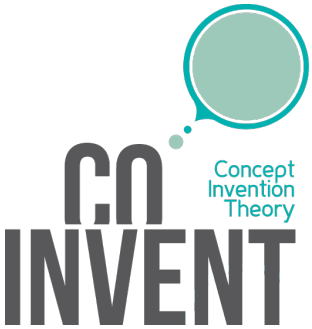


D6.1
Survey of conceptual blending in mathematics and formal reasoning

Authors	Jacques Fleuriot, Ewen Maclean, Alan Smaill and Daniel Winterstein
Reviewers	Marco Schorlemmer

Grant agreement no.	611553
Project acronym	COINVENT - Concept Invention Theory
Date	October 1, 2014
Distribution	PU/RE/CO

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 presents the mathematical theories so far investigated for the Coinvent project. We motivate applying concept blending to mathematics by investigating discussions on the subject of mathematical creativity by significant mathematicians and theoreticians. The approach presented here is to use the HDTP Gust, Kühnberger and U. Schmidt, 2006; M. Schmidt, 2010 software to discover automatically commonalities between two or more input theories, and to identify morphisms between their symbols. We also employ the HETS Mossakowski, Maeder and Lüttich, 2007 software which allows us toexploit such morphisms to compute a *colimit*, which describes the *blend* of the input theories. We give examples of mathematical theories which we have investigated and formalised, and describe in detail examples which we have mechanised using the HDTP and HETS software.
Keyword list: **concept invention**, **creativity**

Executive Summary

- The Coinvent project aims to introduce the concept of blending to various domains — this document introduces some of the mathematical domains investigated.
- Investigated domains exploit mechanised calulation of a Generic space and computation of a blended theory.
- There is much historical evidence of the use of analogy and blending in creativity in mathematics.
- Implemented examples include the complex numbers, investigation of data structures and construction of the integers from the natural numbers.
- Formalised examples include development of the Quaternions, Octonians, projectile motion and symmetry examples.
- Making theories more concrete is done via refinement. Discovery of definitions of missing functions is explained.

Contents

1 Introduction

This document presents various investigations that have been carried out in the Mathematics Domain for the COINVENT project. In all cases we try to show how the domains in question can be analysed using the notion of *blending*. We exploit the notion of theory morphisms to identify commonalities between two or more theories, and use the notion of a categorical colimit to compute a blended theory.

1.1 Two Different Theories

For the purposes of explaining concept blending in mathematics, let us first consider a very simple example from a mathematical domain. Let us assume we know of two different theories $T1$ and $T2$, whose signature and definitions are shown in Figure 1. Contained within the theories are the fundamental sorts and typed definitions of constants and axioms or theorems of the theory.

$$\begin{array}{rc} \mathit{sort}: & \sigma \\ \mathit{ops} & e_1 : \sigma \\ & \cdot_1 : \sigma \times \sigma \to \sigma \\ \forall a,b,c:\sigma. & (a \cdot_1 b) \cdot_1 c = a \cdot_1 (b \cdot_1 c) \\ & a \cdot_1 e_1 = a \\ & a \cdot_1 b = b \cdot_1 a \end{array} \qquad \begin{array}{rc} \mathit{sort} & \upsilon \\ \mathit{ops} & e_2 : \upsilon \\ & \cdot_2 : \upsilon \times \upsilon \to \upsilon \\ \forall a,b,c:\upsilon. & (a \cdot_2 b) \cdot_2 c = a \cdot_2 (b \cdot_2 c) \\ & a \cdot_2 e_2 = a \\ & (a \cdot_2 b) \cdot_2 a = b \end{array}$$

Figure 1: Two theories $T1$ (left) and $T2$ (right) with some commonality

1.2 Calculating a Generic Theory

There are commonalities in the signatures of theories $T1$ and $T2$. It is possible to identify these commonalities and define a *signature morphism* which identifies symbol mappings required to express a theory which contains just the common elements. We describe this common theory as the *generic* theory. In this case is can be defined as

$$\begin{array}{rc} \mathit{sort} & \tau \\ \mathit{ops} & e_g : \tau \\ & \cdot_g : \tau \times \tau \to \tau \\ \forall a,b,c:\tau. & (a \cdot_g b) \cdot_g c = a \cdot_g (b \cdot_g c) \\ & a \cdot_g e_g = a \end{array}$$

where a signature morphism can be defined between the generic theory and $T1$ and $T2$ which identifies the symbol mappings:

$$\begin{array}{ccccc} \sigma & \leftarrow & \tau & \rightarrow & \upsilon \\ e_1 & \leftarrow & e & \rightarrow & e_2 \\ \cdot_1 & \leftarrow & \cdot_g & \rightarrow & \cdot_2 \end{array}$$

1.3 Computing a Colimit

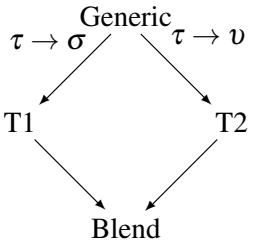
Now that we have defined the generic space it is possible to compute the *colimit* of the theory morphism. Computing the *colimit* is one way of discovering a *blend* and in the context of mathematics we use the two terms interchangeably. The elements of theories $T1$ and $T2$ not mapped to part of the Generic theory are combined into the theory represented by the colimit:

$$\begin{array}{rc} \textit{sort} & \chi \\ \textit{ops} & e_c : \chi \\ & \cdot_c : \chi \times \chi \to \chi \\ \forall a,b,c : \chi. & (a \cdot_c b) \cdot_c c = a \cdot_c (b \cdot_c c) \\ & a \cdot_c e_c = a \\ & a \cdot_c b = b \cdot_c a \\ & (a \cdot_c b) \cdot_c a = b \end{array}$$

Reasoning with this theory shows that every element must be its own inverse: $a \cdot_c a = e_c$, meaning this is a definition of a boolean group.

1.4 General Creative Process

In this document we present various examples of blending in mathematics. Where possible we describe the behaviour in terms of Generic Theories and Blends, or Colimits. The purpose of the project is to combine existing systems such as HDTP (Gust, Kühnberger and U. Schmidt, 2006; M. Schmidt, 2010) and HETS (Mossakowski, Maeder and Lüttich, 2007) to generate automatically Blends. In general we show blends using commutative diagrams. An example of which can be seen below which descibes graphically the process behind discovering the definition of a boolean group.



1.5 Structure of this Document

This document aims to explore the mathematical theories to which concept blending has been applied. In Section 2 we descibe some historical examples of blending in mathematics, and give motivation from existing literature. In Section 3 we describe the examples that have been mechanised in system. In Section 4 we describe some other examples that have so far only been formalised on paper. In Section 5 we describe the ways in which theories that have been generated by the blending process can be made more concrete through refinement, and in Section 6 we draw some conclusions and discuss further work.

2 Examples from the literature

In this section, we describe some relevant work in the literature relating to conceptual blending in mathematics, and more generally to analogical and metaphorical reasoning in mathematics. By and large, this work is suggestive and schematic, but not worked out in the level of detail needed for the sort of support we intend to provide for human and machine reasoners. We indicate some exceptions to this where appropriate.

2.1 Mathematical Analogies

The mathematician's view There is some provocative writing from mathematicians on the role of analogy in the process of developing new mathematics. One such example is in a short article by André Weil, writing for a popularising journal, but describing and generalising his experience of developing a new mathematical field (Weil, 1960). For example:

> 18th century mathematicians often talked about the "metaphysics of the infinitesimal calculus" or the "metaphysics of the theory of equations". They took this to refer to a collection of vague analogies, hard to grasp, which however seemed to them to play a key role at a particular period in research and invention in mathematics. ...
>
> Nothing is more fruitful, every mathematician knows it, than these obscure analogies, these disputes reflected from one theory to another, these secret caresses, these quarrels for no reason; indeed nothing gives the researcher more pleasure. The day comes when the illusion falls away; intuition turns into certainty, twin theories reveal their common source, before vanishing; ...
>
> So for our part we understand what Lagrange was looking for, when he talked about metaphysics in connection with his work on algebra: Galois theory, which he nearly puts a finger on, through a screen that he can't quite find a way through. Where Lagrange saw analogies, we now see theorems. ... As long as Lagrange simply had a premonition of these ideas, as long as he struggled to work out their essential unity through their many changing concrete manifestations, he remained within metaphysics. Still, he found the thread which let him move from problem to problem, to bring the right tools to bear, to organise the area in preparation for the future general theory. Thanks to the crucial concept of group, all this turned into mathematics with Galois.
>
> Weil, 1960

Comments stressing the role of analogy in mathematical invention are also found in the writings of Poincaré (Poincaré, 1914) and in Hamilton (see §8 on quaternions).

Cognitive Science and Philosophy approaches The notion of blending is closely associated with that of analogy and metaphor, and of metaphorical and analogical inference. The principal difference is that in analogical reasoning two different problem domains are correlated to allow reasoning in one to reflect to some extent reasoning in the other; in blending, a new domain of reasoning is opened up, combining aspects of both initial domains, yet distinct from both. Still, the correlation between domains in the analogical case is expressible via the notion of *generic*

space, with reasoning under analogy corresponding to use of notions like signature morphisms as we use in our work.

Recent work from the Philosophy of Science by Dirk Schlimm which bears attention suggests that analogy in the mathematical case has some distinctive properties; see Schlimm (2008). A more general plea for the importance of analogical reasoning is contained in Arbib and Hesse (1986).

There is a sizeable literature on computational models of analogical reasoning, including work on mathematical and scientific domains. For one mainstream approach to analogical reasoning in general from a Cognitive Science point of view, there is the extensive work of authors such as Gentner, Holyoak, Forbus and Hofstadter represented in a series of books on the topic (Gentner, K. J. Holyoak and B. N. Kokinov, 2001; B. Kokinov, K. Holyoak and Gentner, 2009).

On mathematics in particular, the book by Lakoff and Núñez (Lakoff and Núñez, 2000) provides many examples of mathematical metaphors, with some evidence of their cognitive plausibility, ranging from basic arithmetic to university-level topics.

Recent work on metaphor closely related to the topic of the current project is in Pease, Guhe and Smaill (2010), Pease, Guhe and Smaill (2009) and Pease, Colton et al. (2010).

2.2 Scientific Discovery

The approach of looking to provide computational accounts that provide rational reconstructions of major historical shifts in (empirical) science goes back some way (Langley et al., 1987; Thagard, 1993), and typically involve computational means of hypothesis or theory evaluation. The earlier work has been criticised for excessive hand tuning of systems, for a lack of any novel examples. Clearly we face similar methodological hurdles in our work.

It has been less common to look at episodes in the developments of mathematics in this way, but the examples so far suggest that there are opportunities here. The claims by Lenat for discovery of the notion of prime number by his AM system (Lenat, 1982) have proved controversial. However the HR system receives credit for coming up with the notion of "refactorable number" (Colton, 1999), showing that machines are already capable of isolating mathematically interesting concepts, not pre-targeted by the programmer.

