
D2.1

Reasoning with Amalgams

| | |
|-----------|---|
| Authors | Félix Bou, Manfred Eppe, Enric Plaza, Marco Schorlemmer |
| Reviewers | |

| | |
|---------------------|-------------------------------------|
| Grant agreement no. | 611553 |
| Project acronym | COINVENT - Concept Invention Theory |
| Date | October 15, 2014 |
| Distribution | PU |

Disclaimer

The information in this document is subject to change without notice. Company or product names mentioned in this document may be trademarks or registered trademarks of their respective companies.

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

In this deliverable we survey how Fauconnier and Turner's theory of conceptual blending has been implemented computationally, and we do an in-depth theoretical exploration of the mathematical framework proposed by Goguen to model conceptual blending in a domain- and system-independent manner. We also discuss several alternative categorical frameworks that still capture Goguen's basic insights and propose a formal model of blending that is also a generalisation of the closely related notion of 'amalgam', originally proposed as a method for knowledge transfer in case-based reasoning. A computational realisation of this model is presented by means of a well-known example of creative thinking, specified in the CASL specification language and implemented in the Hets reasoning system.

Keywords: **conceptual blending, amalgams, category theory, colimits, image schemas, computational creativity, concept invention.**

Contents

| | | |
|----------|---|-----------|
| 1 | Introduction | 1 |
| 2 | Background | 2 |
| 2.1 | Conceptual Blending | 2 |
| 2.2 | Constitutional Elements of Conceptual Blending | 2 |
| 2.3 | Optimality Principles | 5 |
| 2.4 | The Generic Space | 6 |
| 2.4.1 | The CONTAINER Image Schema | 7 |
| 2.4.2 | The SOURCE-PATH-GOAL Image Schema | 8 |
| 3 | Computational Models of Conceptual Blending | 8 |
| 3.1 | The Alloy Algorithm and the Griot System | 8 |
| 3.1.1 | Constitutive Elements of Conceptual Blending in Alloy | 8 |
| 3.1.2 | Optimality Principles | 10 |
| 3.1.3 | Structural Blending | 10 |
| 3.1.4 | Strengths and Weaknesses of Alloy and Griot | 11 |
| 3.2 | Sapper | 11 |
| 3.2.1 | Constitutive Elements of Blending | 12 |
| 3.2.2 | Optimality Principles | 13 |
| 3.2.3 | Strengths and Weaknesses | 14 |
| 3.3 | Divago | 15 |
| 3.3.1 | Constitutive Elements of Blending | 16 |
| 3.3.2 | Optimality Principles | 18 |
| 3.3.3 | Strengths and Weaknesses | 20 |
| 3.4 | Blending by Heuristic-Driven Theory Projection (HDTP) | 20 |
| 3.4.1 | Constitutive Elements of Blending | 21 |
| 3.4.2 | Optimality Principles | 22 |
| 3.4.3 | Strengths and Weaknesses | 22 |
| 3.5 | A Neurological Account on Combinatorial Creativity | 23 |
| 3.5.1 | Constitutive Elements of Blending | 24 |
| 3.5.2 | Optimality Principles | 25 |
| 3.5.3 | Strengths and Weaknesses | 25 |
| 4 | A Formal Theory of Blending as Colimits and Amalgams | 26 |
| 4.1 | Blending as Colimits | 26 |
| 4.1.1 | Category Theory Preliminaries | 26 |
| 4.1.2 | Colimits in Ordered Categories | 39 |
| 4.2 | Blending as Amalgams | 46 |
| 4.2.1 | Amalgams | 47 |
| 4.2.2 | Comparing Amalgams and Blends | 49 |
| 4.2.3 | An Example: Computer Icons | 50 |
| 4.3 | Relating Colimits and Amalgams | 53 |
| 4.3.1 | Preliminaries | 53 |
| 4.3.2 | A Category-Theoretical Account of Amalgams | 54 |

| | | |
|----------|---|-----------|
| 5 | The Theory at Work | 55 |
| 5.1 | The Buddhist Monk Riddle | 55 |
| 5.2 | CASL and its Application for Blending | 56 |
| 5.3 | The Formalisation of the Riddle | 58 |
| 5.3.1 | Input Spaces | 60 |
| 5.3.2 | The Generic Space | 60 |
| 5.3.3 | Composition of the Blend – Amalgamating the Input Spaces | 61 |
| 5.3.4 | Completion of the Blend – Adding Background Knowledge | 64 |
| 5.3.5 | Elaboration of the Blend – Proving the Riddle | 65 |
| 6 | Concluding Thoughts | 67 |
| | Bibliography | 67 |
| A | Characterisation of $\frac{3}{2}$-colimits in Pfn | 74 |

1 Introduction

This deliverable accounts for the work done under Task T1 of work package WP 2, where the goal is to develop a computational model of conceptual blending based on amalgams while taking into account previous work on formal and computational approaches to the cognitive science notions involved in conceptual blending.

The notion of amalgams in a lattice of generalisations was developed at IIIA-CSIC in the framework of modelling analogical inference, and case amalgamation in case-based reasoning (CBR). Case amalgamation models the process of combining two different cases into a new *blended* case to be used in the CBR problem solving process. As such, the notion of amalgam seems related to but not identical to the notions of conceptual blending and belief merging. These related notions have in common that there is some combination or fusion of two different sources into a new entity that encompasses selected parts of the sources, but they differ in the assumptions on the entities upon they work: amalgams work on *cases* (expressed as terms in some language), conceptual blending works on *mental spaces*, and merge operations work on *sets of beliefs*.

Our motivation being that the account on conceptual blending by Fauconnier and Turner [22] is very detailed and comprehensive from a cognitive viewpoint, but it lacks a clear formal description which is necessary to realise *computational* creativity. We take the approach of generalising the original notion of amalgam from CBR to see if it was amenable to be used in the development of a theory of conceptual blending that was close to, and even compatible with, Goguen's work on blending [24]. This means taking a category-theoretic approach to model amalgams in the world of conceptual blending. Moreover, we also take into account preexisting computational models of conceptual blending, regardless of the theoretical approach taken.

For this purpose, after this introduction, we start by summarising the cognitive science notions relevant to conceptual blending in the [Section 2](#). There have been several attempts to formalise the ideas of [Fauconnier and Turner](#), which lay some important grounds and show how blending can be formalised for certain domains or certain representations of input spaces. Hence, in [Section 3](#) we review the existing computational models of conceptual blending. However, existing approaches are either not *general* enough to capture the idea of conceptual blending in a way that is independent from domain and representation, or they lack the implementation of other ideas behind blending theory, such as optimality principles that govern what a good blend is (see [Section 2.3](#)). The category-theoretic approach to model conceptual blending as colimits and amalgams is presented in [Section 4](#), which comprises the core formal contribution of this deliverable. To better understand this contribution, [Section 5](#) develops in detail a well-known conceptual blending example, the Buddhist Monk Riddle, using the theory presented in the previous section. The deliverable's last section presents some concluding thoughts on the approach taken, the results so far obtained and the road ahead.

2 Background

2.1 Conceptual Blending

Creativity understood as unfamiliar combinations of familiar ideas goes back to the notion of *bisociation*, presented by Arthur Koestler in his book *The Act of Creation* in 1964 [41], but cognitive science has not until more recently proposed approaches on how to produce novel ideas (concepts, theories, solutions, works of art). This approach, known as the theory of *conceptual blending* or *conceptual integration* has been proposed by Fauconnier and Turner [21] as a kind of primitive or fundamental cognitive operation underlying much of everyday thought and language. The process by which two concepts are *blended* into a novel idea is a complex process by which particular elements and their relations pertaining to the initial two concepts are combined into a new whole that is more than the mere commonalities of the two concepts.

For instance, a ‘houseboat’ or a ‘boathouse’ are neither the intersection nor the union of the concepts of ‘house’ and ‘boat’. Instead, the concepts ‘houseboat’ and ‘boathouse’ selectively integrate different aspects of the input concepts ‘house’ and ‘boat’ in order to produce two different new concepts, ‘houseboat’ and ‘boathouse’, each with its own distinct internal structure. For instance, consider the ‘houseboat’ blend shown in Figure 1. According to the dictionary a houseboat is “a vessel, such as a barge, used as a dwelling.” From the point of view of blending, ‘houseboat’ is a new concept where the house (dwelling object) rests on water instead of land, and the person is considered a resident (dweller) instead of a passenger on a boat. Thus, only *selected* parts of the input mental spaces are projected into the blend: a person that is a resident (but not passenger) that lives on a *dwelling* (called ‘houseboat’) that rests on water (and not on land).

Fauconnier’s view of concepts is prior to the notion of blending, and his *Mental Spaces Theory* is a highly influential cognitive theory of *meaning construction*, developed in [19] and [20]. According to Fauconnier, meaning construction involves two processes: (1) the building of *mental spaces*; and (2) the establishment of *mappings* between those mental spaces. Moreover, the mapping relations are guided by the local discourse context, which means that meaning construction is always situated or context-dependent.

2.2 Constitutional Elements of Conceptual Blending

Fauconnier and Turner [22] describe several constitutive elements of conceptual blending. These are (i) the *input spaces* that are to be blended, (ii) a partial *cross-space mapping* that connects counterparts in the input mental spaces, (iii) a *generic space* that is an abstraction from what the input spaces have in common, (iv) a *blending* operation that produces a blend of the input spaces and into which the structure of the input spaces is selectively projected, and (v) an *emergent structure*, i.e., structure that is a synergistic gain to the naive sum of the structure of input spaces.

These constitutive elements can be organised in a *conceptual integration network*, i.e., the network of all input spaces, generic spaces and blend spaces together with the selective projections that model a particular blending process. Finally, Fauconnier and Turner propose certain *optimality principles* that govern the blending process, and that can be taken as a way to assess the quality of a blend. Let us briefly review these constitutive elements and optimality principles as put forth in Fauconnier and Turner’s model.

