)

[14] Lang, S.: Algebra (revised third edition). Graduate Texts in Mathematics 211, Springer-Verlag, New York (2002)

[15] McCasland, R.L., Bundy, A., Smith, P.F.: Ascertaining mathematical theorems. ENTCS **151**, 21–38 (2006)

[16] Meinke, K., Tucker, J.V.: Many-sorted logic and its applications. John Wiley & Sons, Inc., New York, NY, USA (1993)

[17] Mossakowski, T.: Colimits of order-sorted specifications. In: F.P. Presicce (ed.) Recent trends in algebraic development techniques. Proc. 12th International Workshop, Vol 1376, pp. 316-332, Lecture Notes in Computer Sciences, Springer-Verlag, London (1998)

[18] Mossakowski, T., Maeder, C., Codescu, M.: (2014). URL `www.informatik.uni-bremen.de/agbkb/forschung/formal\_methods/CoFI/hets/UserGuide.pdf`

[19] Munkres, J.: Topology. Second Edition. Prentice Hall, Inc (2000)

[20] Rotman, J.: An Introduction to algebraic topology. Springer (2004)

[21] Schorlemmer, M., Smaill, A., Kuehnberger, K.U., Kutz, O., Colton, S., Cambouropoulos, E., Pease, A.: COINVENT: Towards a computational concept invention theory. In: 5th International Conference on Computational Creativity (ICCC)

[22] Schwering, A., Krumnack, U., Kuehnberger, K.U., Gust, H.: Syntactic principles of heuristic driven theory projection. Cognitive Systems Research **10(3)**, 251–269 (2009)

5 Computational Invention of Cadences and Chord Progressions by Conceptual Chord-Blending (Article 4)

M. Eppe, R. Confalonieri, E. Maclean, M. Kaliakatsos-Papakostas, E. Cambouropoulos, M. Schorlemmer, M. Codescu, and K.-U. Kühnberger: Computational Invention of Cadences and Chord Progressions by Conceptual Chord-Blending. Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI) 2015.

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.

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

Manfred Eppe
IIIA-CSIC, Barcelona, Spain
ICSI, Berkeley, USA

Roberto Confalonieri
IIIA-CSIC,
Barcelona, Spain

Ewen Maclean
University of Edinburgh,
UK

Maximos Kaliakatsos
University of Thessaloniki,
Greece

Emilios Cambouropoulos
University of Thessaloniki,
Greece

Marco Schorlemmer
IIIA-CSIC,
Barcelona, Spain

Mihai Codescu
University of Magdeburg,
Germany

Kai-Uwe Kühnberger
University of Osnabrück,
Germany

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).

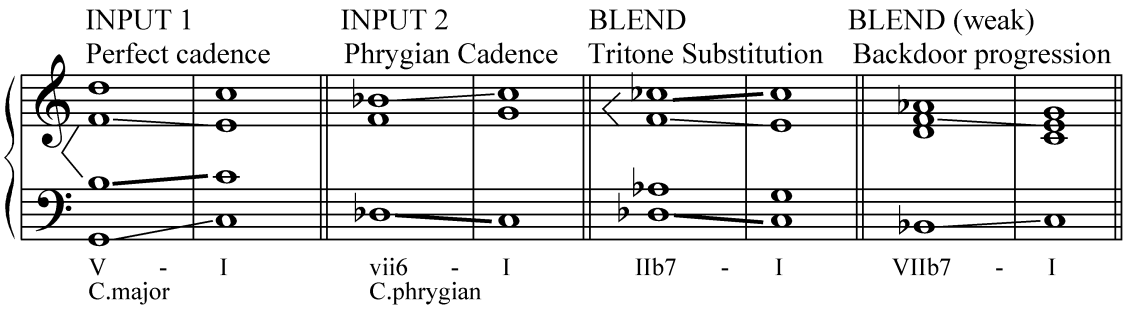


Figure 1: Conceptual blending between the perfect and Phrygian cadence gives rise to the tritone substitution progression and the backdoor progression

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.