The seminal work in Lakatos (1976) on successive formulations of Euler's conjecture is an important source of ideas and understanding of the evolution of mathematical ideas as theories are being developed. Lakatos's own exposition is not given in explicitly computational terms, but subsequent work has provided implementations of the methods proposed by Lakatos (see for example Pease (2007), Pease, Guhe and Smaill (2009) and Pease, Guhe and Smaill (2010)). There are opportunities here to recast some of this work in terms of mathematical blends.

2.3 Mathematical blending

On explicit notions of blending, the initial work in the area comes from semantic analyses in linguistics, for example in Turner and Fauconnier (1995). Although the ideas are presented in fairly informal terms, the examples used include mathematical ones (for an article in this tradition focusing on mathematics, see Turner (2005)).

A survey using blending ideas, written by a mathematician, with good associated historical information, is given in Alexander (2011).

Lakoff and Núñez's book "Where mathematics comes from" (Lakoff and Núñez, 2000) develops a cognitively inspired theory of the origin of mathematical concepts and theories; blending plays a key role, and there are plentiful examples to work with. For example, in Lakoff and Núñez (2000, p 447) we find a blend involving five different domains, associated with the equation $e^{i\pi} + 1 = 0$.

Some recent work involving Edinburgh has built a more formal account of what is going on in some of the blends involving the so-called "basic metaphors" of Lakoff and Núñez very much in the spirit of the COINVENT proposal (see Guhe et al. (2011)).

For the use of concept blends to generate conjectures, Pease, Guhe and Smaill (2009, §2) looks at relationships between geometrical theories and 2 and 3 dimensions, with associated conjectures as well as new concepts. §3 of the same paper reviews Lakatos's famous case study in the same spirit.

Our main interest is in the use of blending as a means to generate new (and interesting!) mathematical theories. The recent work by Dirk Schlimm, highlighting the creative aspect of the process of axiomatisation is clearly relevant here, and provides historical as well as philosophical interest (Schlimm, 2009; Schlimm, 2011; Schlimm, 2013).

3 Implemented examples

In this section we explore some of the examples which have been formalised, and for which we have used the machinery provided by systems such as Hets (Mossakowski, Maeder and Lüttich, 2007) and HDTP (Gust, Kühnberger and U. Schmidt, 2006; M. Schmidt, 2010). As mentioned in §1, the key to mechanising conceptual blending is to define appropriate signature morphisms. In order to do this we need the following components which are referred to throughout the sections which follow:

Generic Space The HDTP software automates the process of discovering potential common elements between theories and calculates signature morphisms between them, identifying a generic space which describes the commonalities between theories or concepts. The HDTP finds the maximal generic space – i.e. that which describes the greatest number of commonalities between theories or concepts.

Colimit or Blend The Hets software is a powerful reasoning tool which is able to compute the Colimit of two theories given a generic space calculated by HDTP. The categorical description of the colimit is equivalent to the notion of a conceptual blend, but is a mathematical rigorous definition. In the case of mathematics, as described here, the two notions of colimit and blend are interchangeable.

Specification language In order to define concepts or theories, we must first decide on a language with which to represent axioms and rules. In Hets the DOL framework Lange et al., 2012 allows various specification languages to be employed. For our work on mathematics, we generally use the CASL language Bidoit and Mosses, 2004 which is expressive enough for our needs.

4 Introduction

We provide a preliminary presentation of a blending approach for the construction of the integers given the natural numbers.

The approach taken attempts to use aspects of spatial cognition. Lakoff and Núñez (2000, p. 24) suggest, following Dehaene (1997), that numbers relate to space in various ways, e.g. that "the integers are conceptualized as being spread in space over a number line."

Lakoff and Núñez (2000, pp. 71–73) treat the integers briefly as part of an extension to the fourth and last grounding metaphor for arithmetic: motion along a path, extended by supposing that "the origin [is] somewhere on on a pathway extending indefinitely in both directions." This note assumes that we are still working with the discrete case, say with markers at regular interval along the path. Lakoff and Núñez do not present a blend construction.

Mark Turner deals with (positive) rationals and reals before moving to negative numbers. In Turner (2005), he says:

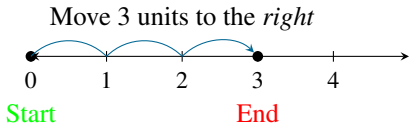
> In the next step, there is a number input, which has a half-line, a ray. But the geometric input has a full line in 2D space. Mirror image points can be obtained by

rotation by 180 degrees. So in the blended space, we have negative numbers on the line through 180-degree rotation from positive numbers. Addition is emergent as a geometric operation on segments: rotate from the extremity of segment 1 around the origin of segment 2 (by either 0 degrees or 180 degrees). Only the line is projected from the 2D space input.

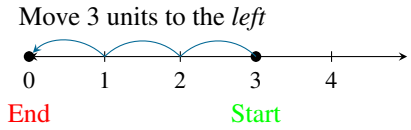
Here we do something similar for the discrete case.

5 The plan

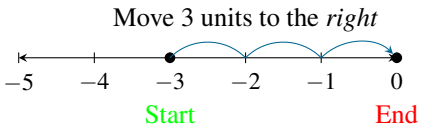
First take the naturals as involving a successor relation, an origin (0), and an associated notion of "greater" depicted by a number line where numbers increase conventionally when moving to the right. Here move some particular distance corresponding to a natural number:



Now imagine reversing direction, so moving towards the origin:



When this is rotated to correspond to negative numbers. we get the following picture.



Now put these side by side, as the two input spaces:

Notice that the reverse version of the successor relation of the left corresponds to the successor relation on the right; but these structures are not isomorphic (one has a first element, but no last; the other has a last element, but no first).

So a blend can be made by identifying the two versions of the origin, the different notions of successor, and therefore of size, and throwing out the maximimum and minimum properties that make the naive blend inconsistent.

6 Formulation in DOL

Here is a DOL axiomatisation of the situation in CASL which follows the plan above. There is no attempt to specify distinct domains; rather take all the theories here to be giving (different) notions of number, and let the blending do its work.

`NatSuc` axiomatises natural numbers, including recursion equations for addition. `NatRotate` does the same job for the rotated line, but restricts attention to properties of order and relative size. It is plausible that the generic space can be recognised by HDTP (not tested yet).

```
logic CASL

%%% blending natural numbers to get integers ...

spec NatSuc =
     sort Num
     ops  zero:Num
          __+__: Num * Num -> Num
     preds suc: Num*Num
           __<__: Num*Num
%%% < is strict linear order
       forall x,y,z,x2,y2,z2 : Num
            . not (x < x)                          %anti-sym%
            . x < y /\ y < z => x < z               %transitive%
            . x < y \/ x = y \/ y < x               %linear%
            . x < y <=> suc(x,y) \/
                        exists sx:Num .
                        (suc(x, sx) /\ sx < y) %discrete%
%%% successor
            . suc(x, y) /\ suc(x, z) => y = z  %suc_functional%
            . suc(x, z) /\ suc(y, z) => x = y  %suc_injective%
            . exists a:Num . suc(x, a)          %suc_total%
%%% zero not successor
             . not (suc(x, zero)                %zero_not_suc%)
%%% recursion eqs for <
            . suc(x, y) => zero < y             %less_base%
            . x < y /\ suc(x, x2) /\ suc(y, y2) => y < y2   %less_step%
%%% and for addition
            . zero + y = y                     %plus_base%
            . x + y = z /\ suc(x, x2) /\ suc(z, z2) =>  x2 + y = z2
end

spec NatRotate =
     sort Num
     op  zero:Num
     preds  suc2     : Num*Num              %%% rotated/reflected version
```

```
          __<-__  : Num*Num                %%% <- is linear order
    forall x,y,z:Num
          . not (x <- x)                   %anti-sym%
          . x <- y /\ y <- z => x <- z      %transitive%
          . x <- y \/ x = y \/ y <- x      %linear%
%%% predecessor
          . suc2(x, y) /\ suc2(x, z) => y = z          %suc2_functional%
          . suc2(x, z) /\ suc2(y, z) => x = y          %suc2_injective%
          . exists a:Num . suc2(a, x)                  %suc2_inverse_tota
%%% zero has no successor2
          . not (suc2(zero, x))                        %zero_no_successor
end

%%% generic space

spec Generic =
    sort Num
    op zero:Num
    preds next : Num * Num
          less : Num * Num
%%% less is strict linear order
    forall x,y,z:Num
          . not (less (x,x))                           %less_anti-sym%
          . less (x,y) /\ less (y,z) => less (x,z)     %less_transitive
          . less(x,y) \/ x = y \/ less(y,x)            %less_linear%
%%% next
          . next(x, y) /\ next(x, z) => y = z          %next_functional
          . next(x, z) /\ next(y, z) => x = y          %next_injective%
end

%%%views

view I1: Generic to NatSuc =
    zero |-> zero, less |-> __<__, next |-> suc

view I2: Generic to NatRotate =
    zero |-> zero, less |-> __<-__, next |-> suc2


spec Colimit =
    combine I1,I2
end
```

The colimit is as expected inconsistent, and recognised as such very quickly by the eprover system which is integrated in HETS. So we weaken the input theories by not mapping axioms about the limiting properties of the origin. In this case, the colimit is shown in figure 2.

```
logic CASL.FOL=

sorts Num
op __+__ : Num * Num -> Num
op zero : Num
pred less : Num * Num
pred suc2 : Num * Num

forall x : Num . not less(x, x) %(Ax1)%
forall x, y, z : Num . less(x, y) /\ less(y, z) => less(x, z)
                                                    %(Ax2)%
forall x, y : Num . less(x, y) \/ x = y \/ less(y, x) %(Ax3)%
forall x, y : Num
. less(x, y)
  <=> suc2(x, y) \/ exists sx : Num . suc2(x, sx) /\ less(sx, y)
                                                    %(Ax4)%
forall x : Num . exists a : Num . suc2(x, a) %(Ax7)%
forall x, y : Num . suc2(x, y) => less(zero, y) %(Ax8)%
forall x, y, x', y' : Num
. less(x, y) /\ suc2(x, x') /\ suc2(y, y') => less(x', y') %(Ax9)%
forall y : Num . zero + y = y %(Ax10)%
forall x, y, z, x', z' : Num
. x + y = z /\ suc2(x, x') /\ suc2(z, z') => x' + y = z'  %(Ax11)%
forall x, y, z : Num . suc2(x, y) /\ suc2(x, z) => y = z %(Ax4_15)%
forall x, y, z : Num . suc2(x, z) /\ suc2(y, z) => x = y %(Ax5_16)%
forall x : Num . exists a : Num . suc2(a, x) %(Ax6_17)%
```

Figure 2: Colimit theory

7 Comments

The colimit theory is reasonable as a first approach: I believe all the axioms hold of the integers, in particular:

- The order is that of a discrete linear order with no first or last element, as it should be.
- The properties of addition enforced are true in the integers.
- The relationship between successor and order is also as expected for the integers.

In addition, a conservative extension can supply constants for integers, since we can show that there is a unique x which succeeds 0 (call it 1), similarly for 2, etc; and there is a unique x such that 0 is its successor (call it -1), and so on. And we can show from uniqueness properties that $-1 + 1 = 0$, for example.