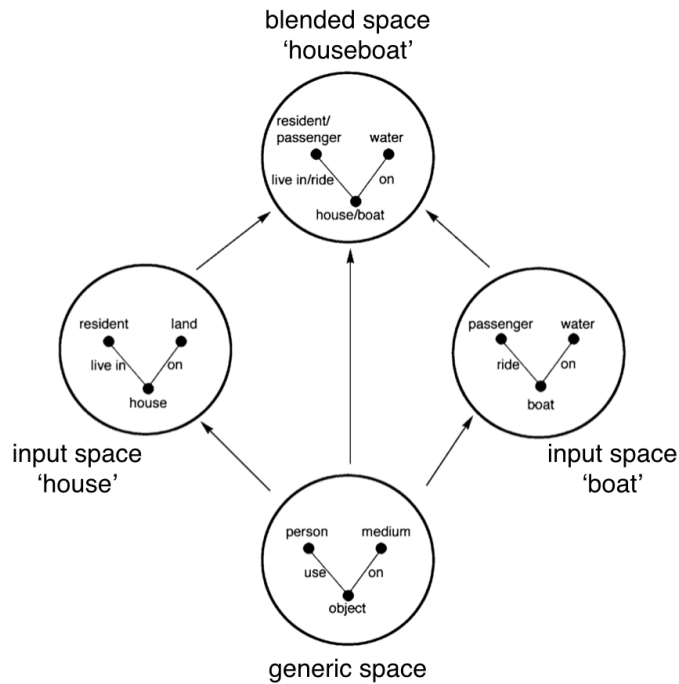


Figure 1: The ‘houseboat’ blend, adapted from [30].

Input Spaces. Fauconnier and Turner [22] consider the input spaces of a blend to be *mental spaces* —small conceptual packets constructed as we think and talk, for purposes of local understanding and action. According to Fauconnier [19], mental spaces are connected to long-term schematic knowledge, by means of *frames*, which model how certain elements and relations are organised as a package that we already know about. Often these frames are so-called *image schemas* —a recurring dynamic pattern of our perceptual interaction and motor programs that gives coherence and structure to our experience. An example of frame of this sort is the SOURCE-PATH-GOAL schema which underlies our experience of walking along a path. Mental spaces have also access to long-term specific memory, e.g., the episode in your memory of climbing Mont Blanc last year.

Moreover, mental spaces are built on-demand, for a particular linguistic situation. For example, contrary to a naive view of meaning, there is no fixed understanding of the concepts of ‘house’ and ‘boat’: there are mental spaces for ‘house’ or ‘boat’ that are constructed in particular linguistic situations. Hence, one cannot assume that input mental spaces, such as those of ‘house’ or ‘boat’, are the same in all possible output blends, since the content on a mental space dubbed ‘house’ or ‘boat’ is neither constant nor preexisting to the context in which they are used. However, many computational blending systems (e.g., [32, 74]) that emanate from the work by Fauconnier and Turner [22] often simplify this aspect of their theory and consider blending to be an operation that takes two input spaces as given and generates one or more output spaces, i.e., blends.

Cross-space mapping. According to [22, p. 41], “[...] a partial cross-space mapping connects counterparts in the input mental spaces.” As an example consider the ‘house-boat’ blend, where the passenger of the boat is mapped to the resident of the house, and the *ride* relation between

passenger and *boat* is mapped to the *live-in* relation between *resident* and *house*.

Generic Space. The cross-space mapping points to what might constitute the generic space, which essentially contains what is common to or shared among the two input mental spaces. For example, the mapping between *passenger* and *resident* in the house-boat blend suggests a generic space with a concept *person* that generalises the concepts of *passenger* and *resident*.

Blend. The most precise statement about the actual blending operation is possibly the following, from Fauconnier and Turner [22, p. 47]: “*In blending, structure from two input mental spaces is projected to a new space, the blend. Generic spaces and blended spaces are related: Blends contain generic structure captured in the generic space but also contain more specific structure, and they can contain structure that is impossible for the inputs [...]*”

The blend is neither the intersection nor the union of the input spaces. Parts of the input spaces are selectively projected into the blend, other parts do not become part of the blend. For example, blending ‘house’ and ‘boat’ to ‘houseboat’ requires to select the concept *water* from the ‘boat’ input space, but not the concept *land* from the ‘house’ input space.

The close relationship between the generic and the blend spaces, makes it important for any formal model of blending to come to grips how the generic space arises from the blending process as well as what role it plays. In [Section 2.4](#) we shall come back to this issue and propose several approaches to what can constitute the generic space or spaces of a conceptual integration network.

Emergent Structure – Composition, Completion and Elaboration. The emergent structure refers to additional structure in the blend that is not directly copied from the inputs. Emergent structure is built through three processes: composition, completion, and elaboration. First, *composition* simply brings the elements of the input spaces together, without any further effect; second, *completion* is the inference of additional information, obtained from accessing related frames and scenarios; and third, *elaboration* intuitively means simulating the behaviour of the elements in the blend interacting among them. As emphasised in Fauconnier and Turner [22, p. 49], “[t]he creative possibilities of blending stem from the open-ended nature of completion and elaboration.”

From an Artificial Intelligence viewpoint, composition combines a selective projection of inputs. Completion adds background knowledge to the blend, until it is completed, i.e., has all that is needed for the purpose at hand. Elaboration involves reasoning or inference, in the sense that background knowledge is used to reason about actions that some elements can perform, or actions that can be performed on some elements, in order to elucidate the eventual consequences of the blended mental space.

Conceptual Integration Network. As stated in Fauconnier and Turner [22, p. 44], “*Building an integration network involves setting up mental spaces, matching across spaces, projecting selectively to a blend, locating shared structures, projecting backward to inputs, recruiting new structure to the inputs or the blend, and running various operations in the blend itself.*” An example for an integration network is the basic houseboat diagram in [Figure 1](#). There are four mental spaces: the two inputs, the generic space, and the blend. However, this example is only a minimal

network. In general, “[...] *conceptual integration networks can have several input spaces and even multiple blended spaces.*” [22, p. 47].

2.3 Optimality Principles

Optimality principles play the role of an assessment measure for blends. As we mention earlier, two input spaces can be blended together in many different ways, and finding a particular blend that is creative and useful is not trivial.

The optimality principles that Fauconnier and Turner [22] mention are defined in an “other things being equal”-manner, i.e., they compete with each other, and satisfying one principle may dissatisfy another one. In the following we summarise how Fauconnier and Turner [21, 22] define the particular principles that we consider in this paper, and we provide examples by explaining their role in the ‘house-boat’ blend.

- *Topology Principle:* “For any input space and any element in that space projected into the blend, it is optimal for the relations of the element in the blend to match the relations of its counterpart.” [21]

In terms of the ‘houseboat’ blend for example, if the elements *passenger* and *resident* are blended to a *passenger-resident* of a *houseboat*, then the *lives-in* relation between the *resident* and the *house* remains the *lives-in* relation between the *passenger-resident* and the *houseboat*.

- *Pattern Completion Principle:* “Other things being equal, complete elements in the blend by using existing integrated patterns as additional inputs. Other things being equal, use a completing frame that has relations that can be the compressed versions of the important outer-space vital relations between the inputs.” [22, p. 328]

This principle is related to the work on image schemas by Lakoff [46]. Image schemas are completing frames that are abstract versions of the input spaces. For example, in the houseboat blend, the CONTAINER image schema is used as an abstraction for a house, which is a container for a resident and a boat, which is a container for a passenger. The pattern completion principle is related to the *emergent structure* of a blend, because emergent structure only arises through completion and elaboration.

- *Integration Principle:* “The blend must constitute a tightly integrated scene that can be manipulated as a unit. More generally, every space in the blend structure should have integration.” [21]

As an example consider the ‘houseboat’ blend again. One could blend a scene of a house and a scene of a boat as the simple union of both scenes, so that there is a house with a resident and a boat with a passenger in the blend. However, this is not perceived well as a unit. A better blend in terms of integration is the new concept of ‘houseboat’, which can be treated much better as a unit.

- *Maximisation of Vital Relations Principle:* “Other things being equal, maximise vital relations in the network. In particular, maximise the vital relations in the blended space and reflect them in outer-space vital relations.” [22, p. 330]

According to Fauconnier and Turner [22], blending does not only happen to creatively invent new concepts, but it also serves as a means to compress the information in the input spaces using so-called *vital relations*, which are fundamental in the particular network of interest. As examples, Fauconnier and Turner [22] mention cause-effect, time, space, identity, change, and uniqueness relations. If such relations exist between the input spaces, then blending causes them to reappear in compressed form in the blend. Maximising vital relations in a blend also maximises the degree of compression that a blend produces. This underpins not only the importance of blending as a means for creativity, but also for efficient cognitive operation.

- *Web Principle:* “Other things being equal, manipulating the blend as a unit must maintain the web of appropriate connections to the input spaces easily and without additional surveillance or computation.” [22, p. 331]

This refers to relations between spaces in the network. For example, placing a houseboat into a new environment such as a river scene maintains the connections to the input spaces of house and boat: the relation between the *resident* of a *house* and the *passenger-resident* of a *houseboat* is still there. As another example, consider the case where some form of resident fee has to be paid for a houseboat. If the fee is raised for a houseboat, then one should be able to infer that it is also raised for a house.

- *The Unpacking Principle:* “Other things being equal, the blend all by itself should prompt for the reconstruction of the entire network.” [22, p. 332]

For example, by focussing on the concept of a ‘houseboat’ we still can access the concept of ‘house’ with its properties and its relationship to other concepts. Similarly, we can also access the notion of ‘boat’ with its properties and its relationship to other concepts.

- *Relevance Principle:* This is sometimes also called *Good Reason*. Fauconnier and Turner [22, p. 333] describe it as follows: “Other things being equal, an element in the blend should have relevance, including relevance for establishing links to other spaces and for running the blend. Conversely, an outer-space relation between the inputs that is important for the purpose of the network should have a corresponding compression in the blend.”

What eventually will constitute the ‘houseboat’ concept will very much depend on what we are pursuing when blending. So, if the relation *live-in* is relevant, it should be included in the blend. But maybe another relation (say *number-of-rooms*) might not be relevant anymore, and should not be considered in the blend.

2.4 The Generic Space

In Section 2.2 we mention that determining the generic space for a conceptual integration network is crucial for creating a blend. We shall consider two alternative approaches to what the generic space of given input spaces might be.

A generalisation of the input spaces. One way to determine what is common to the input spaces is by means of looking at the cross-space mapping between them. Hence, structural mapping techniques that identify isomorphic substructures of the inputs might be useful to create an

abstraction of this substructure. This is the approach described in project deliverable D1.1 [5]. There, cross-space mappings are established by means of the HDTP algorithm [68], which computes a restricted higher-order anti-unification of two input spaces represented as first-order logical theories. This anti-unification then serves as a generic space for the blend of the original first-order theories. Project deliverable D1.1. describes this approach in detail and illustrates it for blending in the project working domains of mathematics and music.

Image schemas framing the structure of input space. An alternative is to adopt an approach that is grounded of an embodied understanding of cognition, and to focus on the image schemas that underlay the structure given in the input spaces.