2 An Algebraic Model of Chords

For our blending framework, we follow [Goguen, 1999]'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* $S = (\langle \mathscr{S}\mathscr{T}, \lesssim_{\mathscr{S}\mathscr{T}} \rangle, \mathscr{O}, \mathscr{P}, \langle \mathscr{A}, \leq_{\mathscr{A}} \rangle)$ *consists of a set* $\mathscr{S}\mathscr{T}$ *of sorts along with a preorder* $\lesssim_{\mathscr{S}\mathscr{T}}$ *that defines a subsort relationship, a set* $\mathscr{O}$ *of operators that map objects of argument sorts to the respective domain sort, a set* $\mathscr{P}$ *of predicates that map objects to Boolean values, and a set of axioms* $\mathscr{A}$ *with a partial priority order* $\leq_{\mathscr{A}}$*.*

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

[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.

priority ordering of axioms. As notational convention, we use superscript $\mathscr{A}^S$ 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 $root : AbsRelChord \to Note$ 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:

$$\begin{aligned} \forall n : Note; c : AbsRelChord.\ & relNote(c,n) \\ & \iff absNote(c, n + root(c)) \end{aligned} \tag{1}$$

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**

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

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:

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

$$\forall c : AbsRelChord.\ \neg relNote(c,1) \tag{3b}$$

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

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, Fauconnier and Turner, 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:

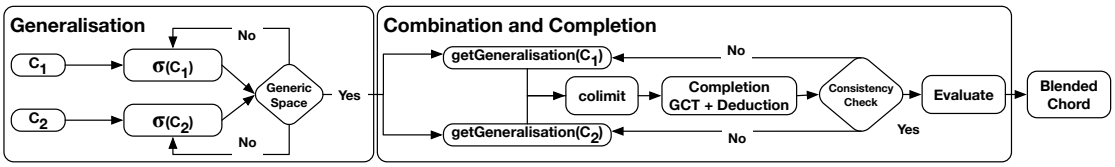


Figure 2: Blending as interleaved generalisation, combination and completion process

Definition 2 (Generic space)**.** *Given a set* **S** *of chord specifications, we say that a specification G is a generic space of* **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:

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

We realise the generalisation of chords by successive application of *generalisation operators*, which remove axioms from a chord specification. For each axiom $ax \in \mathscr{A}^S$ 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*

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

where $S \setminus ax$ *denotes the removal of an axiom ax from the poset of axioms* $\mathscr{A}^S$ *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 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* $\sigma_1, \ldots, \sigma_n$ *be generalisation operators. We denote a generalisation path as* $p = \sigma_1; \ldots; \sigma_n$ *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).

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

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 $absNote(c, 11)$ 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 $\sigma(absNote(c,2)); \sigma(absNote(c,7))$.
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:

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

First, $cost(p)$ 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, $totalCost$ 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, 2002], 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.

	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	∅	∅	G11	root G	G11 → C (total cost = 0)
Perfect/Plagal **Diatonic extension**	G7 ⟶ C F ⟶ C	∅	$\sigma(abs(0))$; $\sigma(rel(7))$	G9	root G	G9 → C (total cost = 4)
Phrygian/Plagal **Modified Phrygian**	B♭min ⟶ C F ⟶ C	∅	$\sigma(rel(4))$	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

▷ *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

	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°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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[Mossakowski, 1998] Mossakowski, T. (1998). 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.

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

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

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

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

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

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

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

[Schulz, 2013] Schulz, S. (2013). System Description: E 1.8. In McMillan, K., Middeldorp, A., and Voronkov, A., editors, *Proceedings of LPAR*, volume 8312 of *LNCS*. Springer.

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

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

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

6 Conclusions

Deliverable 1.2 summarizes the modeling of hard examples of concept invention by conceptual blending in the domains of mathematics and music. It is intended to document in a detailed and precise way the working of the blending process in a variety of examples. It is based on four papers that give a representative overview of the strengths of the underlying approach. The concepts invented by the blending process in both domains are abstract, rich, and complex. The deliverable is considered as a proof of concept of the feasibility of the proposed methodology. In both domains, the former collection of examples can be seen as an initial motivation for developing a new cognitively inspired meta-formalization of concept creation in mathematics and music.