However, since we do not have induction here, it is presumably not provable that addition has an inverse. There is possibly a way around this, by using `HolCASL` to give second-order versions of induction axioms that correspond to first-order schemas. As long as HDTP can work with a first-order formulation as above, we can keep our original planned overall system. Within CASL, a semantic version of the strengthened axiomatisation is possible by using generated types

(for natural numbers), and related constructs in the integer constraints. This is worthy of future investigations.

There are of course different ways to use blending for current purposes. Even using this approach, while it is obvious given our purpose which axioms to drop, it is hard so far to see how to prioritise this particular weakening here. What role is geometrical or spatial intuition playing?

7.1 Complex Numbers

The complex numbers example is one which historically took many years to be established in mathematics, and is therefore a hard case to model. Nevertheless we present this work as showing that a rational reconstruction of the modern understanding of complex numbers is possible using the techniques we describe below.

7.1.1 Formal Statement of Problem

The complex numbers can be looked at as a blend between the geometrical notion of normed real vector space, and the algebraic notion of a field.

The general case As a normed real vector space V, we have axioms (in a first-order sorted presentation, with two sorts, one for vectors and one for scalars, in this case reals) as follows.

- The ordered field axioms for $\mathscr{R}$:
  1. $\mathscr{R}$ with $+_{\mathscr{R}}$, real 0 and unary -$_{\mathscr{R}}$ form a commutative group;
  2. $\mathscr{R}$ without the real 0, with $\times_{\mathscr{R}}$, **1** and $^{-1_{\mathscr{R}}}$ form a commutative group;
  3. and $\times_{\mathscr{R}}$ distributes over $+_{\mathscr{R}}$.
  4. $\leq$ is a total order on $\mathscr{R}$, which respects addition, and the product of positive numbers is positive.
- The elements of V form a commutative group with identity **0** and inverse operation (here as prefix minus).

$$\forall x,y : V \qquad x+y = y+x \tag{1}$$
$$\forall x,y,z : V \qquad (x+y)+z = x+(y+x) \tag{2}$$
$$\forall x : V \qquad x+\mathbf{0} = x \tag{3}$$
$$\forall x : V \qquad x+(-x) = \mathbf{0} \tag{4}$$

- Interaction between field operations and vector operations. Here the field is the real numbers, operations are written with subscript $\mathscr{R}$; **1** is the field multiplicative identity.

$$\forall x,y : V\ \forall \lambda : \mathscr{R} \qquad \lambda(x+y) = \lambda x + \lambda y \tag{5}$$
$$\forall x : V\ \forall \lambda,\mu : \mathscr{R} \qquad (\lambda +_{\mathscr{R}} \mu)x = \lambda x + \mu x \tag{6}$$
$$\forall x : V\ \forall \lambda,\mu : \mathscr{R} \qquad \lambda(\mu x) = (\lambda \times_{\mathscr{R}} \mu)x \tag{7}$$
$$\forall x : V \qquad \mathbf{1}x = x \tag{8}$$

- Finally there is a norm on the vector space (giving the size of vectors): $\|.\| : V \to \mathscr{R}$, with associated axioms.

$$\forall x : V \qquad \|x\| \geq 0 \tag{9}$$
$$\forall x : V \qquad \|x\| > 0 \leftrightarrow x \neq 0 \tag{10}$$
$$\forall x : V\ \forall \lambda : \mathscr{R} \qquad \|\lambda x\| = |\lambda| \times_{\mathscr{R}} \|x\| \tag{11}$$
$$\forall x, y, z : V \qquad \|x+y\| \leq \|x\| + \|y\| \tag{12}$$

On the other side, we have the field axioms, which are already part of the vector space axiomatisation. The desired generalised theory linking the axiomatisations will not use that link, but it is an obvious common feature of the two axiomatisations.

For completeness, here is an axiomatisation.

- F with $+_{\mathrm{F}}, 0_F, \text{-}_{\mathrm{F}}$ is a commutative group:

$$\forall x, y : F \qquad x +_{\mathrm{F}} y = y +_{\mathrm{F}} x \tag{13}$$
$$\forall x, y, z : F \qquad (x +_{\mathrm{F}} y) +_{\mathrm{F}} z = x +_{\mathrm{F}} (y +_{\mathrm{F}} x) \tag{14}$$
$$\forall x : F \qquad x +_{\mathrm{F}} 0_F = x \tag{15}$$
$$\forall x : F \qquad x +_{\mathrm{F}} (\text{-}_{\mathrm{F}} x) = 0_F \tag{16}$$

- F apart from 0_F with $\times_{\mathrm{F}}, 1_F, {}^{\text{-}1_{\mathrm{F}}}$ is a commutative group.
  Here use a subtype $F_{\neq 0}$ of F with an obvious definition.

$$\forall x, y : F_{\neq 0} \qquad x \times_{\mathrm{F}} y = y \times_{\mathrm{F}} x \tag{17}$$
$$\forall x, y, z : F_{\neq 0} \qquad (x \times_{\mathrm{F}} y) \times_{\mathrm{F}} z = x \times_{\mathrm{F}} (y \times_{\mathrm{F}} x) \tag{18}$$
$$\forall x : F_{\neq 0} \qquad x \times_{\mathrm{F}} 1_F = x \tag{19}$$
$$\forall x : F_{\neq 0} \qquad x \times_{\mathrm{F}} x^{\text{-}1_{\mathrm{F}}} = 1_F \tag{20}$$

- Distributivity:

$$\forall x, y, z : F \qquad x \times_{\mathrm{F}} (y +_{\mathrm{F}} z) = (x \times_{\mathrm{F}} y) +_{\mathrm{F}} (x \times_{\mathrm{F}} z) \tag{21}$$

We may now make clear what we are looking for in a blend. First, we are working in first-order sorted logic, and the morphisms should map formulae to formulae while respecting the logical form of our formulae. The generic theory should look for a morphism which maps axioms into axioms (or perhaps theorems) of *both* input theories. And the blend should include aspects of both theories, while respecting the generic aspects.

There are a couple of obvious, but non-creative, answers to the problem as stated.

1. Since the theory of the reals appears in both input spaces, we can take that to be the generic space.

   In this case we get nothing new, the blend is identical to the vector space theory.

2. Much more interestingly, the generic space can identify the addition operation in the theory of the reals with the addition operation on vectors (in both cases these are abelian groups).

   In this case, the blend says that the operations on vectors should include vector multiplication, following the example of real multiplication.

The latter case we happen to know to be consistent, because the zero-dimensional real vector space is a model (in which there is only one (zero) vector).

This is not taking us where we want to go, so far, because we have not insisted that the vector space is two-dimensional. We can do this, by adding to the vector space axioms.

Two-dimensional case We now give ourselves products of sorts and pairings directly, and then work with a refinement of normed vector spaces, i.e. one where the axioms, suitably interpreted, hold, as well as the two-dimensionality axioms, again suitably interpreted.

So, work with pairs of reals:

$$(x_1,y_1)+(x_2,y_2) =_{def} (x_1 +_{\mathscr{R}} x_2, y_1 +_{\mathscr{R}} y_2) \tag{22}$$

$$\mathbf{0} =_{def} (0,0) \tag{23}$$

$$-(x,y) =_{def} (-_{\mathscr{R}} x, -_{\mathscr{R}} y) \tag{24}$$

$$\lambda(x,y) =_{def} (\lambda \times_{\mathscr{R}} x, \lambda \times_{\mathscr{R}} y) \tag{25}$$

$$\|(x,y)\|^2 = ((x \times_{\mathscr{R}} x) +_{\mathscr{R}} (y \times_{\mathscr{R}} y))^2 \tag{26}$$

We can in fact show that, with these conditions, we do indeed have a real (normed) vector space.

Now we can look at the blend of 2-dimensional vector spaces with the real field. Again we can look to see whether the blend is consistent, i.e. is there some multiplication operation over vectors that turns the vectors into a field?

A way to show that the blend here is consistent (assuming the input spaces are) is to provide definitions of multiplication, identity and inverse in terms of what we already have, satisfying the extra field axioms. Such abbreviational definitions are guaranteed to yield a conservative extension, so not to introduce new inconsistency.

For the case of complex numbers, the standard definitions are at hand (finding them from scratch is of course a different matter, and is discussed in §9).

$$\mathbf{1} =_{def} (1,0) \tag{27}$$

$$\begin{aligned}(x_1,y_1) \times (x_2,y_2) =_{def} ((x_1 \times_{\mathscr{R}} x_2) +_{\mathscr{R}} (-_{\mathscr{R}} (y_1 \times_{\mathscr{R}} y_2)),\\ (x_1 \times_{\mathscr{R}} y_2) +_{\mathscr{R}} (y_1 \times_{\mathscr{R}} x_2))\end{aligned} \tag{28}$$

$$(x,y)^{-1} =_{def} \|(x,y)\|^{-2_{\mathscr{R}}} (x, -_{\mathscr{R}} y) \tag{29}$$

(where the inverse square operation is shorthand for the obvious thing).

The preservation of norm under multiplication While the norm operation seems to have played little role so far, we note the importance to the historical development of the role of one of the properties of the usual Euclidean norm.

The following is part of the presentation in Conway and D. A. Smith (2003):

> The geometrical properties of the complex numbers follow from the fact that they form a composition algebra for the Euclidean norm[1]
>
> $$N(x+iy) = x^2 + y^2$$
>
> which means that
>
> $$N(z_1 z_2) = N(z_1)N(z_2).$$
>
> This entails that multiplication by a fixed (non-zero) number z_0 multiplies all lengths by $\sqrt{N(z_0)}$; ...
>
> (Conway and D. A. Smith, 2003)

Their claim depends on the fact that, in a real vector space, a Euclidean norm with this compositional property uniquely determines the corresponding inner product – (see Faraut and Koranyi, 1994 for justification of the claim).

At any event, this compositionality for norms played an important role in Hamilton's discovery of Hamiltonians, where he says of the property:

> ... without which consistence being verified, I should have regarded the whole speculation as a failure.
>
> (Hamilton, 1844a)

The importance of this property suggests that the blending process should not concentrate entirely on axioms. While our earlier thoughts on blending focused exclusively on operations on sets of axioms, it is quite possible to include other statements such as key theorems in the formalism associated with the initial theory.

In the case of complex numbers, this is the statement

$$\|z_1.z_2\|^2 = \|z_1\|^2.\|z_2\|^2$$

using a less fussy notation than that above. The centrality of this feature is further emphasised by Argand's discussion of the role of absolute magnitude in his description both of real and complex numbers Argand, 1813, one of the first expositions of the modern understanding of complex numbers.

The role of key properties associated with a particular theory is reminiscent of Lakatos's characterisation of the *hard core* properties of a scientific research programme: features which are central to the identity of the theory and not to be given up, unlike other dispensable features (Lakatos, 1974). Work in belief revision has also made use of the notion of *epistemic retrenchment* to explain choice aming competing possible theory revisions.

[1] Conway & Smith here take the norm to be the square of the usual Euclidean distance.