The theory of image schemas was jointly proposed by Lakoff [46] and Johnson [37] and is constitutive of the view that human cognition is essentially grounded on our bodily experience with our environment, and that this embodied experience is at the heart of how we structure our concepts, even the most abstract ones. Quoting Johnson, “*an image schema is a recurring dynamic pattern of our perceptual interaction and motor programs that gives coherence and structure to our experience.*” [37]

Hampe [32] provides the following characterisations: Image schemas are *directly meaningful* (“experiential”/“embodied”), *pre-conceptual structures*, which arise from or are grounded in human recurrent bodily movements through space, perceptual interactions, and ways of manipulating objects; are highly *schematic gestalts* that capture the structural *contours* of sensory-motor experience, integrating information from multiple modalities; exist as *continuous* and *analogue* patterns *beneath* conscious awareness, prior to and independently of other concepts; and are both *internally structured* and highly *flexible*.

Lakoff [46] and Johnson [37] identify several of these image schemas and show how they ground on our bodily experience the meaning we give to abstract concepts and situations. Because of the central role they play in structuring our concepts, they may also serve to identify shared commonalities between concepts or conceptual spaces. Consequently they may be constitutive elements of the generic spaces of a conceptual blend. As a first exercise, in Section 5 we show a conceptual integration network that adopts this approach to structure the generic space. Below we briefly review two of these image schemas which we shall use in the discussions contained in this deliverable.

2.4.1 The CONTAINER Image Schema

This image schema —sometimes also referred to as the CONTAINMENT schema— consists of three parts: an *interior*, a *boundary*, and an *exterior*.

The structure of image schemas form a gestalt, which means that the parts only make sense in relationship to each other, and cannot be isolated.

Image schemas come also “equipped” with an internal logic and built-in inferences by virtue of which we do our reasoning. So if, for instance, an object is in the interior of a container *A* that is in the interior of the container *B*, then this object is also in the interior of the container *B*. We shall use this image schema in Section 4.2

2.4.2 The SOURCE-PATH-GOAL Image Schema

This image schemas —sometimes also referred to as simply the PATH schema— is presented in slightly different ways in the literature, but in its simplest form it consists of: a *source* or starting point, a *goal* or end-point, and a *path* or sequence of contiguous locations connecting the source with the goal. As such, the SOURCE-PATH-GOAL is a specialisation of another image schema, called the LINK schema, which consists of two entities and a link between them. Here, we have a source and a goal location which are linked by a path.

Since the SOURCE-PATH-GOAL schema arises from our bodily experience of moving about in space, it often includes the notion of a *trajectory* moving from source to goal, and the associated notion of this trajectory “being on the path”. Furthermore, because our experience of moving about in space is tightly linked with our perception of time, often the temporal dimension is also included into the schema, so as to take into account that the trajectory starts at a certain time at the source position and ends at a later time at the goal position, being on the path at any intermediate time instance.

The internal logic and built-in inferences of this schema allow us to state, for instance, that if somebody has travelled from *A* to *B* and from *B* to *C*, then he or she has travelled from *A* to *C*. We shall see a formalisation of this schema in [Section 5](#).

3 Computational Models of Conceptual Blending

Several approaches of formal and computational models for conceptual blending have been proposed in the past. Many of these models are inspired by the work of Fauconnier and Turner [22], but there are also other approaches emanating from analogical reasoning [69] and neuroscience [71].

In this section we present a survey on the most relevant existing work, and describe how the different formalisations relate to Fauconnier and Turner’s constitutional elements and optimality principles that we summarise in [Section 2.2](#). We conclude each approach by summarising its strengths and weaknesses.

3.1 The Alloy Algorithm and the Griot System

The Alloy algorithm [29] for conceptual blending incorporates many ideas of the algebraic semiotics approach by Goguen [24] and the conceptual blending theory by Fauconnier and Turner [21, 22]. Alloy has been integrated in the Griot system for automated narrative generation [29, 34, 33]. Apart from the primary *conceptual* blending approach realised with Alloy, Griot also uses a secondary *structural* blending mechanism that blends the dynamic elements of natural language narratives to generate poetry.

3.1.1 Constitutive Elements of Conceptual Blending in Alloy

Input Spaces. The input spaces of the Alloy algorithm are theories defined in the algebraic specification language BOBJ (see e.g. [52]). This allows one to represent sorts, operators, constants

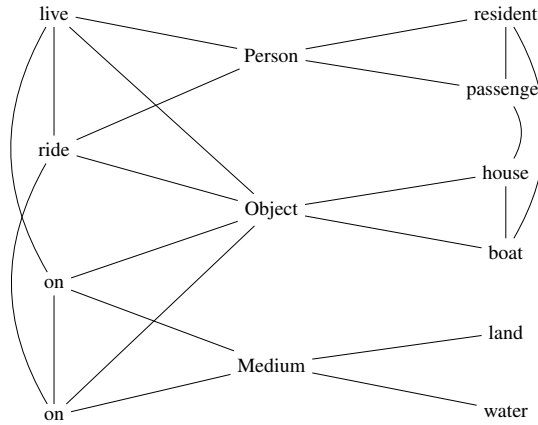


Figure 2: The input graph for the Alloy algorithm and the blend of HOUSE and BOAT

and axioms. As an example consider the following specifications of the concepts HOUSE and BOAT, as depicted in Figure 1.

```

th HOUSE is
  sorts Object Person.
  pr MEDIUM.
  op resident : -> Person.
  op house    : -> Object.
  op live-in  : Person Object -> Bool.
  op on       : Object Medium -> Bool.
  eq live-in(resident, house) = true.
  eq on(house, land) = true.
endth

th BOAT is
  sorts Object Person.
  pr MEDIUM.
  op boat      : -> Object.
  op passenger  : -> Person.
  op ride      : Person Object -> Bool.
  op on        : Object Medium -> Bool.
  eq ride(passenger, boat) = true.
  eq on(boat, water) = true.
endth

```

The keyword `sorts` denotes sort definitions, `op` denotes operators, and `eq` denotes axioms. The keyword `pr` indicates that theories inherit structure from a parent theory called `MEDIUM`, that defines a data sort `Medium` with constants `land` and `water`.

Before the actual blending happens, Alloy maps the relations, data sorts, non-data sorts and constants of the input theories to a graph with three columns (see Figure 2). The left column of the graph has as nodes relations, in the middle column there are sorts, and on the right are constants. Edges between relations and sorts indicate that the relation has an argument of the respective sort, and edges between relations denote that the relations come from different input spaces. This is

also done analogously for constants.

Blend. The blending algorithm generates two binary trees that are based on the input graph. The two trees represent (i) the space of possible mappings of relations and (ii) the space of possible mappings of constants respectively. The trees are then combined in the sense that the leaves of the constants-tree are applied to the relations-tree. The resulting combined tree has leaves that represent all possible sort-preserving mappings of relations and constants, i.e., all possible blends. During the tree generation, certain optimality principles are applied to prune the space (see below). Finally, after the tree is generated, the powerset of possible axioms wrt. the leafs is enumerated.

Cross-Space Mapping and Generic Space. The authors do not explicitly account for a cross-space mapping. However, the edges between relations and constants from different theories can be understood as a weak form of a cross-space mapping. Alloy uses these edges (and hence the potential cross-space mapping) to determine a generic space.

Emergent Structure. The emergent structure is generated by blending axioms of the input spaces. However, there is no clear distinction between the composition, completion and elaboration of the blend.

3.1.2 Optimality Principles

The optimality principles by Fauconnier and Turner are considered in the articles on Alloy and Griot [33, 29, 30], but the authors argue, that implementing these principles is difficult because they are computationally not efficient. Therefore, the authors propose other *structural* optimality principles which have better computational properties. These principles are based on degrees of commutativity, axiom preservation, and type casting, and they serve the purpose to prune the space of possible blends and as a means to produce only “good” blends.

3.1.3 Structural Blending

Another kind of blending that Goguen and Harrell [29, 30] use to generate narrative structure is *structural blending*. The approach is based on the work on narrative structure by Labove [44], and the authors use the following EBNF proposed in [44] as a basis for structure generation:

$$\begin{aligned} \langle \text{Narr} \rangle &::= \langle \text{Open} \rangle (\langle \text{Cls} \rangle \langle \text{Eval} \rangle^*)^+ [\langle \text{Coda} \rangle] \\ \langle \text{Open} \rangle &::= ((\langle \text{Abs} \rangle + \langle \text{Ornt} \rangle) \langle \text{Eval} \rangle^*)^* \end{aligned}$$

$\langle \text{Open} \rangle$ denotes the opening of a narrative, $\langle \text{Cls} \rangle$ stands for narrative clauses, $\langle \text{Eval} \rangle$ stands for evaluative clauses and $\langle \text{Coda} \rangle$ denotes an ending. More details can be found in [44].

For structural blending, a narrative space is generated by expanding the “*”s and by applying certain other rules which are not further discussed. The authors also mention that structural optimality principles¹ are used to assess the generated blends, but a formal definition of these

¹These *structural* optimality principles are not to be confused with the *conceptual* principles defined by Fauconnier and Turner [21].

principles is missing. The authors say that some elements of the generated narrative space are combined to generate a new structure, but they give no further elaborate definition of how this is implemented and what structural blending is. The most descriptive sentence to describe structural blending in [30, p.7-8] is as follows: *“Thus, to use blending as a basis for multimedia narrative, we must generalise conceptual spaces to take account of structure, which requires constructors and axioms; it also helps to have a hierarchical type system. Hence we distinguish conceptual blending from structural blending [...], where the former is blending of conceptual spaces and the latter is blending that in general involves non-trivial constructors.”*

3.1.4 Strengths and Weaknesses of Alloy and Griot

Alloy and the Griot system are based on the solid theory of algebraic semiotics by Goguen [24], it accounts for the work on conceptual integration by Fauconnier and Turner [22, 21] and it is based on the work on narrative theory by Labove [44]. The results that Griot produces are very impressive. For example, the system is capable of generating complex poems like the following:

her tale began when she was infected with smugnessloveitis.
she began her days looking in the mirror
at her own itchy entitled face.
her failure was ignoring her tormented angel nature.
life was an astounding miracle.
nordic-beauty death-figure vapor steamed from her pores
when she rode her bicycle. that was nothing lovely.
when 21 she was a homely woman. she decided to persevere;
in the rain, she fears only epidermis imperialists.
she believes that evil pride devours and alternates with pride of hope.
it was no laughing matter.
she snuggles in angel skin sheets and sleeps.
inside she was resolved to never find
a smug or paranoid love. [29]

A weakness of the Alloy approach is the input language BOBJ. Though BOBJ is already comparably expressive from an algebraic-logic point of view, it is unclear how informal information such as image schemas and emotion can be represented in BOBJ. Another weakness is that the generated search tree is of exponential size wrt. the number of axioms of the input theories, because during the generation of the blendoid tree, the powerset of axioms is computed.