7.1.2 Implementation

In order to automate the process of discovering blends between theories, we employ the Hets system (Mossakowski, Maeder and Lüttich, 2007). Hets supports reasoning about logical theories and relationships between them (it also supports reasoning about relationships between different logics, though we make no use of this for present purposes). It will generate proof obligations when claims are made about relationships, such as that everything provable in one theory is provable in another theory, under some translation into different syntax. It is thus well suited to help in organising and verifying claims about analogical inference.

When Hets is given signature morphisms between theories (a particular case of translations that preserve provability), it can compute a *colimit* which is the categorical equivalent of what we have thus far referred to as a *blend*. In order to compute a blend between two theories, we must first calculate the signature morphisms defining the *generic* space which is common to both theories. To do this we exploit the HDTP software (Gust, Kühnberger and U. Schmidt, 2006; M. Schmidt, 2010) which computes a generalised version of the input theories.

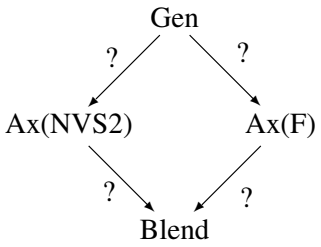
Specification using DOL In order to specify a theory, and to further define signature morphisms we use the *DOL* specification language (Lange et al., 2012). For our example of complex numbers we use the CASL logic (Bidoit and Mosses, 2004) to describe the theories. Initially we define the signature of an ordered field and a two-dimensional vector space. The specification files themselves are too big to show here but can be found in full online in the Ontohub system at `http://ontohub.org/complex-blend/complex_blend`. For the purposes of exposition we show simplified files here. Below is part of a definition of an ordered field, and part of a definition of a vector space:

```
spec Field =
  sort Real
  ops
    0:Real;
    __ + __: Real * Real -> Real;
    - __   : Real -> Real;
    __ * __: Real * Real -> Real
  forall x,y:Real
    . x+0=x    %f_plus_ident%
    . x+y=y+x  %f_com_plus%
    . x+(-x)=0 %f_plus_inv%
end

spec VectorSpace =
  sort Real
  free type Vec ::= pair(r:Real;c:Real)
  ops
    vzero:     Vec;
    0:         Real;
    vmi __:    Vec -> Vec;
    __ vpl __: Vec * Vec -> Vec
  forall x,y:Vec;
    . x vpl vzero = x;     %v_plus_ident%
    . x vpl y = y vpl x    %v_com_plus%
```

```
    . x vpl (vmi x) = zero %v_plus_inv%
end
```

This mirrors the exposition given for the definitions given for a field and vector space, introducing a pair type to indicate that we want to develop a theory of two-dimensional vectors. This gives us the following diagram. $Ax(F)$ are the axioms of the real ordered field, and $Ax(NVS2)$ are the axioms of a normed two-dimensional vectors space. The ? correspond to the signature morphisms that will be calculated by HDTP and Hets.



Calculation of Generic Space Since a vector space itself comprises an ordered field, there is a generic space which is calculated using identity morphisms which reproduces the vector space in the blend, as described in §7.1.1. We are interested here in a morphism which identifies a different morphism between the two theories. We exploit the HDTP system to generate the following signature morphism:

$$0 \quad \leftarrow_{G\to f} \quad \textit{zero} \quad \rightarrow_{G\to v} \quad \textit{vzero} \tag{30}$$

$$+ \quad \leftarrow_{G\to f} \quad \textit{plus} \quad \rightarrow_{G\to v} \quad \textit{vpl} \tag{31}$$

$$- \quad \leftarrow_{G\to f} \quad \textit{minus} \quad \rightarrow_{G\to v} \quad \textit{vmi} \tag{32}$$

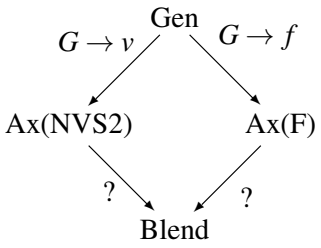
which generates the following Generic space in Hets:

```
spec Gen =
  sort Generic
  ops
    __ plus __: Generic * Generic -> Generic;
    zero:       Generic;
    minus __:   Generic -> Generic
  forall x,y,z:Generic
    . x plus  zero=x          %gen_plus_ident%
    . x plus y=y plus x       %gen_com_plus%
    . x plus (minus x) = zero %gen_plus_inv%
end
```

We now have a calculated definition of a generic space by which we can calculate a colimit. We need first to indicate in Hets using the DOL language how the signature morphism calculated by HDTP is specified:

```
view I1: Gen to Field =
  __ plus __ |->  __ + __,
  zero       |->  0,
  minus __   |->  - __

view I2: Gen to VectorSpace =
  __ plus __ |->  __ vpl __,
  zero       |->  vzero,
  minus __   |->  vmi __
```



It now remains to establish the signature morphisms to the computed colimit. The resulting theory is the blend of the ordered real field and the two dimensional vector space.

Calculation of Colimit (Blend) Hets automatically computes the colimit by determining which laws must be associated with symbols in the blend. The symbol `plus` from the Generic theory is now both associated with field addition from the ordered field and vector addition from the two dimensional vector space. The effect of this is that the two dimensional vector space "inherits" the notion of multiplication from the ordered real field. For the complex numbers this is indeed what we want, and we also inherit a definition of multiplicative inverse for vectors. We henceforth refer to the theory expressed in the colimit as $\mathscr{C}_{\alpha 0}$. See §9 for a further discussion on how a theory of the complex numbers is consistent with these notions of vector multiplication.

7.2 Data Structures

Data structures are defined using concise grammars exploiting elements of type theory, and through recursion allow for interesting complexity. We have investigated the possibility of using conceptual blending as a means for incrementally constructing complexity in data structures. In this section we develop an example which discovers a definition of binary trees with data at the nodes, by initially considering non-recursive data structures.

7.2.1 Calculation of Generic Space and Colimit

In what follows we present the process of using blending to calculate new data structures. As in the case of complex numbers, this requires the calculation of a Generic space and the computation of a colimit. We once again exploit the HDTP software to discover commonalities and define a

Generic space and a set of signature morphisms. We use the Hets system to calculate the colimit given the input spaces described and the calculated Generic space.

7.2.2 Creating Recursion using Blending

Consider two non-recursive data structures based on the notion of pairs:

```
spec Pair1  =
     sort S1,S2
     free type Pair ::= nil | c(S1,S2)
end

spec Pair2 =
     sort S1,S2
     free type Pair ::= c(S1,S2)
end
```

which are fairly uninteresting in themselves as data structures. However one can use the notions of conceptual blending to create a recursively defined data structure. If one defines a generic specification to be

```
spec Gen =
   sort G1
   sort G2
end
```

then it is possible to exploit the colimit calculation in Hets to generate a definition of a new data structure by assigning these types to different parts within each pair structure:

```
view I1: Gen to Pair1 =
     G1 |-> Pair,
     G2 |-> S2

view I2: Gen to Pair2 =
     G1 |-> Pair,
     G2 |-> Pair

spec Colimit =
     combine I1,I2
```

which generates a new specification:

```
spec Blend =
   sorts S1,S2,S1',S2'
   free type S2 = c(S1,S2)|nil|c'(S1',S2')
end
```

which is a description which contains a recursively defined datatype `S2`, similar to a typical definition of a linked list. There is a terminating element `nil` and a recursively defined constructor `c`. On account of the colimit calculation there is also another terminating element defined by a non-recursuvely defined constructor `c'`. Figure 7.2.2 shows graphically how taking a generic space of just two sorts and attributing them to similar non-recursive pair types allows recursion to be introduced via the calculation of a colimit. The key is that `G2` in the generic space `GEN` is both mapped in the signature morphism to the second element of the `PAIR2`, and `PAIR1` itself. `G1` then maps to both `PAIR1` and `PAIR2` ensuring that the colimit must contain a recursively defined pair as shown the `BLEND` type in Figure 7.2.2.

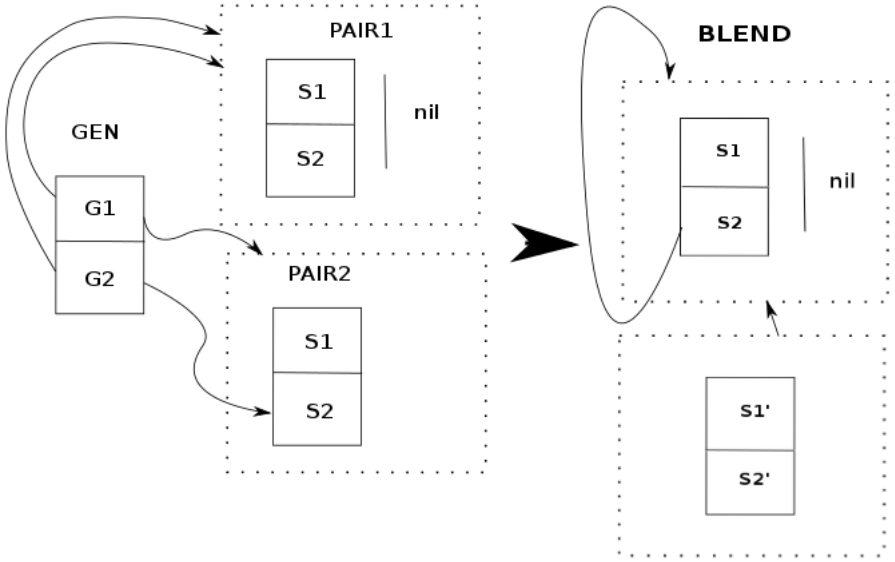


Figure 3: Using the concept of blend to introduce the notion of recursion in datastructures

7.2.3 Discovering a Binary Tree

Following on from using concept blending to discover recursion, we show an example of how blending can be used to create new structures, and in particular how it is possible to use the blending process to define constraints for functions which could be discovered about such data structures.

The intuition here is that if we take a standard definition of a linked list and a pair, then if we blend the concept of pair into the recursive element of the linked list then this should constitute a binary tree with data at the nodes.

Naïve Attempt Let us initially define again a definition of pair, this time of like-typed elements:

```
spec Pair =
     sort S
```

```
     free type Pair ::= (first:S,second:S)
end
```

and now define the concept of a linked list

```
spec Linked_list =
     sort S
     free type List ::= nil | cons(head:?S,tail:?List)
end
```

and define a generic space once again with two sorts:

```
spec Gen =
  sort G1
  sort G2
end
```

in order to attribute the pair type to the recursive element of linked list we want to try and define a signature morphism as:

```
view I1: Gen to Pair =
     G1 |-> Pair,
     G2 |-> S

view I2: Gen to Pair2 =
     G1 |-> List
     G2 |-> List
```

computing the colimit according to this morphism results in a theory:

```
spec Blend =
    sorts S,Tree
    free type Tree ::= nil
        | cons(head:?S,tail:?Tree)
        | pair(first:Tree,second:Tree)
```

which is not the concept blend one would expect or want. What we want is for the tail element of the cons to be a pair of trees, which is no reflected in this data structure. Unfortunately this data structure is also inconsistent as the destructor functions `first` and `second` are defined as total in the `Pair` type, but cannot be in the free type `Tree` since there are other ways of defining an element of the type.