3.2 Sapper

Sapper was originally developed by Veale and Keane [73] as a computational model of metaphor and analogy. It computes a mapping between two separate domains —understood as graphs of concepts— that respects the relational structure between the concepts in each domain. [Veale and O’Donoghue](#) argue in a later publication [74] that Sapper can also be seen as a computational model for conceptual blending, because the pairs of concepts that constitute its output can be

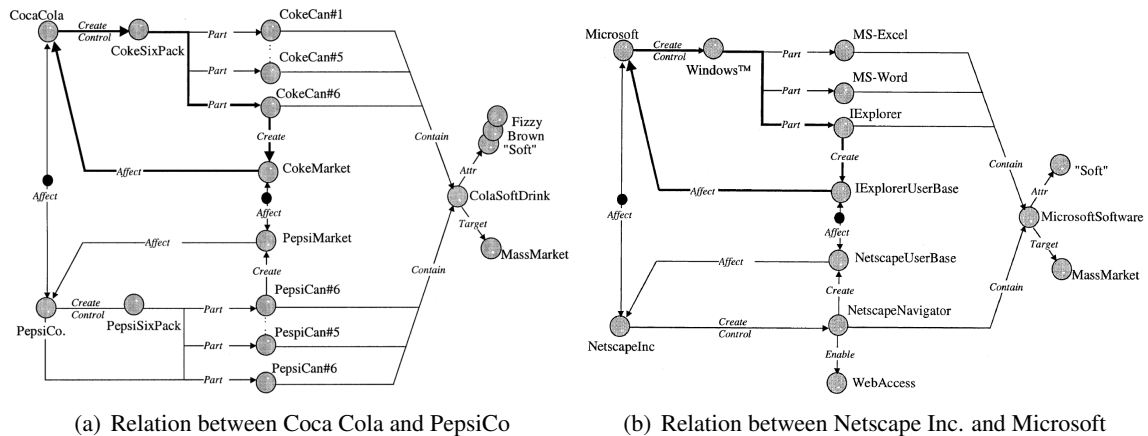


Figure 3: Two networks showing the economic relation between rival companies [74]

manipulated as atomic units, as blended concepts; and they also claim that the model captures most of the optimality principles of blending put forward by Fauconnier and Turner.

3.2.1 Constitutive Elements of Blending

Input Spaces. Strictly speaking, Sapper does not work with a priori given input spaces. It is the structure mapping algorithm itself which, given two domains to be mapped, determines the set of concepts and relations between these concepts that constitute the spaces that are blended. It does so by searching within its semantic memory for the largest substructures (bounded by a previously fixed size) at the root concepts of these domains that are isomorphic as with respect to their relational structure, and whose concepts are metaphorically related.

Semantic memory is represented as a semantic network [64], a graph whose nodes represent concepts and whose edges represent binary relations between concepts. As an example, consider Figure 3, that shows two semantic networks representing the economical relation between rival companies.

Cross-Space Mapping and Generic Space. Before the structure mapping algorithm is applied, Sapper attempts to identify potential metaphorical relations between concepts. It does so by identifying so called *bridgeable* concepts., i.e., two concepts that are linked by some semantic relation

- to one or more common concepts —a process called *triangulation*— or
- to two other concepts that are already related by a bridge relation —a process called *squaring*.

Bridges link concepts that share some set of semantic features with each other and that might be put into correspondence when blended. It is this set of shared features that plays the role of the generic space for blending. As an example, consider the concept *MassMarket* in Figure 3, which is part of the *Target* relation in both input networks.

Blend. The actual correspondence or blend between concepts is computed by applying a spreading activation algorithm (up to a fix horizon H) to the root nodes of the domains to be blended until a wave of activation from the target node T meets a wave from the source node S at a bridge between T' and S' . This way Sapper locates pairs of paths, of which one path is rooted at T and the other at S , that are structurally isomorphic (of equal length and constituted by the same sequence of semantic relations) and that terminate at concepts that Sapper considers semantically bridgeable. As an example, consider the following paths from Figure 3, which suggest to bridge the concepts *Microsoft* and *CocaCola*.

- *Mircosoft* $\xrightarrow{\text{Create}}$ *Windows* $\xrightarrow{\text{Part}}$ *MS-Excel* $\xrightarrow{\text{Contain}}$ *MicrosoftSoftware* $\xrightarrow{\text{Target}}$ *MassMarket*
- *CocaCola* $\xrightarrow{\text{Create}}$ *CokeSixPack* $\xrightarrow{\text{Part}}$ *CokeCan\#1* $\xrightarrow{\text{Contain}}$ *ColaSoftDrink* $\xrightarrow{\text{Target}}$ *MassMarket*

However, since many of these bridges will not coherently interact with others, only a subset of the possible bridges between input concepts will be chosen to form the blend. The final blend consists of the coherent combination of the one-to-one correspondence that can be established between intermediate nodes in isomorphic paths. This set of bridges can be seen as the output of Sapper, which can then serve as input to other inferential processes.

Emergent Structure. The correspondences found by the spreading activation algorithm that constitute the blend outlive the computation of the blend and are added as links into the semantic network of semantic memory. Consequently they can spread activation in subsequent applications of the spreading activation algorithm. This allow for carrying out inference and supports the computation of emergent properties of the blended space.

Conceptual Integration Network. The conceptual integration network handled by Sapper consists of just two semantic networks of separate domains which are mapped by applying spreading activation on the domain's root concepts, to identifying isomorphic substructures that are grounded on literal similarity, which constitute the generic space. [Veale and O'Donoghue](#) further claim for the inclusion into the network of a so called "constructor space", not part of Fauconnier and Turner's original model, but required for computational reasons, as different pragmatic contexts appear to call for different ways of constructing the generic and blend spaces.

The constructor space is a toolbox in which a computationalist may place structures and schemas (though, not processes) responsible for the construction of the generic space from input spaces, and then of the blended space. In Sapper, this constructor space would be constituted by the triangulation, squaring and so-called slippage rules (see Topology principle).

3.2.2 Optimality Principles

[Veale and O'Donoghue](#) claim that Sapper captures most of the optimality principles of blending put forward by Fauconnier and Turner. Optimality principles serve to rank and filter the correspondences that comprise the mapping computed by Sapper.

Topology. Sapper’s focus on graph isomorphism is the basis of establishing cross-space correspondences trivially ensures the topology principle. Since bridges are derived from isomorphic relationships between the semantic structure of the input spaces, they exhibit the same topological relationships to each other as do the original input concepts.

Sapper allows the relaxation of isomorphisms to accommodate to contexts that are more tolerant to so-called *structural slippage*, when there is not a strict one-to-one correspondence between relations. The constructor space uses certain *slippage rules*, to determine both the generic content of the blend and how this content structurally reconciles the input spaces.

Integration and Web. Integration is achieved by manipulating bridges as a single entity, and can be made explicit by labelling the bridge with a new concept name. Looking at bridges as new integrated concepts, the squaring rule is actually a triangulation rule where two concepts share a common bridge instead of a common concept.

Bridges allow Sapper to access the component concept allowing one to reconstruct the inputs as needed. And since bridges monotonically add to the structure of semantic memory, the concepts linked by a bridge still maintain their connectivity to the rest of semantic memory. Hence, Sapper adheres to the web principle.

Unpacking. Given a particular bridge, the squaring rule can be used to find neighboring bridges in the same conceptual space with which it is structurally coherent. This allows a blended concept to retrace the associations found by spreading activation back to the original input concepts, i.e., to “unpack” and reveal the additional conceptual structures that contribute to the bridge.

Relevance. The squaring and slippage rules on which Sapper grounds the formation of semantic bridges underpin the Relevance principle, by ensuring that each bridge in the blended space is the result of a structural or semantic consequence of another bridge, ultimately bottoming out at pairs of concepts that were identified to be bridgeable on ground of sharing the same set of features.

3.2.3 Strengths and Weaknesses

A computational model like Sapper demonstrates how key properties and principles of integration theory effectively emerge from the interaction of some well-tested notions from cognitive science and artificial intelligence (e.g., spreading activation, semantic networks, graph isomorphism). It also demonstrates the inherent computational soundness of conceptual integration theory as a paradigm for research in cognitive science.

The authors provide strong links to the work by Fauconnier and Turner, in particular with respect to the optimality principles. Veale and O’Donoghue [74] claim that “ [...] *optimality principles are entirely compatible with a number of computational ideas that make blending a computationally tractable process.*” This however contradicts Harrell [33], who state that Fauconnier and Turner’s optimality principles are computationally inefficient. With respect to this claim, it would be interesting to learn about the computational scalability of the Sapper system, e.g., in terms of an empirical performance analysis for sample data.

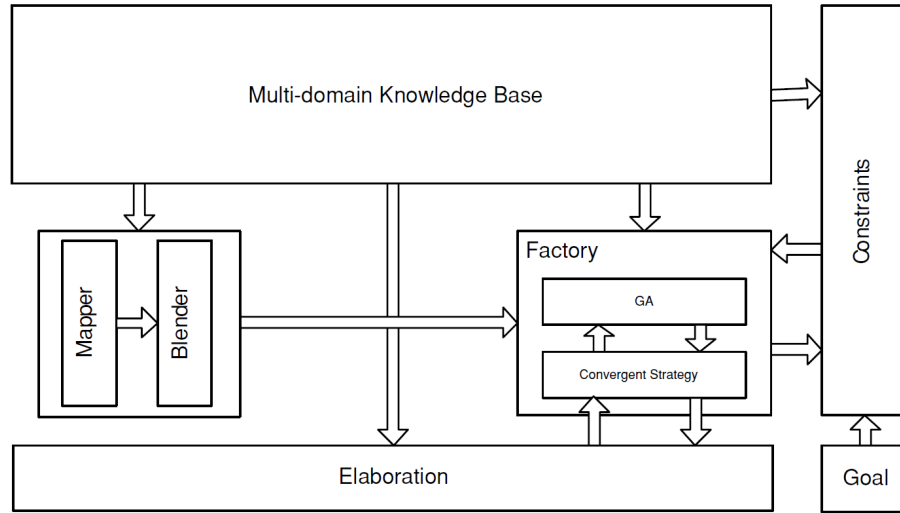


Figure 4: The Divago architecture

An explanation for the contradiction with Harrell is, that Veale and O’Donoghue use relatively simple graphs as input which have the expressiveness of existential-conjunctive logic with only binary predicates. In contrast, Harrell uses the more expressive input language BOBJ. In addition, Veale and O’Donoghue use (among other algorithms) a sophisticated structure mapping engine (SME) which is optimised so that it can operate with a polynomial time ceiling (see [73] for details).