One could redefine pair as

```
spec Pair =
     sort S
     free type Pair ::= (first:?S,second:?S)
end
```

so as to define the blend:

```
spec Blend =
    sorts S,Tree
    free type Tree ::= nil
          | cons(head:?S,tail:?Tree)
          | pair(first:?Tree,second:?Tree)
```

which is consistent and describes a more complicated structure than a binary tree as it allows construction of a mix between a binary tree with data at the nodes, and one with data at the leaves.

Attempt with Weakening We can investigate the possibility of weakening certain theories before we compute a colimit and determine a blend. Consider revisiting the above example with a weakened definition of linked list:

```
spec Linked_list =
     sorts S1,S2
     free type List ::= nil | cons(head:?S1,tail:?S2)
end
```

and now defining a signature morphism which is:

```
view I1: Gen to Pair =
     G1 |-> Pair,
     G2 |-> S

view I2: Gen to Pair2 =
     G1 |-> S2
     G2 |-> List
```

The calculation of the colimit now results in the blend theory:

```
spec Blend =
    sorts S
    free type Tree ::= nil | cons(head:?S,tail:?Pair)
    Pair ::= pair(left:Tree,right:Tree)
```

which is a typical definition of a data structure describing a binary tree with data at the nodes.

7.2.4 Function Construction

We have shown that concept blending via the computation of colimits can discover new definitions of data structures. A potentially more interesting question is how the colimit computation deals with functions defined on these datatypes. Let us consider introducing some functions to the example in 7.2.2. Imagine we define functions on the definition of `Pair1`:

```
spec Pair1  =
     sort S1,S2
     free type Pair ::= nil | c(S1,S2)
     ops
       func  __   : Pair -> Pair
       func2 __   : S2 -> Pair
       func3 __ __: S1 * Pair -> Pair
     forall x:S1,y:S2
       . func nil = nil
       . func c(x,y) = func3 x (func2 y)
end
```

where we leave the precise definitions of the functions in this incomplete form. When the Colimit is calculated as in Section 7.2.2, the types of the functions become:

```
spec Blend =
   sorts S1,S2,S1',S2'
   free type S2 = c(S1,S2)|nil|c'(S1',S2')
   ops
     func  __    : S2 -> S2
     func2 __    : S2 -> S2
     func3 __ __ : S1 * S2 -> S2
   forall x:S1,y:S2
     . func nil = nil
     . func c(x,y) = func3 x (func2 y)
end
```

as can be seen, the definition of `func2` has now become recursive just allowing us to use the notion of "schemes" (Montano, n.d.) to discover recursive function on the newly defined recursive datatype. For example we can use the IsaScheme system (Montano, n.d.) to generate instantiations for `func3` and `func2`. See §9 for a more detailed discussion on how scheme based approaches can be employed to discover function definitions.

7.3 Analogy in Data Structures

The following example is taken from an exam given to computer science students, and to which the notion of blending applies. Imagine we have defined a new datatype:

```
spec Fatlist =
    sorts S,Fatlist
    free type Fatlist ::= nil
         | cons(S,Fatlist)
         | append(Fatlist,Fatlist)
```

the questions for the students are:

1. Define variants of the list functions length, reverse and map for fat lists. Define a function `null : Fatlist -> bool` which tells if a fat list is empty (contains no elements) or not.

2. For some type `T`, define a function `fold : (S * T -> T) -> T -> Fatlist -> T` that is analogous to foldr on lists. Give a function `f` and a value `v` such that `fold f v l` is `null l` for all fat lists `l`. Give a function `g` and a value `w` such that `fold g w l` is `length l` for all fat lists `l`.

We can exploit blending to determine the answers to some of these questions. Let us define the theory of lists with a function for reverse:

```
spec List =
    sorts S,List
    free type List ::= nil
         | cons(S,List)
    ops
       reverse __: List -> List
       append __ __ : List * List -> List
    forall h:S;t,x:List
       . reverse nil = nil
       . reverse cons(h,t) = append (reverse t) cons(h,nil)
       . append nil x = x
       . append cons(h,t) x = cons(h,append t x)
end
```

Now let us determine the generic space by examining the commonality between Fatlist and List:

```
spec Generic =
    sorts S,Gen
    free type Gen ::= nil
         | cons(S,Gen)
```

allows an incomplete theory to be defined for `Fatlist` when a colimit is calculated. If we assign a signature morphism:

```
view I1: Gen to List =
     S |-> S,
     Gen |-> List,
     nil |-> nil,
     cons |-> cons

view I2: Gen to Fatlist =
     S |-> S,
     Gen |-> Fatlist,
     nil |-> nil,
     cons |-> cons
```

```
spec Blend =
    combine I1,I2
```

the colimit becomes a theory:

```
spec List =
    sorts S,T
    free type T ::= nil
         | cons(S,T) | append(T,T)
    ops
       reverse __: T -> T
       append __ __ : T * T -> T
    forall h:S;t,x:T
       . reverse nil = nil
       . reverse cons(h,t) = append (reverse t) cons(h,nil)
       . append nil x = x
       . append cons(h,t) x = cons(h,append t x)
end
```

now note how functions have been defined on the new datatype `T`. Now we need to construct the missing cases for the function *reverse*:

```
reverse append(t1,t2) = append(reverse t2,reverse t1)
```

In order to tackle the second question we need to extend our analysis to a higher order setting. Our implementation here is incomplete, but it is still possible given a rich enough logic to express the properties we need. If we define `foldr` on lists to be

$$\begin{aligned} foldr\ f\ i\ [] &= i \\ foldr\ f\ i\ cons(h,t) &= f(h, foldr(f,i,t)) \end{aligned}$$

then this gets propagated to the colimit where the definition holds on the new type Fatlist. See §9 for a discussion of how this can be done.

8 Formalised examples

In this section we present some formalised examples of blending in mathematics. These example are not fully implemented yet and are presented here as partially formalised ideas for investigation.

8.1 Quaternions and Octonians

The discovery of the quaternions is an example of a significant mathematical breakthrough. It may very well prove hard to give a convincing route to machine discovery of the notion, starting with the examples of the reals and the complex numbers.

There is quite a lot known about the historical context, which should relate at least anecdotally to the process of discovery.

The story of the discovery is presented for example by Waerden (1976). A rational reconstruction of the discovery process is in Hersh (2011), along with some other examples that are worth looking at. Not least, Hamilton himself wrote about how he found his way to the quaternions — there is a letter by him written immediately after the discovery (Hamilton, 1844a). There are many hints that analogy was playing a crucial role; for example there are 11 occurrences of "analogy" or "analogous" in Hamilton (1844b).

John Baez has written about the historical context in a mathematically informed way (Baez, 2002). For a recent account of the still active research around the quaternions and octonions, see the book Conway and D. A. Smith (2003), and Baez's review (Baez, 2005). Various of Hamilton's mathematical writings are available on-line.[2]

As for looking at the topic in terms of blends, Alexander has a section on imaginary numbers which mentions quaternions and octonions (Alexander, 2011, Sec 8), and Lakoff and Núñez (2000) give some analysis of the situation also.

8.1.1 Quaternions as a blend

The quaternions can be considered as a candidate for discovery using the frameworks described in §3. Below we describe some considerations when formalising the discovering of the quaternions as a blend.

- As for complex numbers, look for a blend of the notion of a field with that of a normed real vector space. In this case we consider the 3-dimensional real vector space by supposing a basis of three vectors of unit size, mutually perpendicular.
- The blend corresponding to that in the 2-dimensional case will turn out to be inconsistent. We can hope that this inconsistency can be noticed by our system; it should be reported in some way.
- Here we can imagine being proposed some weakening of either the field axioms, or those of the normed vector space;

[2] e.g. at `http://www.maths.soton.ac.uk/EMIS/classics/Hamilton/`

- Alternatively we can consider looking at the 4-dimensional case.
  In this case, a consistent blend can be formed, **provided** the commutativity axiom is dropped from the field axioms. We'd like to get out the definition for multiplication of quaternions here.

This says that the quaternions are somewhere in the space of blends, where we allow weakening of input theories, and looking at different dimensions.

This will however not be very close to the discovery route taken by Hamilton, who used in a Lakatos-like way the failure of the attempted construction of an algebra over 3 dimensions to point to the right answer in the 4-dimensional case. The next section gives Hamilton's account of the discovery.

8.1.2 The discovery

> No more important, or indeed fundamental question, in the whole Theory of Quaternions, can be proposed than that which thus inquires What is such MULTIPLICATION? What are its Rules, its Objects, its Results? What Analogies exist between it and other Operations, which have received the same general Name? And finally, what is (if any) its Utility?
>
> Letter from Sir W. R. Hamilton to Rev. Archibald H. Hamilton
> dated August 5, 1865, (Hamilton, 1855)

Hamilton spend a long time trying to find a notion of multiplication of vectors in a three dimensional real vector space, which was intended to include the complex numbers as a substructure. The discovery of a correct notion of multiplication falls into the category of refinement described in §9 and is a very challenging problem.

8.2 Projectile Motion

In this section we consider how it is possible to combine mathematical theories describing motion in one dimension to that in two dimensions. Often this motion is referred to as *projectile motion* and can be generalised to arbitrary dimensions. The idea is to use the notion of conceptual blending to automatically discover a theory which allows calculation of projectile paths in more than one dimension.

8.2.1 Motion in general

Suppose we are looking at motion under a constant force in an n-dimensional space. Here is a small theory describing the situation. Here we have some sort declarations and some definitions, using closed interval notation, and some form of the physical equational theory. This is meant to model the motion of a point mass under a constant force.

We assume we are working with an n-dimensional vector space V. Here we take a single body of mass m to be at the origin at time 0.

$t_1 : \mathscr{R}_{\geq 0}$
$m : \mathscr{R}_{\geq 0}$
$Time = [0, t_1]$
$v_0 : V$
$F : V$
$x : Time \rightarrow V$
$v : Time \rightarrow V$
Physical theory

A model for this theory is given by a set of values for the parameters (t_1, v_0, F) which therefore determine the trajectory, using the physics, by giving the position and velocity functions over the interval $[0, t_1]$.

In terms of categories, we can work with the category of twice differentiable functions from closed intervals in the reals to real normed vector spaces. From the usual category of vector spaces, we get projection morphisms into subspaces; we therefore get "projections" also of paths into such spaces. These projections are continuous w.r.t. the norm. Let's use $P(V_n)$ for such paths in n-dimensional spaces.

Given a 2-D vector space V, we then have a unique decomposition of V as the product of two 1-dimensional spaces, V_F based on vector F, and its orthogonal subspace $V_F^{\perp}$. In fact, we can already be a bit more general: just take the given V to be a finite-dimensional space, which we decompose into a 1-d space based on the force vector, and the orthogonal subspace.

We can now look at the two cases, one in 1-D with constant force, and the other in $(n-1)$-D with zero force. Here there are easy known solutions.

Now the inherited product structure allows us to combine the solutions to solve the given problem.