3.3 Divago

Divago, by Pereira [59, 60], is probably the first complete implementation of conceptual blending that aims at capturing as much elements as possible of Fauconnier and Turner’s cognitive model, and that attempts to provide a concrete computational account for them. Pereira draws the terminology and definitions for his formal and computational model from Wiggins’s formalisation of creative systems [75]. The implementation of Divago is realised in Prolog.

For details pertaining to the Divago architecture consider Figure 4. The knowledge base contains different micro-theories and their instantiations. Of these, two are selected for the blending. The mapper then generated the alignment, i.e., the generic space between the inputs, and passes on the blender which generates the “blendoid”, i.e., a projection that defines the space of possible blends. The factory is used to select the best blends among the blendoid by means of a genetic algorithm. This selection is interleaved with (i) enriching the blendoid with background knowledge in the elaboration module and (ii) incorporating the constraint module where the optimality principles are applied.

| | |
|---|--|
| <code>isa(house, physical structure)</code> | <code>isa(boat, physical structure)</code> |
| <code>isa(physical structure, physical entity)</code> | <code>isa(sailing boat, boat)</code> |
| <code>purpose(roof, protection)</code> | <code>shape(hatch, circle)</code> |
| <code>isa(protection, task)</code> | <code>have(sailing boat, sail)</code> |
| <code>live in(human, house)</code> | <code>have(sailing boat, hatch)</code> |
| <code>color(night, black)</code> | <code>have(sailing boat, mast)</code> |
| <code>have(house, door)</code> | <code>purpose(sail, movement)</code> |
| <code>have(house, roof)</code> | <code>purpose(hatch, observation)</code> |
| <code>have(house, body)</code> | <code>purpose(mast, support)</code> |
| <code>purpose(body, container)</code> | <code>purpose(vessel, container)</code> |
| <code>purpose(door, entrance)</code> | <code>property(sailing boat, slow)</code> |
| <code>have many(skyscraper, house)</code> | <code>property(hatch, tiny)</code> |
| <code>have many(house, room)</code> | <code>use(human, sailing boat)</code> |
| <code>:</code> | <code>sail(human, sailing boat)</code> |
| <code>:</code> | <code>:</code> |
| (a) Excerpt of the <i>house</i> concept map | (b) Excerpt of the <i>boat</i> concept map |

Figure 5: Input spaces for Divago in form of Prolog facts

3.3.1 Constitutive Elements of Blending

Input Spaces. Blending in Divago is done on a pair of concepts, taken from the multi-domain² knowledge base of the system.

Pereira adopts Murphy and Medin’s “theory view” of what a concept is [56]. Hence, in Divago concepts are micro-theories that describe facts about them (or about related concepts), and which include causal connections between them. In particular, a concept micro-theory $C = (CM, R, F, IC)$ consists of:

- a concept map CM for the factual part — that can be represented as a directed graph with nodes standing for concepts (they will be called ‘elements’ to distinguish them from the notion of ‘concept’, which involves the entire concept map) and edges standing for relations;³
- rules R , frames F , and integrity constraints IC for the inferential part — that play an important role in the blending process and the further elaboration of the blend, as we shall explain below.

Concept micro-theories in Divago are essentially logic programs, and are written as Prolog clauses. Instances can also be provided. As an example consider the micro-theories for *house* and *boat* in Figure 5.

²A domain for Pereira is a set of concepts such that all of them relate to a unique, underlying concept.

³Pereira clarifies the notions of ‘domain’, ‘concept’ and ‘element’ he uses in his work as follows: “In order to avoid confusion, we adopt the convention that each of these nodes of a concept map will be named element, instead of concept. Thus, a Concept will be made up of the concept maps, rules, frames, etc. [...] following the micro-theory view.” (p. 104). A domain is a concept map with no central concept. Although he is not entirely consistent with this view, we will try to follow his intended terminology.

Cross-Space Mapping. Mappings are initially formalised as a function $\phi : \mathcal{U} \times \mathcal{U} \rightarrow \mathcal{M}$, where \mathcal{M} is the power set of all possible pairs of substructures from concepts of \mathcal{U} . In particular, in Divago, mapping is implemented to yield as output a set of pairs of elements (e_1, e_2) where e_1 is an element of the concept map of one input concept, and e_2 is an element of the concept map of the other input concept. Hence, a mapping between concepts C_1 and C_2 , is just a binary relation $m \subseteq \mathcal{AE}_1 \times \mathcal{AE}_2$, where \mathcal{AE}_i is the subset of element symbols occurring at nodes of the concept map CM_i of concept C_i .

Like Sapper, the mapper module of Divago uses spreading activation to search for the largest bijection m such that, for all $Y_1, Z_1 \in \mathcal{AE}_1$ and $Y_2, Z_2 \in \mathcal{AE}_2$, whenever $(Y_1, Y_2) \in m$ and $(Z_1, Z_2) \in m$, then $X(Y_1, Z_1) \in CM_1$ if, and only if, $X(Y_2, Z_2) \in CM_2$. That is, it looks for the largest isomorphic pair of subgraphs (contained in the concept maps), where two graphs are considered isomorphic when they have the same edge structure (relations), independently of the symbols occurring at nodes (elements).

Generic Space. In Divago there is no generic space as understood by Fauconnier and Turner. Instead, Divago always includes a generic domain that participates in all blend computations. This domain contains:

- all knowledge that is applicable to all concepts and to the process of concept invention;
- the Generalised Upper Model [4], a general task and domain independent linguistic ontology, intended for organising information for expression in natural language, and from which all labels for relations in Divago’s concept maps are taken from;
- an IS-A ontology used by the mapper for finding correspondences between elements of different concepts;
- various frames that govern the creation of blends;
- integrity constraints and rules.

Blend. Blending is formalised by means of a ‘transfer operation’ $\omega : \mathcal{U} \times \mathcal{U} \rightarrow \mathcal{U}$, that is dependent on the mapping between inputs: $\omega(x, y) = \emptyset$ if $\phi(x, y) = 0$. No further constraints on ω are given, and Pereira claims that “there is no way to specify this operation in more detail, since there are many different accounts for how concepts are combined together.” However, this cannot be entirely true, because in retrospect one can tell whether a newly created concept is the combination of two previously existing concepts or not; hence, a minimal characterisation for ω should be possible.

According to Pereira’s formalisation, there is directionality in blending, and unlike the mapping function ϕ , the transfer operation ω is not necessarily symmetric. Furthermore, ω is understood as a non-deterministic algorithm, rather than a mathematical function. Thus, the set $\Omega = \{k \mid k = \omega(x, y), x, y \in \mathcal{U}\}$ of all possible blends of two concepts is called the *bisociation set*.

In Divago, blending is implemented by computing the *blendoid*, which merges all possible blends into one single concept map. The blendoid is generated by means of a *blending projection* function $\gamma : \mathcal{A} \rightarrow \mathcal{A}_B$ that maps the atomic symbols (constants and variables) \mathcal{A} of the input concept

maps into the blended concept map, whose atomic symbols include also compound symbols: $\mathcal{A}_B = \mathcal{A} \cup \{x|y \mid x, y \in \mathcal{A} \wedge x \neq y\}$. Given a mapping m between concept maps, the function γ has to satisfy that, for all $x, y \in \mathcal{A}$, $\gamma(x) = \emptyset$, $\gamma(x) = x$, or, whenever $(x, y) \in m$, $\gamma(x) = y$ or $\gamma(x) = x|y$. But even if two concepts x and y are mapped according to m , nothing prevents that $\gamma(x) \neq \gamma(y)$. Rules, frames, integrity constraints, and instances are all included into the blend, after translating the atomic symbols according to the blending projection γ .

Out of the blendoid, the best blend is selected applying a genetic algorithm that searches for the best blending projections. The genotype of the individuals of the population over which the genetic algorithm is run are sequences of projected atomic symbols for a particular blending projection γ . The population evolves via crossover, mutation and asexual reproduction (direct copy of the genotype) and evaluated by a fitness function applied on the actual blend that is generated given a particular blending projection γ , the phenotype. The fitness function is a weighted sum based on the Optimality Principles.

Emergent Structure. The blends in the *bisociation set* Ω are considered to be in a pre-inventive state, and Divago handles the emergent structure that arises in the blend by means of an elaboration function $\theta : \mathcal{U} \times \mathcal{U} \rightarrow \mathcal{U}$. This function can be either:

- rule-based, when a rule or set of rules from the blend or the generic domain is applied (e.g., heuristics, causal rules, or other production rules)
- internal-logic-based, when new structure is added by reasoning within the concept micro-theory (e.g. by deduction, induction, or abduction)
- cross-concept-based, when new structure is added or removed by comparison to other concepts in the knowledge base (e.g., by means of analogy)

In Divago, this elaboration of the blend is interleaved with the generic algorithm so as to allow less fit blends to gain value via the application of rules, frames and inference rules.

Conceptual Integration Network. The conceptual integration network handled by Divago consists of just two concept micro-theories which are mapped by applying spreading activation on the concept's concept maps and identifying isomorphic substructures. The role of the generic space is covered by the generic domain, which further provides also the knowledge, rules and frames that trigger the elaboration of the blends.

3.3.2 Optimality Principles

Divago's architecture includes a dedicated module that implements the optimality principles. Given a blend, this module computes a measure for each principle which aims at capturing the degree of adherence to the principle. These measures are added up in a weighted sum to yield a preference value of the blend. This value is then taken as the fitness value of the genetic algorithm.

Topology. The value of adherence to the Topology principle is computed by taking the ratio of topologically correct relations in the blend with respect to the size of the concept map of the blend. Depending on the objectives of creating the blend one may want it to preserve more the structure of the input blends or to allow to be more flexible. This can be tuned with the weight associated to this value.

Pattern Completion. Pattern Completion is implemented by using frames that are partially satisfied by the blend and asserting the truth of its grounded condition. For this a *completion evidence* is computed for each frame with respect to the blend, by looking at the ratio of satisfied conditions with respect to all conditions of the frame, weighted by a penalty factor determined by the number of violated integrity constraints. If this evidence is above a certain threshold, the frame is used for completion of the blend. This completion is then taken up by the Elaboration module to infer new consequences from the blend.

The Pattern Completion measure of a blend is computed with respect to a set of frames F , by taking the union of all its conditions and computing the completion evidence of this union.

Integration. Again, Divago takes frame as the notion behind integration, because a frame organises a concept into an understandable whole. Divago interprets integration as the coverage of as much of the blend by as little frames as possible, where coverage of a frame means that all conditions of the frame are satisfied by the concept map of the blend. For each frame a integration value I_f is computed, by computing how much of the blend is covered by the frame and how little of the integrity constraints are violated. The values for each frame are then added up taking into account how much of the blend remain uncovered, how much overlap there is between frames and how many frames are involved in the coverage.

Maximisation of Vital Relations. Divago interprets vital relations as salient relations, i.e., there is not specific treatment of their role in compression —when inter-space relations between concepts of different inputs intra-space relation in the blend. Furthermore, only maximisation is considered. The measure computed is just the ratio of vital relations in the blend with respect to all possible vital relations in the blendoid.

Web. In Divago, the Web principle is derived from the Topology and Unpacking principle, and its measure is just a weighted sum of these two principles.

Unpacking. Divago implements the Unpacking Principle using the notion of a *defining frame* of an element x in the blend, which is a frame whose conditions relate the element of the input space that is projected onto x to its neighboring elements. The structure of a defining frame that is preserved in the blend points to the original concept. Consequently, they provide a measure for Unpacking, which Divago determines as the normalised sum of ratios computed for each element x , namely the ration of defining-frame conditions satisfied by the blend for x with respect to the number of elements to which x is connected.

Relevance. Finally, Relevance or “Good Reason” of the blend is computed with respect to a given set of goal frames. Maximum relevance is achieved if the blend satisfies all goal frames. If not all goal frames are satisfied, Divago computes the Pattern Completion value of unsatisfied blends and takes these also into account. The Relevance value is taken as the degree of usefulness of the creative act.

3.3.3 Strengths and Weaknesses

Divago’s main strength is that it addresses most of the elements and principles of conceptual blending as put forth by Fauconnier and Turner in their cognitive model, and that it offers a concrete implementation of these elements and principles. This however is also a weakness, because the formal model it gives for blending is determined and tailored towards the concrete implementation. Consequently, the concept representation language of Divago is restricted to the expressiveness of the Prolog-based rules.

3.4 Blending by Heuristic-Driven Theory Projection (HDTP)

Guhe et al. [31] present an approach to use Heuristic-Driven Theory Projection (HDTP) [69] for blending and concept invention. HDTP is originally a framework for analogical reasoning, using a many-sorted first order language to represent conceptual spaces. The authors state that “*HDTP extends [the] classical form of anti-unification of terms to formulae and logical theories by iteratively picking pairs of formulae to be generalised from the domains. This process is driven by heuristics.*” In particular, HDTP uses second-order anti-unification, which is restricted in way that renders the process decidable.

In HDTP-based analogical reasoning, knowledge is mapped and transferred from a usually well-known source domain S to a target domain T . This happens in two phases. In the mapping phase, source and target are compared to find commonalities. In the transfer phase, unmatched knowledge in source is mapped to the target to establish new hypotheses.

The authors summarise how they turn the analogical reasoning framework into a conceptual blending framework as follows [31] (see Figure 6): “*The main point, where conceptual blending differs from analogical reasoning is the second phase - the analogical transfer of knowledge from S to T . To turn HDTP into a blending framework, we replace this transfer by a process that results in the creation of new domain, as follows:*

1. *Compute a common generalisation of the domains S and T , thus setting up relationships between the source and target domains.*
2. *Create the conceptual blend B by merging knowledge from S and T based on this mapping: in the ideal case, B respects the shared features of S and T (those with common generalisations), and inherits independently the other features of S and T . ”*

The constitutive elements of Guhe et al.’s approach are similar to those proposed by Fauconnier and Turner, though some important elements such as optimality principles or the distinction between composition, completion and elaboration are not explicitly accounted for.

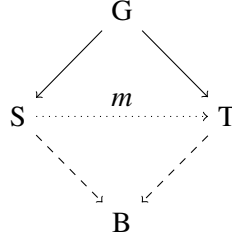


Figure 6: Extending the analogical reasoning framework to conceptual blending. m is an analogy mapping from a well-known source domain S to a new target domain T .

3.4.1 Constitutive Elements of Blending

Input Spaces. As conceptual spaces, the authors use many-sorted first-order theories representing the image schemas proposed by Lakoff and Núñez [48]. In particular, the authors use the following four schemas: OBJECT COLLECTION (OColl), OBJECT CONSTRUCTION (OConst), MEASURING STICK (MS) and MOTION ALONG A PATH (MAP). Guhe et al. argue with Lakoff and Núñez, that these metaphors are the main building blocks to represent mathematical structures, e.g., rational numbers. As an example they consider the specification of MS and MAP input spaces in Figure 7.

Cross-Space Mapping and Generic Space. Guhe et al. [31] state that “*HDTP [...] provides an explicit generalisation of two domains as a by-product of establishing an analogy. Such a generalisation can be a base for concept creation by abstraction.*” Hence, the established analogy is a cross-space mapping that is determined by the restricted second-order anti-unification method presented in [69]. The analogy directly determines the generic space.

Blend. The blending of two theories happens in three steps. First, core blend laws are applied, second, preferred conjectures are added, and third, extra conjectures are added.

1. **Core blend laws.** The system unites two input signatures Σ_1 and Σ_2 to generate a new signature as follows:
 - (a) Add a predicate symbol for each sort that occurs in one of the input signatures, and assign entities e to sorts $sort$ by adding the fact $sort(e)$ to the theory.
 - (b) For formulas $\forall x : \text{Sort}(\phi)$ in an input theory the authors add $\forall x : sort(x) \rightarrow \phi$ to the blend. Similarly, for $\exists x : \text{Sort}(\phi)$ in an input theory the authors add $\exists x : sort(x) \wedge \phi$.
 - (c) For function symbols $f : \text{Sort}_1 \times \dots \times \text{Sort}_n \rightarrow \text{Sort}$, add $\forall x_1, \dots, x_n : ((sort_1(x_1) \wedge \dots \wedge sort_n(x_n)) \rightarrow sort(f(x_1, \dots, x_n)))$ to the blend.
2. **Preferred conjectures.** The system generates and adds laws that concern equality of analogous entities, functions and relations.
3. **Extra conjectures.** The system adds additional laws from the input spaces, if they use only symbols covered by the analogy. It also tries to add more symbols of the input spaces to the blend, if these do not lead to inconsistencies.

Sorts: *seg*

Entities:

unitSeg : *seg*

Predicates:

longer, shorter : *seg* \times *seg*
extend, chop : *seg* \times *seg* \times *seg*

Laws:

μ_1 : $\forall S_1, S_2 : \text{shorter}(S_1, S_2) \leftrightarrow \text{longer}(S_2, S_1)$
 μ_2 : $\forall S_1, S_2, S_3 : \text{extend}(S_1, S_2, S_3) \leftrightarrow \text{chop}(S_3, S_2, S_1)$
 μ_{3a} : $\forall S_1, S_2 : \text{shorter}(S_1, S_2) \vee (S_1 = S_2) \vee \text{longer}(S_1, S_2)$
 μ_{3b} : $\forall S_1, S_2 : \text{shorter}(S_1, S_2) \rightarrow \neg \text{shorter}(S_2, S_1)$
 μ_4 : $\forall S : \neg \text{longer}(\text{unitSeg}, S)$
 μ_5 : $\forall S_1, S_2, S_3 : \text{extend}(S_1, S_2, S_3) \rightarrow \text{longer}(S_3, S_2) \wedge \text{longer}(S_3, S_1)$

(a) The MEASURING STICK (MS) input schema

Sorts: *point*

Entities:

origin : *point*

Predicates:

farther, closer : *point* \times *point*
moveAway, moveCloser : *point* \times *point* \times *point*

Laws:

π_1 : $\forall P_1, P_2 : \text{closer}(P_1, P_2) \leftrightarrow \text{farther}(P_2, P_1)$
 π_2 : $\forall P_1, P_2, P_3 : \text{moveAway}(P_1, P_2, P_3) \leftrightarrow \text{moveCloser}(P_3, P_2, P_1)$
 π_{3a} : $\forall P_1, P_2 : \text{closer}(P_1, P_2) \vee (P_1 = P_2) \vee \text{farther}(P_1, P_2)$
 π_{3b} : $\forall P_1, P_2 : \text{closer}(P_1, P_2) \rightarrow \neg \text{closer}(P_2, P_1)$
 π_4 : $\forall P : \neg \text{farther}(\text{origin}, P)$
 π_{5a} : $\forall P_1, P_2, P_3 : (\text{moveAway}(P_1, P_2, P_3) \wedge P_2 \neq \text{origin}) \rightarrow \text{farther}(P_3, P_2) \wedge \text{farther}(P_3, P_1)$
 π_{5b} : $\forall P_1, P_3 : \text{moveAway}(P_1, \text{origin}, P_3) \rightarrow P_1 = P_3$

(b) The MOTION ALONG A PATH (MAP) input schema

Figure 7: Input schemas for HDTP-based blending (see [31])

Figure 8 shows an example where the MS and MAP input spaces are blended to generate the DISCRETE MOTION ALONG PATH (DMAP) schema. The mapping between the laws $\mu_1 - \mu_4$ and $\pi_1 - \pi_4$ is straight-forward, and used to generate the generalisation space: *shorter* is analogous to *closer*, *longer* is analogous to *farther*, *Add* is analogous to *extend*, etc.

3.4.2 Optimality Principles

The authors do not directly account for the optimality principles by Fauconnier and Turner. However, the preferred and extra conjectures can be seen as alternative optimality principles, similar to the structural optimality principles proposed by Goguen and Harrell [29], that guide the algorithm to select useful blends among the huge space of possible blends.