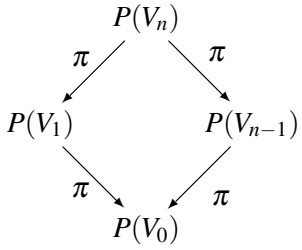
8.2.2 Generic case

Since we are looking for the blend diamond shape to appear, what should we expect for the generic space?

The usual blend based on a syntactic view of theories uses a pushout. So we expect on the model side to look for a pullback.

It looks like the answer here is to make use of the zero-dimensional vector space: this has a single vector $\mathbf{0}$, and of course $x.\mathbf{0} = \mathbf{0}$ for every real x. Then there is a category of (necessarily smooth) maps from closed real intervals into this vector space. Call it $P(V_0)$.

So the claim is that we have a pull back, where each π is a projection:



It looks as though everything is thrown away by the time we get to $P(V_0)$; however what is kept is the existence of a smooth function from a particular closed interval $[0,t_1]$ to some vector space. This is the commonality that must be inherited in the other, syntactic, direction. Furthermore the physics expressed for the general finite-dimensional case holds in the degenerate case of V_0 also.

8.2.3 The syntactic view

To fill in the story more, we should consider how this works out on the more usual presentation of blends, which concentrates on the theories involved, rather than the structures.

A good place to look here is chapter 5 of Barwise and Seligman (1997), in particular section 5.1, which looks at the related situation for classifications. There the authors show that where there are two classifications, the *sum* of the classifications is given by types (roughly, formulas) from either classification, tagged to distinguish their origin; and the tokens (roughly, models) are given by the Cartesian product of pairs of tokens from the two classifications.

So this works nicely here:

- The underlying physics is shared everywhere in the blend.
- The generic theory forces sharing of the time period between the two constituents of the blend.
- The two paths over the shared time are completely independent.
- The sum of the theories of the two paths is realised by importing different versions of v_0, F, x, v into the blend.
- Since we happen to have a good product structure available in the blend, pairs of assertions about the subspaces combine easily into single assertions about the product space.

8.2.4 Comments

- This is a case where the two sides of blending (syntactic/semantic) both give us something interesting. It is open how much of this we want to or will be able to take forward in COINVENT.
- This would be a good example to work through with Hets.

- There is a general notion here of sum of theories where we are interested in two instances of entities defined by the theory, that have no dependence between them. Maybe this needs special attention.
- This is a case where HDTP will want to generalise much more than is appropriate. This may not be a problem — the envisaged scenario is that we start with a given problem, and want to see it as a blend, so the sub-problems cannot be taken to be the same.
- Finally, something can presumably be done where the subspaces chosen are *not* orthogonal, with a bit more work. Still, the product/sum structure is something we should surely be aware of in particular.

8.3 Symmetry Examples

Another potential area of exploration is that of symmetry considerations. In mathematics, often considering potential symmetries of a problem can lead to simplified calculations. The motivation here is to calculate a blend of two theories which are symmetries of each other. This is similar to the notion of the integers given in §3, but is extended to a more general case.

8.3.1 The problem

A version of the problem appears in van Fraassen (1989).

In the situation depicted in figure 4, the vertical line represents a river; the farmer B leaves his farm at point B with a bucket which he will fill with water at the river and take to the geese at point G. What is the shortest path to achieve this?

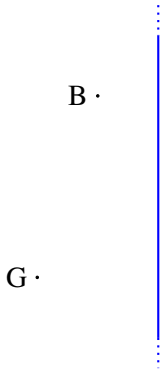


Figure 4: Initial problem layout

One way to solve this is to look at straight line paths via an intermediate point on the river, work out the distance as a function of the position of the intermediate point, and identify the minimum where the derivative vanishes. There is an alternative, geometrical argument using symmetry though.

In a footnote, van Fraassen (1989) credits Martin (1982) with an exposition of the symmetry argument; he also points out that Ostrowski (1968) describes the two solution methods, and relates these to Fermat's reasoning about optical reflection.

Now let us suppose or imagine that there is a mirror image set-up on the other side of the river (farmer A takes water to some Hens); seeing that this would simplify the solution feels like the most creative step in finding the solution.

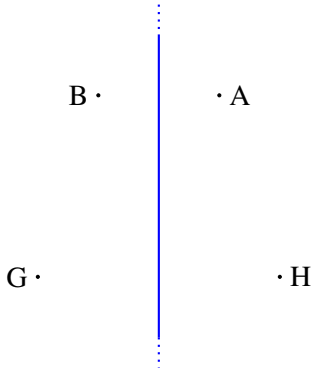


Figure 5: symmetric version of problem

How can this help solve the original problem?

Suppose B goes to the river at some point P and on to the geese, as in figure 6. From the symmetry of the situation, there is a corresponding path for farmer A. Now the distances AP, BP are the same, as are PH and PG. So all four paths (starting from A or B, ending at G or H) have the same length. Thus the minimal path from B to G corresponds to the minimal path from B to H. But in that case the straight line BH crosses the river, so must be the solution. The shortest path therefore goes via P'.

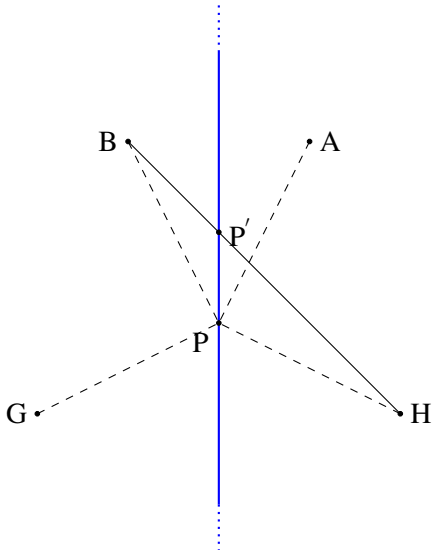


Figure 6: Using symmetry

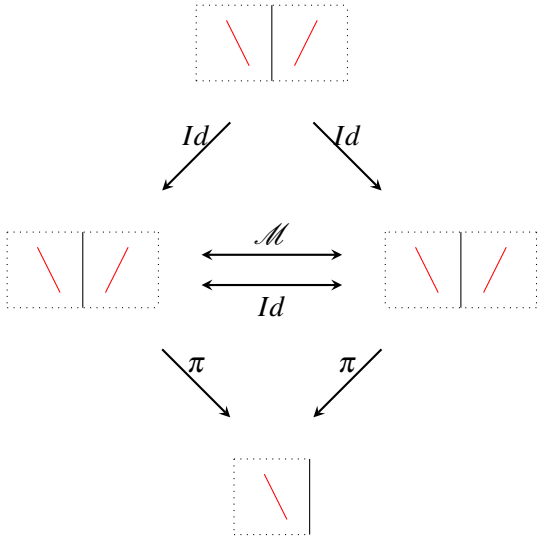
8.3.2 Blend formulation

As for the example of projectile motion, there are two complementary analyses of the situation, one using syntactic theories used to describe the configurations in question, and another that looks at the mathematical structures involved and relations between them (i.e. the models of our theories). Let's look at the latter first.

We can work here with subsets of Euclidean two-dimensional space; let's suppose these are convex, to simplify arguments about paths. Any such convex set X also has a (possibly empty) set of paths; here, paths are given by continuous functions $f : [0,1] \to X$, that are piecewise smooth (since we care about their lengths).

What we want here for morphisms is not isometry of the geometrical spaces concerned, since distance is not always preserved, but the weaker notion of *path isometry*, a.k.a. *arcwise isometry* which preserves the length of curves — see for example Gromov (1999).

The following commutative diagram gives one way to look at the situation.



geometrical symmetry

Here:

- The bottom object is a closed half-plane, with a path;
- The other objects are planes, with a distinguished axis;
- The morphism $\mathscr{M}$ is the mirror mapping based on the distinguised axis.
- The projection π maps the plane, and also any paths, into the half plane — using coordinates, if reflecting around the y-axis, it maps point (x,y) to $(-|x|,y)$. This preserves path length.
- The commutativity condition here is taken to mean that $\mathscr{M}$ and Id are equal (extensionally).

Claim 1 *In the diagram, the top-most object and two arrows form the limit of the remaining part of the diagram.*

This is thus similar to a pull-back, with however extra arrows between the two "input" spaces.

Syntactic formulation For the complementary syntactic version, with arrows in the opposite direction, we need to find congenial axiomatisations for the spaces in question (closed half-plane with paths, plane with distinguished axes and paths); a cut-down version where paths are just for piecewise straight lines would be enough.

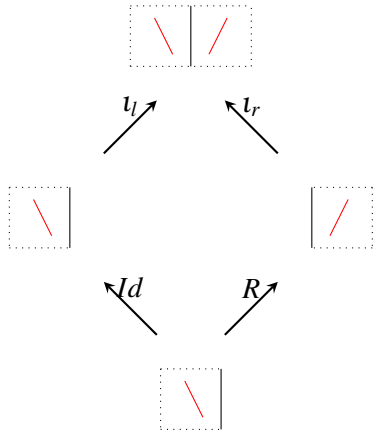
Computing the co-limit should be very easy, assuming the claim above is correct.

Comments It is pleasing that the diagram has the same symmetry as the problem. The diamond-like shape comes from the fact that we are looking at a symmetry of order two, though. And all the action happens in the bottom triangle – the "blend" is equal to either of the input spaces here.

So, although this has pretty much the overall shape we are looking for, this is giving us in fact a way to think about (reflective) symmetry. The symmetry allows us to produce a quotient space by identifying points that correspond under the symmetry.

8.3.3 An alternative?

Below is an alternative forumulation using a different diagram, at the level of the plane/half plane structure. Here the arrows go in the opposite direction. Arrows as before are path isometries.



building the symmetrical space

Here:

- The morphism R produces the mirror image of the half plane, and any paths.

- The injections ι_l, ι_r map the half planes into the whole plane as respectively the left and right regions.

Claim 2 *The diagram gives the symmetric plane as a push-out of the two symmetric variants of the half plane.*

In this case, the formation of the amplified version of the problem comes from computing the push-out of the structures. I do not know what to make of this, in the context of a possible general account of creativity in mathematics.

9 Refinement

When a theory is created by computing a colimit, the resulting blend does not necessarily contain the information necessary for explicit definitions for its function symbols. An interesting problem is to determine whether a refinement can be defined which obeys the laws of the colimit and provides a more concrete theory – for example giving definitions for function symbols. HETS provides a way of extracting the proof obligations which are required to show that a refinement is indeed possible in this sense.