3.4.3 Strengths and Weaknesses

The HDTP-based blending approach has a comparably expressive input language. Furthermore, the method is, in comparison with other approaches, well described in detail. However, it would

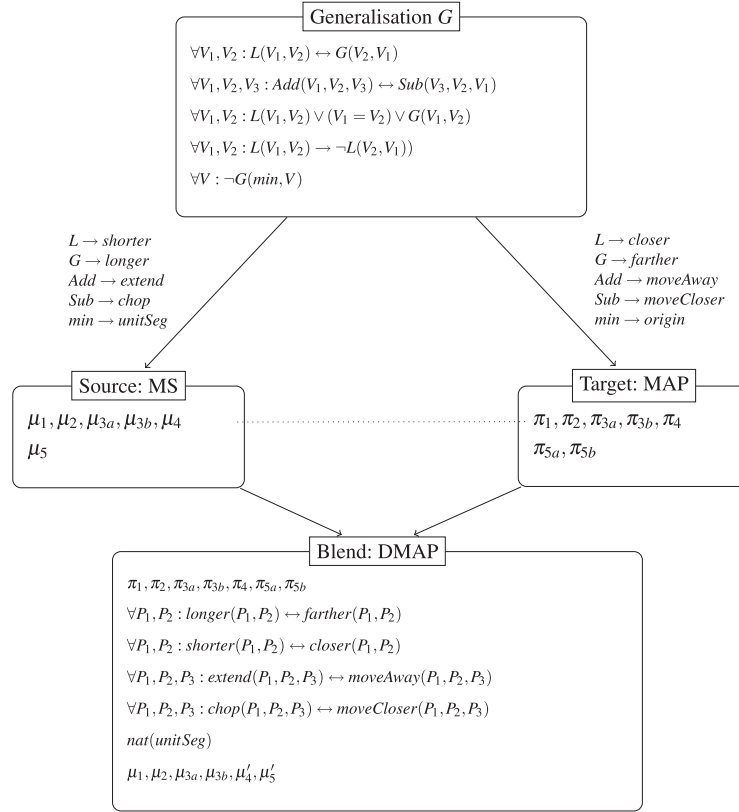


Figure 8: Blending the MS and the MAP input space (see [31])

be interesting to investigate the scalability of the approach for blending bigger ontologies. The approach does not consider optimality principles and partial morphisms.

3.5 A Neurological Account on Combinatorial Creativity

The combinatorial kind of creativity [6] that we are interested in has been investigated from a neurological perspective by Thagard and Stewart [72]. The major motivation of their approach is to explain and to model the *Aha!* or *Eureka!* effect that occurs when humans make serendipitous discoveries by means of creative thinking. The authors build their work on findings from neuroscience and approaches to realise human thinking with neural networks [71, 72, 15, 16, 62]. The key idea is to represent mental concepts as activity patterns of vectors of neurons and to perform a convolution operation to combine these patterns.

Activity patterns of neurons are mathematically represented as vectors of numbers that represent the firing rate of neurons. For example, if the maximal firing rate of neurons is 200Hz, a firing rate of 100Hz is represented as the number 0.5. According to Thagard and Stewart [72], a mental concept can then be represented as a huge but finite vector of such numbers. This representation allows one to elegantly combine two mental concepts by the discrete mathematical *convolution* operation on vectors.

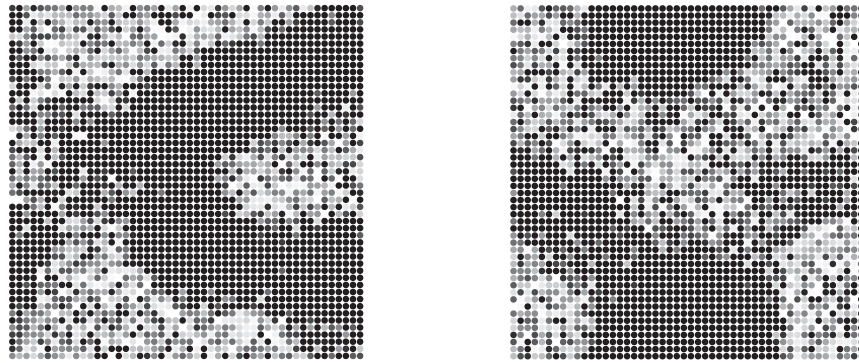


Figure 9: Patterns of neuronal activity as input spaces in Thagard and Stewart [72]

3.5.1 Constitutive Elements of Blending

Input Spaces. Input spaces are activity patterns of neurons, which in turn are represented by vectors of numbers that represent the firing rates of neurons. As an example, consider the two patterns in Figure 9, where brighter dots represent a higher firing rate and darker dots a lower firing rate. Note that this model of input spaces is biologically very realistic: Neurons do not store discrete bits like computers, but instead are highly stochastic devices which even may die. Towards this, a high number of neurons is used to store information redundantly and to make the system robust. This approach coincides with recent findings in neurology (see e.g. [71]).

The goal of the author's work is not only to combine mental spaces, but also to model the Aha! effect. Towards this, the authors claim that for the Aha! effect to occur, two more input spaces are required: a representation of appraisal for the convolved input spaces and a representation of the perception of the physiological reaction (e.g. heart beat) on the convolution. These are also modelled as mental representations in form of vectors, and convolved to a vector representing the overall *emotional reaction*. The emotional reaction is then again convolved with the convolution of the two original input spaces and may trigger the Aha! effect if the appraisal is high and the physiological reaction is remarkable (see Figure 10).

Generic Space and Cross-Space Mapping The convolution operation that the authors use does not require a generalisation space as in Fauconnier and Turner's theory. Furthermore, the cross-space mapping in the sense of Fauconnier and Turner is not explicitly accounted for.

Blend. The blend is generated by mathematical convolution of vectors. The underlying mathematical model is based on the so-called LIF model of neuronal activity (see e.g. [71]). It accounts for various details on the neuronal level, such as neuron voltage, input current, membrane time, direction vector of neuron patterns, and synaptic connection weights. The mathematical convolution

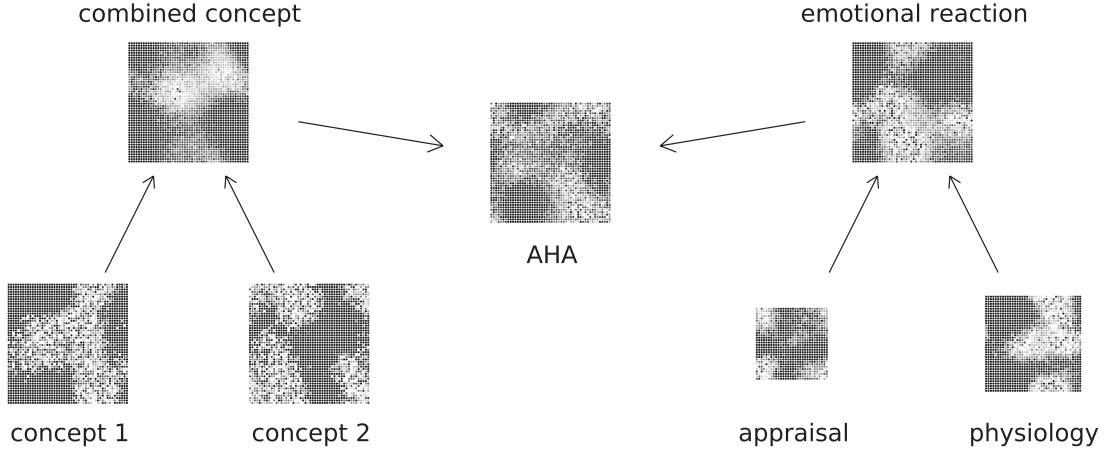


Figure 10: Thagard and Stewart’s convolution model of the *Aha!* effect [72]

“*” of two input vectors \bar{x} and \bar{y} is defined by Equation (3.1) as follows:

$$\bar{z} = \bar{x} * \bar{y} \quad \text{where} \quad z_i = \sum_{j=1}^N x_j y_{(i-j) \bmod N} \quad (3.1)$$

Here, N is the size of the vector and i, j iterate over the vectors’ constituents. The form of this equation allows the authors to apply the convolution theorem, such that Fourier transformation can be used to compute the convolution and hence the blend.

3.5.2 Optimality Principles

Thagard and Stewart [72] do not use Fauconnier and Turner’s optimality principles to distinguish reasonable blends within the huge space of possible blends. Instead, they combine the blend of two input spaces with another space representing emotional reaction (see Figure 10) to assess blends. However, the authors do not provide a detailed description how to model the emotional input spaces computationally.

Though optimality principles are not explicitly accounted for by Thagard and Stewart, there is one obvious analogy concerning the “Unpacking” principle. Thagard and Stewart [72] claim that the mathematical *deconvolution* operation on the input spaces allows one to reconstruct the input spaces from the combined space, at least to a certain extend.

3.5.3 Strengths and Weaknesses

Thagard and Stewart [72] state that the advantage of their approach before symbolic knowledge representations is, that all kinds of mental representations, such as emotional and sensory information, can be accounted for with the vector-based knowledge representation. On the one hand, this is true because the basic representation of input spaces makes the approach very general. For ex-

ample, the authors use the application domain of natural language text, where they use Plate [62]’s ‘holographic reduced representation’ approach to map natural language sentences to vectors.

On the other hand however, the vector-based knowledge representation also raises many questions. For example, it is very unclear how other high-level input spaces such as music pieces or mathematical theories can be mapped to their vector representation. Furthermore, their approach is limited in that it does not account for selective combination, i.e., merging only parts of conceptual spaces. In [Section 5](#) we present the Buddhist monk example which illustrates that selective combination is necessary to relax input spaces and to obtain certain conclusions within a blend.

4 A Formal Theory of Blending as Colimits and Amalgams

4.1 Blending as Colimits

Category theory, although initially designed to describe mathematical entities, has proven a successful cornerstone in many computer science applications; a trend which has attracted a lot of attention and researchers, and which has been nicely advocated in Goguen’s paper [28]. One of the most interesting advantages of categorical approaches to computational theories is precisely the fact of being independent of any particular implementation. For this very reason, it is very appealing to search for a (basic) categorical framework where the theory of conceptual blending can be developed. In particular, Goguen [24, 25, 26] already introduced a categorical approach to blending based on colimits.

In this section, besides diving into category theory, we revisit [Goguen’s](#) approach, and some subsequent categorical approximations to blending [42, 43].

4.1.1 Category Theory Preliminaries

The aim of this section is to familiarise the reader with the notion of pushout in a category, which is a particular case of the colimit construction. The colimit construction plays a crucial role in Goguen’s approach to blending. Towards this, consider the following quotation:

Given a species of structure, say widgets, then the result of interconnecting a system of widgets to form a super-widget corresponds to taking the *colimit* of the diagram of widgets in which the morphisms show how they are interconnected. [28, Section 6]

As an starting illustration of this informal intuition, which we present in a formalised setting in [Definition 4.4](#), the reader can consider the example given in [Figure 11](#), and borrowed from Tarlecki [70]. In this example the pushout, shown in the top right corner, is computed from the two morphisms starting in the bottom left corner. This pushout essentially replaces arrays of elements in one of the input specifications with arrays of strings. The notion of string is extracted from other input specification.

In this section, no attempt of being completely self-contained is made, so we suggest the reader to supplement the information here provided, whenever necessary, with any standard category theory textbook (e.g., [3, 61, 53, 49]) or short introductions to the subject (e.g., [14, Chapter 2] and [67, Chapter 3]).