Since the ultimate aim of a system such as HETS is to refine to a theory which is *computable*, each refinement operation is referred to as *stepwise* refinement. If a theory A refines theory B, and hence is more concrete than A, then we say B is an abstraction of A and write

$$B \rightsquigarrow A$$

9.1 Simple Example

As a simple example let us consider `https://ontohub.org/complex-blend/test.dol`, which is a specification of an example small theory. Here a theory `Gen` is created which specifies operations `zero`,`plus` and `minus`. The usual field axioms apply to this operation. We also introduce a theory `Vec`, and want to show that `Vec` $\rightsquigarrow$ `Gen`, when the vector operations are associated with the field axioms in `Gen`. Below we show the fragment of the specification for `Gen` and `Vec` pertaining to the commutativity of symbols `plus`, `vpl` and `+`:

```
spec Gen =
 sort Gen_type
 ops
 __ plus __: Gen_type * Gen_type -> Gen_type;
forall x,y: Gen_type
  . x plus y = y plus x  %com_plus%
end

spec Vec =
 sort Real
 free type Vec ::= pair(x:Real;y:Real)
 ops
 __ + __: Real * Real -> Real;
 op __ vpl__ : Vec * Vec -> Vec;
forall x,y: Real
 . x + y = y + x
 . pair(x,y) vpl pair(z,t) = pair(x+z,y+t)
end
```

Crucially there is a definition for `vplus` in the theory of `Vec`, and the commutativity axiom for `+`. In order to show that associating `vpl` with `plus` in `Gen`, we must show that the definition for `vpl` given by `%def_vplus%` obeys the laws given for `plus` in `Gen`. Through the use of the

`view` operation in the specification file, and using HETS Development Graph calculus, such proof obligations are generated automatically, and it is possible to export them to a theorem prove such as Isabelle and prove them directly.

```
view I: Gen to Vec =
   Gen_type |-> Vec,
   __ plus __ |-> __ vpl __
```

For example in this case, proving commutativity of `vpl` can be done in Isabelle, using the Isabelle theory files generated by HETS.

```
Ax6 [rule_format] :
"ALL X_x. ALL X_y. X_x +' X_y = X_y +' X_x"
Ax9 [rule_format] :
"ALL X_x.
 ALL X_y.
 ALL z.
 ALL t.
  pair(X_x, X_y) vpl pair(z, t) = pair(X_x +' z, X_y +' t)"

theorem Ax2 : "ALL X_x. ALL X_y. X_x vpl X_y = X_y vpl X_x"
apply safe
apply (case_tac X_x)
apply (case_tac X_y)
apply (auto simp add: Ax6 Ax9)
done
```

9.2 The complex numbers

In the example of the complex numbers, we want to be able to find a refinement for the specification given by the computed colimit. This means determining definitions for multiplication over vectors, for example, as described in §7.1.1 and thus in what follows we assume representation of a two dimensional vector as a pair of reals (x,y). The difficulty here is synthesising a definition for multiplication which satisfies the axioms given as specification in the colimit. For example, consider defining a vector multiplication $\cdot$ analogously to `vpl`:

$$(x1,y1)\cdot(x2,y2) = (x1\times x2, y1\times y2)$$

which satisfies axioms for commutativity and for associativity, but causes problems for the definition of a multiplicative inverse. The rules for multiplication for an ordered real field state that

$$(x>0)\wedge(y>0)\rightarrow(x\times y)>0$$

however the vector equivalent which must be true for multiplication is

$$(x>v_0)\wedge(y>v_0)\rightarrow x\cdot y>v_0$$

for which we can find the counterexample $x=(1,0)$ and $y=(0,1)$, where $x\cdot y=v_0$.

One correct version of multiplication is

$$(x1,y1)\cdot(x2,y2) = ((x_1\times x_2)-(y_1\times y_2),(x_1\times y_2)+(y_1\times x_2))$$

for which it is possible to define an inverse and identity. The theory in which we further define multiplication in this way is shown in `https://ontohub.org/complex-blend/complex_numbers.dol`, and we refer to the theory as $\mathscr{C}_{\alpha 1}$. It remains to discharge the proof obligations generated by HETS in showing $\mathscr{C}_{\alpha 1}\rightsquigarrow\mathscr{C}_{\alpha 0}$.

9.2.1 Provability of key properties

As discussed in §7.1.1, we want a theory which preserves the key property

$$N(z_1 \cdot z_2) = N(z_1) \times N(z_2)$$

where N is an Euclidean norm on a two-dimensional vector space. It is possible to add theorems which should be provable in a theory in HETS using the `%implied` directive. As an example of this consider the simple example given at `https://ontohub.org/complex-blend/test.dol`. The theory `Vec` contains the statement

```
forall a:Vec. vminus (vminus a) = a %implied
```

and when we exploit the development graph proof from HETS to prove that `Vec` $\rightsquigarrow$ `Gen` we generate a proof obligation requiring us to prove that this be the case for the more concrete definions of `vminus`, `vpl` and `vzero`.

9.2.2 Theory exploration

The discovery of the definition of multiplication is a difficult and unresolved question which falls into the research area of Theory Exploration. There are various systems which facilitate the discovery of theorems and definitions within a mathematical system such as IsaCosy (Johansson, 2009), Mathsaid (McCasland, Bundy and P. F. Smith, 2005) and IsaScheme (Montano-Rivas et al., 2012). In IsaScheme for example it is possible to state a "scheme" using higher-order variables, and to constrain the functions to which these variables can be instantiated. Sentences are then generated using this scheme and their validity is checked using an associated set of axioms. §7.1.1 shows definitions of functions which we would like to discover. For example let us consider creating a scheme by which to define vector multiplication:

$$\begin{aligned}(x1,x2) \cdot (y1,y2) = (&\mathscr{F}(x1,x2) - \mathscr{F}'(y1,y2),\\ &\mathscr{G}(x1,y2) + \mathscr{G}'(x2,y1))\end{aligned} \tag{33}$$

where each of the higher-order variables $\mathscr{F},\mathscr{F}',\mathscr{G},\mathscr{G}'$ can be instantiated to either $*,+$ or vpl. At each point the expected type constrains this choice - in this case the only choices are $+$ or $*$. Definitions for $\cdot$ are then generated using IsaScheme's synthesis engine, and each sentence involving $\cdot$ in the axiomatisation given by the colimit generated by HETS is checked using a counter-example checker. Any instantiations for which no counter-examples are found are then used to generated statements which can be proved in Isabelle.

As can be seen from this example, user input and expertise is required to find a good starting scheme. Without substantial constraints this process is not realistically possible. A user or expert must provide the following information to constrain the search space:

Initial Scheme the shape of the initial scheme determines the overall shape of the constructed term. As can be seen in (33), the initial starting point is already highly constrained.

Term Pool the function symbols and variables over which the higher-order variables can range. In (33) the term pool is restricted to $*,+$ and vpl.

Search Depth the depth of each synthesised function can be determined. In (33) for example, the search is restricted to depth 1.

9.3 Examples from Data Structures

In §7.2.4 and §7.3 we showed how when applying the idea of blending to data structures, some function symbols exist within the colimit but have yet to be fully defined. We can apply the idea described in this section to these examples. Here let us reconsider the definition of $foldr$ on lists:

$$\begin{aligned} foldr\ f\ i\ [] &= i \\ foldr\ f\ i\ cons(h,t) &= f(h, foldr(f,i,t)) \end{aligned}$$

in the case of the Fatlist data structure we need first to synthesise an extra clause, since Fatlist also has a constructor called `append`:

$$foldr\ f\ i\ append(l1,l2) = foldr(f, foldr(f,i,l2),l1)$$

Discovering this definition is in itself a problem of function discovery which can be done using theory exploration. As described above we need to identify a set of initial function symbols over which we can construct function definitions. In this case for the purposes of presentation, let us just use the function `foldr` itself. We now need som constraints on the function construction - this can be done by creating analogies from theorems about Fatlists. For example:

$$\begin{aligned} append(l,nil) &= append(nil,l) \\ append(append(l1,l2),l3) &= append(l1,append(l2,l3)) \end{aligned}$$

Then given a definition for $foldr$:

$$foldr\ f\ i\ append(l1,l2) = \mathscr{F}(f,i,l1,l2)$$

and the constraints

$$\begin{aligned} \mathscr{F}(f,i,l1,nil) &= \mathscr{F}(f,i,nil,l1) \\ \mathscr{F}(f,i,append(l1,l2),l3) &= \mathscr{F}(f,i,l1,append(l2,l3)) \end{aligned}$$

which can be use to show that generated potential definitions are incorrect.

This part of the work is very speculative currently, as it is necessary to generate many constraints in order to cut down the search space of the generation of possible definitions for functions.

10 Conclusions and further work

The work described in this report gives an overview of some related survey work, followed by several examples worked out in more or less detail; this provides evidence of the the applicability of our initial ideas, and the appropriateness of the tools mentioned (in particular HDTP, DOL and Hets). This is potentially a vast topic, so conclusions at this stage must be tentative. Nevertheless, there are already some lessons that are worthy of note at this stage.

Generic theory In some of the cases examined, HDTP comes up with an appropriate generic theory, if not as first suggestion, then as second. In other cases, such as symmetry reasoning, the generic theory is not motivated by a desire to have a least general anti-unification, but on external considerations of how the input theories should be correlated. It thus appears that the latter case will happen often enough that we should have some way of helping users formulate the generic space in this situation. A possibility here under consideration elsewhere in the project is use of *image schemas*, a cognitively based proposal introduced by Johnson (Johnson, 1987) and Lakoff (Lakoff, 1987)

Choice points Search control here is a hard problem: while things work out fairly straightforwardly for the integers and complex number examples, the quaternion example is more challenging.

Consistency In mathematical cases, consistency is very desirable, and theoretical search algorithms that include consistency checks are problematic in formalisms such us FOL where the property is not decidable. In one case, an initial inconsistency was quickly discovered automatically, but attempts to show consistency typically time out. More experimentation is needed here to see what can be done in practice; identification of sources of inconsistency is also going to be important.

Refinement We see in the complex numbers case that we want a more concrete version of the blend theory than is given by the raw colimit construction. This may come from theory exploration, as described above, or synthesis proofs. We suspect that this corresponds to one version of the notion of "running the blend", in the terminology of Fauconnier and Turner (1998).

Associated theorem proving Here more work is required; the complex numbers case leaves us with Isabelle theorem-proving requirements that seem reasonable, but have not yet been verified. In an interactive tool designed to be creatively provocative, we would like *some* level of automation; it remains to be seen just what is possible here.

Evaluation It is clear from the complex numbers and quaternion cases that we want to be able to make positive requirements of projected blends, in the form of key properties required. We envisage that these may take the form of required theorems in the blend theory, or more generally some properties of the blend that can be computationally tested. There is ongoing discussion with the DOL/Hets researchers on how best to support this.

Recording the process Since our overall goal is to investigate and implement a creative *process*, we clearly have work to do in order to (partially?) automate a blending process. Beyond that,

we need some record of what happens, computationally and creatively, during the process. This again is important for the future success of the project.

References

Alexander, James (2011). 'Mathematical Blending'. In: *Semiotica* 2011.187, pp. 1–48.

Arbib, Michael A. and Mary B. Hesse (1986). *The Construction of Reality*. Cambridge, England: Cambridge University Press.

Argand, Jean-Robert (1813). 'Philosophie Mathématique. Essai sur une manière de représenter les quantités imaginaires, dans les constructions géométriques'. In: *Annales des Mathématiques pures et appliquées* 4, pp. 133–146.

Baez, John C. (2002). 'The Octonions'. In: *Bull. Amer. Math. Soc.* 39, pp. 145–205. URL: http://math.ucr.edu/home/baez/octonions/.

— (2005). 'Review: "On Quaternions and Octonions: Their Geometry, Arithmetic, and Symmetry" by John H. Conway and Derek A. Smith'. In: *Bull. Amer. Math. Soc.* 42, pp. 229–243. URL: http://math.ucr.edu/home/baez/octonions/conway_smith/.

Barwise, Jon and Jerry Seligman (1997). *Information Flow : The Logic of Distributed Systems*. Vol. 44. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press. ISBN: 0521583861.

Bidoit, Michel and Peter D. Mosses (2004). *CASL User Manual*. Lecture Notes in Computer Science 2900. Springer.

Colton, Simon (1999). 'Refactorable Numbers - A Machine Invention'. In: *Journal of Integer Sequences* 2. Article 99.1.2.

Conway, John H. and Derek A. Smith (2003). *On Quaternions and Octonions: Their Geometry, Arithmetic, and Symmetry*. Natick, MA, USA: AK Peters. ISBN: 1-56881-134-9.

Dehaene, Stanislas (1997). *The Number Sense: How the Mind Creates Mathematics*. Oxford University Press.

Faraut, J. and A. Koranyi (1994). *Analysis on symmetric cones*. Oxford Mathematical Monographs. Oxford University Pres. ISBN: 0198534779.

Fauconnier, Gilles and Mark Turner (1998). 'Conceptual Integration Networks'. In: *Cognitive Science* 22.2. Extended version 2001 on-line., pp. 133–187. URL: http://ssrn.com/author=1058129.

Gentner, Dedre, Keith J. Holyoak and Boicho N. Kokinov, eds. (2001). *The Analogical Mind: Perspectives from Cognitive Science*. The MIT Press.

Gromov, Misha (1999). *Metric Structures for Riemannian and Non-Riemannian Spaces*. Modern Birkhäuser Classics. Original French edition: "Structures Métriques des Vari/'et/'es Riemanniennes". Boston, USA: Birkhäuser.

Guhe, Markus et al. (2011). 'A computational account of conceptual blending in basic mathematics'. In: *Cognitive Systems Research* 12.3–4. Special Issue on Complex Cognition, pp. 249–265. DOI: doi:10.1016/j.cogsys.2011.01.004.

Gust, Helmut, Kai-Uwe Kühnberger and Ute Schmidt (2006). 'Metaphors and Heuristic-Driven Theory Projection (HDTP)'. In: *Theoretical Computer Science* 354, pp. 98–117. URL: http://www.sciencedirect.com/science/article/pii/S030439750500856X.

Hamilton, William R. (1844a). 'On Quaternions: Letter to John T. Graves'. In: *London, Edinburgh and Dublin Philosophical Magazine and Journal of Science* XXV, pp. 489–495. URL: http://www.maths.tcd.ie/pub/HistMath/People/Hamilton/QLetter/.

— (1844b). 'On Quaternions; or on a new System of Imaginaries in Algebra'. In: *The London, Edinburgh and Dublin Philosophical Magazine and Journal of Science*. Appeared in instal-

ments 1844–1850. URL: http://www.maths.tcd.ie/pub/HistMath/People/Hamilton/OnQuat/.

— (1855). 'Letter from Sir W. R. Hamilton to Rev. Archibald H. Hamilton'. In: *Life of Sir William Rowan Hamilton*. Ed. by Robert P. Graves. Vol. II. Dublin: Hodges, Figgis & Co. Chap. XXVIII. URL: http://www.maths.tcd.ie/pub/HistMath/People/Hamilton/Letters/BroomeBridge.html.

Hersh, Reuben (2011). 'From Counting to Quaternions – The Agonies and Ecstasies of the Student Repeat Those of D'Alembert and Hamilton'. In: *Journal of Humanistic Mathematics* 1.1, pp. 65–93. URL: http://scholarship.claremont.edu/jhm/vol1/iss1/6.

Johansson, Moa (2009). 'IsaCoSy: Synthesis of Inductive Theorems'. Workshop on Automated Mathematical Theory Exploration (Automatheo).

Johnson, Mark (1987). *The Body in the Mind: The Bodily Basis of Meaning, Imagination, and Reason*. Chicago: Univesrity of Chicago Press.

Kokinov, Boicho, Keith Holyoak and Dedre Gentner, eds. (2009). *New frontiers in analogy research: Proceedings of the 2nd International Conference on Analogy (Analogy '09)*. Sofia, Bulgaria: New Bulgarian University Press.

Lakatos, I. (1976). *Proofs and Refutations: The Logic of Mathematical Discovery*. Cambridge University Press.

— (1974). 'Falsificationism and the Methodology of Scientific Research Programmes'. In: *Criticism and the Growth of Knowledge*. Ed. by I. Lakatos and A. Musgrave. Cambridge: Cambridge University Press, pp. 91–196.

Lakoff, George (1987). *Women, Fire, and Dangerous Things: What Categories Reveal About the Mind*. Chicago: University of Chicago Press.

Lakoff, George and Rafael Núñez (2000). *Where Mathematics Comes From: How the Embodied Mind Brings Mathematics into Being*. New York: Basic Books.

Lange, Christoph et al. (2012). 'The Distributed Ontology Language (DOL): Ontology Integration and Interoperability Applied to Mathematical Formalization'. In: *AISC/MKM/Calculemus*. Ed. by Johan Jeuring et al. Vol. 7362. Lecture Notes in Computer Science. Springer, pp. 463–467. ISBN: 978-3-642-31373-8.

Langley, P. et al. (1987). *Scientific Discovery: Computational Explorations of the Creative Processes*. Cambridge, Mass.: MIT Press.

Lenat, D. B. (1982). 'AM: An Artificial Intelligence approach to discovery in Mathematics as Heuristic Search'. In: *Knowledge-based systems in Artificial Intelligence*. Also available from Stanford as TechReport AIM 286. New York: McGraw Hill.

Martin, George E. (1982). *Transformational Geometry*. Undergraduate Texts in Mathematics. New York: Springer.

McCasland, R. L., A. Bundy and P. F. Smith (2005). 'Ascertaining Mathematical Theorems'. In: *Proceedings of Calculemus 2005*. Ed. by J. Carette and William M. Farmer. Newcastle, UK.

Montano, Omar. *Isascheme*. web site. at http://dream.inf.ed.ac.uk/projects/isascheme/. URL: http://dream.inf.ed.ac.uk/projects/isascheme/.

Montano-Rivas, O. et al. (2012). 'Scheme-based Theorem Discovery and Concept Invention'. In: *Expert Systems with Applications* 39.2, pp. 1637–1646.

Mossakowski, Till, Christian Maeder and Klaus Lüttich (2007). 'The Heterogeneous Tool Set'. In: *TACAS 2007*. Ed. by Orna Grumberg and Michael Huth. Vol. 4424. Lecture Notes in Computer Science. Springer-Verlag Heidelberg, pp. 519–522. URL: www.informatik.uni-bremen.de/~till/papers/hets-tacas-toolpaper.pdf.

Ostrowski, A. (1968). *Differential and Integral Calculus*. Vol. 1. Glenview, Ill.: Scott, Foresman & Co.

Pease, Alison (2007). 'A computational model of Lakatos-style reasoning'. Doctoral dissertation. University of Edinburgh.

Pease, Alison, Simon Colton et al. (2010). 'Using Analogical Representations for Mathematical Concept Formation'. In: *Model-based Reasoning in Science and Technology: Abduction, Logic, And Computational Discovery*. Ed. by Lorenzo Magnani, Walter Carnielli and Claudio Pizzi. Studies in Computational Intelligence 341. Springer, pp. 301–314. URL: http://springerlink.com/content/y1t348758g46q462/fulltext.pdf.

Pease, Alison, Markus Guhe and Alan Smaill (2009). 'Analogy formulation and modification in geometry'. In: *New Frontiers in Analogy Research: Proceedings of the 2nd International Conference on Analogy (Analogy '09)*. Ed. by Boicho Kokinov, Keith Holyoak and Dedre Gentner. Sofia, Bulgaria: New Bulgarian University Press, pp. 358–364.

— (2010). 'Using analogies to find and evaluate mathematical conjectures'. In: *Proceedings of the International Conference on Computational Creativity*. Ed. by Dan Ventura et al. Portugal: Departament of Informatics Engineering, University of Coimbra, pp. 60–64.

Poincaré, Henri (1908). 'L'avenir des mathématiques'. In: *Revue générale des sciences pures et appliquées* 19, pp. 930–939.

— (1914). 'The future of Mathematics'. In: *Science and Method*. Trans. by Francis Maitland. translated from Poincaré, 1908. London: Thomas Nelson and Sons. URL: http://portail.mathdoc.fr/BIBLIOS/PDF/Poincare.pdf.

Schlimm, Dirk (2008). 'Two Ways of Analogy: Extending the Study of Analogies to Mathematical Domains'. In: *Philosophy of Science* 75, pp. 178–200. DOI: 10.1086/590198.

— (2009). 'Bridging Theories with Axioms: Boole, Stone and Tarski'. In: *New Perspectives on Mathematical Practices*. Ed. by B. van Kerkhove. World Scientific, pp. 222–235. URL: http://www.cs.mcgill.ca/~dirk/schlimm-BridgingTheoriesWithAxioms-penultimate.pdf.

— (2011). 'On the creative role of axiomatics. The discovery of lattices by Schröder, Dedekind, Birkhoff, and others'. In: *Synthese* 183.1. URL: http://www.springerlink.com/openurl.asp?genre=article&id=doi:10.1007/s11229-009-9667-9.

— (2013). 'Axioms in mathematical practice'. In: *Philosophia Mathematica* 21.1, pp. 37–92. URL: http://philmat.oxfordjournals.org/content/21/1/37.

Schmidt, Martin (2010). *Restricted Higher-Order Anti-Unification for Heuristic-Driven Theory Projection*. PICS-Report 31-2010. Germany: Univ. Osnabrück.

Thagard, Paul (1993). *Computational Philosophy of Science*. Cambridge, MA: MIT Press.

Turner, Mark (2005). 'Mathematics and Narrative'. In: *International Conference on Mathematics and Narrative*. Mykonos, Greece. URL: http://www.thalesandfriends.org/en/papers/pdf/turner_paper.pdf.

Turner, Mark and Gilles Fauconnier (1995). 'Conceptual Integration and Formal Expression'. In: *Metaphor and Symbolic Activity* 10.3, pp. 183–204. DOI: 10.1207/s15327868ms1003_3.

van Fraassen, Bas C. (1989). *Laws and Symmetry*. Oxford: Oxford University Press.

Waerden, B. L. van der (1976). 'Hamilton's discovery of Quaternions'. In: *Mathematics Magazine* 49.5, pp. 227–234. URL: http://www.jstor.org/stable/2689449.

Weil, André (1960). 'De la métaphysique aux mathématiques'. In: *Sciences*. in Weil, 1979, pp 408–412. URL: http://dream.inf.ed.ac.uk/projects/coinvent/weil60.pdf.

— (1979). *Œuvres Scientifiques/ Collected Papers*. Vol. II. Corrected second printing. New York: Springer-Verlag.