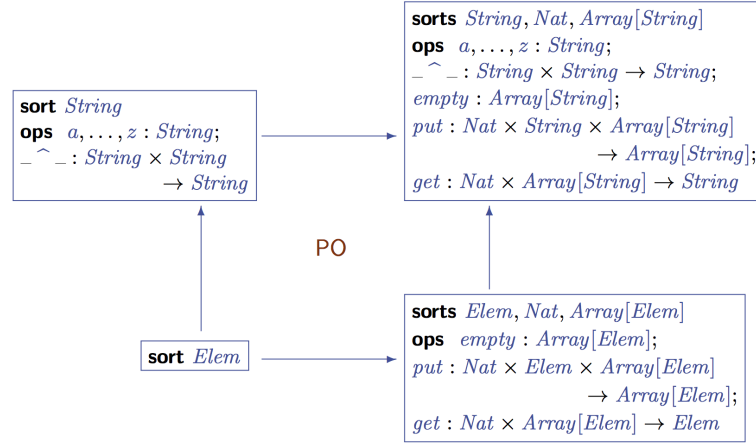


Figure 11: Example of a pushout (PO) of specifications

Definition 4.1 (Category). A *category* \mathbf{C} consists of the following items.

- A collection $\text{obj}(\mathbf{C})$ of *objects*.
- A collection $\text{hom}(\mathbf{C})$ of *morphisms* (sometimes also called homomorphisms, arrows or maps) satisfying that each morphism f has associated a *source* object denoted by $\text{src}(f)$, and a *target* object denoted by $\text{tg}(f)$. The expression $f: A \rightarrow B$ is used as a shorthand for claiming that f is a morphism with source A and target B . The collection of all such morphisms is denoted either by $\mathbf{C}(A, B)$ or $\text{hom}(A, B)$; but this last notation is only adequate when there is no ambiguity about the category.
- For every objects A, B, C , there is a binary associative operation called *composition* from $\text{hom}(A, B) \times \text{hom}(B, C)$ into $\text{hom}(A, C)$. Composition of two morphisms f, g is denoted by either writing

$$f; g \quad (\text{diagrammatic notation}) \quad \text{or} \quad g \circ f \quad (\text{functional notation})$$

to refer to the composition of morphisms $f: A \rightarrow B$ and $g: B \rightarrow C$. We remind that associativity refers to the equality

$$f; (g; h) = (f; g); h$$

and allows to write down finite sequences “ $f_1; f_2; \dots; f_n$ ” without worrying about parentheses.

- For every object A , there is an *identity* morphism id_A belonging to $\text{hom}(A, A)$ which is a neutral element of composition. This neutrality means that
 - $\text{id}_A; f$ is equal to f (for every morphism f with source A)

- $f; \text{id}_A$ is equal to f (for every morphism f with target A).

–

Concerning notation to be used later, we point out that $\text{hom}(A, -)$ will denote the collection of all morphisms with source A and $\text{hom}(-, A)$ will denote the one of morphisms with target A .

Example 4.2 (The categories **Set** and **Pfn**). Among the plethora of examples, there are two outstanding examples of well-known categories that play a key role in this report (see for example [9]).

- The category **Set** has sets as objects and (total) functions as morphisms (endowed with the usual composition of functions).⁴
- The category **Pfn** of sets has sets as objects and partial functions as morphisms (endowed also with the usual composition of functions).

Let us point out that if A and B are finite sets with, respectively, cardinal n and m , then $\text{Set}(A, B)$ has cardinal m^n while $\text{Pfn}(A, B)$ has cardinal $(m + 1)^n$. In case we have a partial function f , we will use the notation $\text{Dom}(f)$ to refer to its set theoretical domain and $\text{Im}(f)$ for its set theoretical image.

The categories **Set** and **Pfn** are well-known in the literature (see for example [9]). Indeed, **Set** can be considered as the paradigmatic category, in the sense that it has the best properties one can expect in a category; for example, it is bicomplete (see Definition 4.8), a topos [38], ... **Set** and **Pfn** underlay the more complex categories that we deal with in this report (e.g., categories of CASL theories with axiom-preserving theory morphisms, see the solution to the Buddhist Monk Riddle given in Section 5), in the sense that in all of these other considered categories it happens that morphisms are either functions or partial functions. Thus, understanding the categories **Set** and **Pfn** is a first approach to understand more involved categories. \square

Besides using the previously introduced notation $f: A \rightarrow B$ to refer to morphisms, it is common to use different kind of graphical arrows to emphasise whether the arrow satisfies some particular property. Thus, we will use

- $f: A \twoheadrightarrow B$ for *epimorphisms* (i.e., for every $h_1, h_2 \in \text{hom}(B, -)$, if $f; h_1 = f; h_2$ then $h_1 = h_2$).
- $f: A \rightarrowtail B$ for *monomorphisms*, (i.e., for every $h_1, h_2 \in \text{hom}(-, A)$, if $h_1; f = h_2; f$ then $h_1 = h_2$).
- $f: A \hookrightarrow B$ only for some very special monomorphisms, i.e., those that live in a category whose morphisms are (set-theoretical) functions preserving some structure and which correspond to inclusions.
- $f: A \xrightarrow{\sim} B$ for *isomorphisms* (i.e., there is some $h \in \text{hom}(B, A)$ such that $f; h = \text{id}_A$ and $h; f = \text{id}_B$).

⁴ It is worth noticing that, by definition of a category, the collections $\text{hom}(A_1, B_1)$ and $\text{hom}(A_2, B_2)$ must be disjoint unless both $A_1 = A_2$ and $B_1 = B_2$ hold. Thus, for technicality issues it is better to think that a morphism in **Set** is given by an ordered triple (A, f, B) where f is a function from A to B .

In the particular cases of **Set** and **Pfn** it is well-known that epimorphisms correspond to being exhaustive on the target object, monomorphisms to injectivity and isomorphisms to bijectivity. Thus, two sets are isomorphic iff they have the same cardinality.

Colimits (and also limits) in a category **C** are introduced via diagrams. A *diagram* \mathcal{D} is a functor from a category **J** to the category **C**, and in such a case it is said that \mathcal{D} is *J-shaped*. In other words, a *diagram* \mathcal{D} in **C** consists of

- a directed graph (where edges are objects-morphisms in **J**),⁵
- a family (indexed by the nodes of the graph) of objects in **C**, i.e., every node X of the graph is associated with an object in **C**,
- a family (indexed by the edges of the graph) of morphisms in **C** satisfying that: for an edge between nodes X and Y , the associated morphism has the object associated with X as source, and the object associated with Y as target.

We are mostly interested in the case of *finite diagrams*, i.e., when **J** has a finite number of objects and morphisms. In most such examples, instead of defining the category **J** in words, we will simply draw a directed graph. We refer the reader to [Figure 12](#), for an illustration of diagrams for some particular basic graphs. It is worth saying that the notion of *span* introduced there, will be crucial. Sometimes in the literature spans have been refereed as *V-shape diagrams*; the reason is that the graph used to define spans has a geometrical shape remembering the capital letter “V”. Analogously, it makes sense to talk about *W-shape diagrams* to refer to the last case given in [Figure 12](#).

Some other families of diagrams with a name in the literature are the following:

- A *sink* is a family of morphisms all sharing the same target.
- A *source* is a family of morphisms all sharing the same source.
- A *connected* diagram is one given by a graph such that for every two nodes, there is an edge connecting them (without worrying about the orientation).
- a *thin* diagram is one given by a graph such that there is at most one edge between any two nodes.

A very illustrative explanation of the employed terminologies of sink and source can be found in the first page of [1, Chapter 3]. Notice that spans are a particular case of sources, while cospans are a particular case of sinks. Also notice that all diagrams in [Figure 12](#) are thin; and all of them, except for the first one, are connected.

Before introducing colimits of a diagram \mathcal{D} in a category **C** we introduce cocones.⁶

Definition 4.3 (Cocones). A *cocone* c over a diagram \mathcal{D} in a category **C** is an object O in **C** together with a family (indexed by the nodes in the graph associated with \mathcal{D}) $\{c_X\}_{X \in \text{Nodes}}$ of morphisms in **C** such that:

⁵ Strictly speaking here one needs to consider only those graphs which are the support of a category, but for the purpose of this report it is not necessary to worry about this detail.

⁶For the dual notions of limits and cones we refer the reader to literature (e.g., [61]).


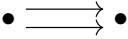
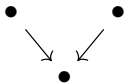
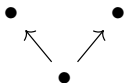
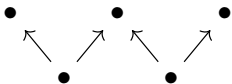
| DIRECTED GRAPH | WHAT IS A DIAGRAM? | TERMINOLOGY |
|--|--|---------------|
|  | $A \quad B$ | |
|  | $A \begin{array}{c} \xrightarrow{f} \\ \xrightarrow{g} \end{array} B$ | |
|  | $\begin{array}{ccc} B & & C \\ & \searrow f & \swarrow g \\ & A & \end{array}$ | <i>cospan</i> |
|  | $\begin{array}{ccc} B & & C \\ & \swarrow f & \searrow g \\ & A & \end{array}$ | <i>span</i> |
|  | $\begin{array}{ccccc} B & & A & & C \\ & \swarrow f & \nearrow h_1 & \nwarrow h_2 & \nearrow g \\ & A_1 & & A_2 & \end{array}$ | |

Figure 12: Some basic diagrams

- c_X has source X (for every node X);
- c_X has target O (for every node X);
- $f; c_Y = c_X$ (for every edge f from node X to node Y).

We refer to the pointed object O , which is called the *apex* of c , as $\text{apex}(c)$. The collection of all cocones over \mathcal{D} is denoted by $\text{Cocones}(\mathcal{D}, -)$. \dashv

It is obvious from Definition 4.3 that all cocones are sinks.⁷ Notice that the third condition in Definition 4.3 is expressing a bunch (one for every edge) of commutativity conditions for triangular graphs; this fact is sometimes emphasised using the terminology *commutative cocone* instead of just saying cocone. In Figure 13, we illustrate cocones for some particular diagrams (the first row deals with the empty diagram, i.e., the one without objects nor morphisms).

Looking at the examples in Figure 13 one realises that sometimes it is not necessary to represent all morphisms of a cocone. For example, when considering a cocone of the span $B \xleftarrow{f} A \xrightarrow{g} C$ it is enough to know the morphisms c_B and c_C because (by the triangular conditions) c_A must be equal to $f; c_B$ (and also equal to $g; c_C$). A greater simplification is even possible in

⁷ Although in the literature, for the sake of brevity, it is common to find that instead of explicitly introducing a cocone c it is only introduced the object $\text{apex}(c)$. However, it is worth emphasising that the definition of cocone indeed refers to the family $\{c_X\}_{X \in \text{Nodes}}$.

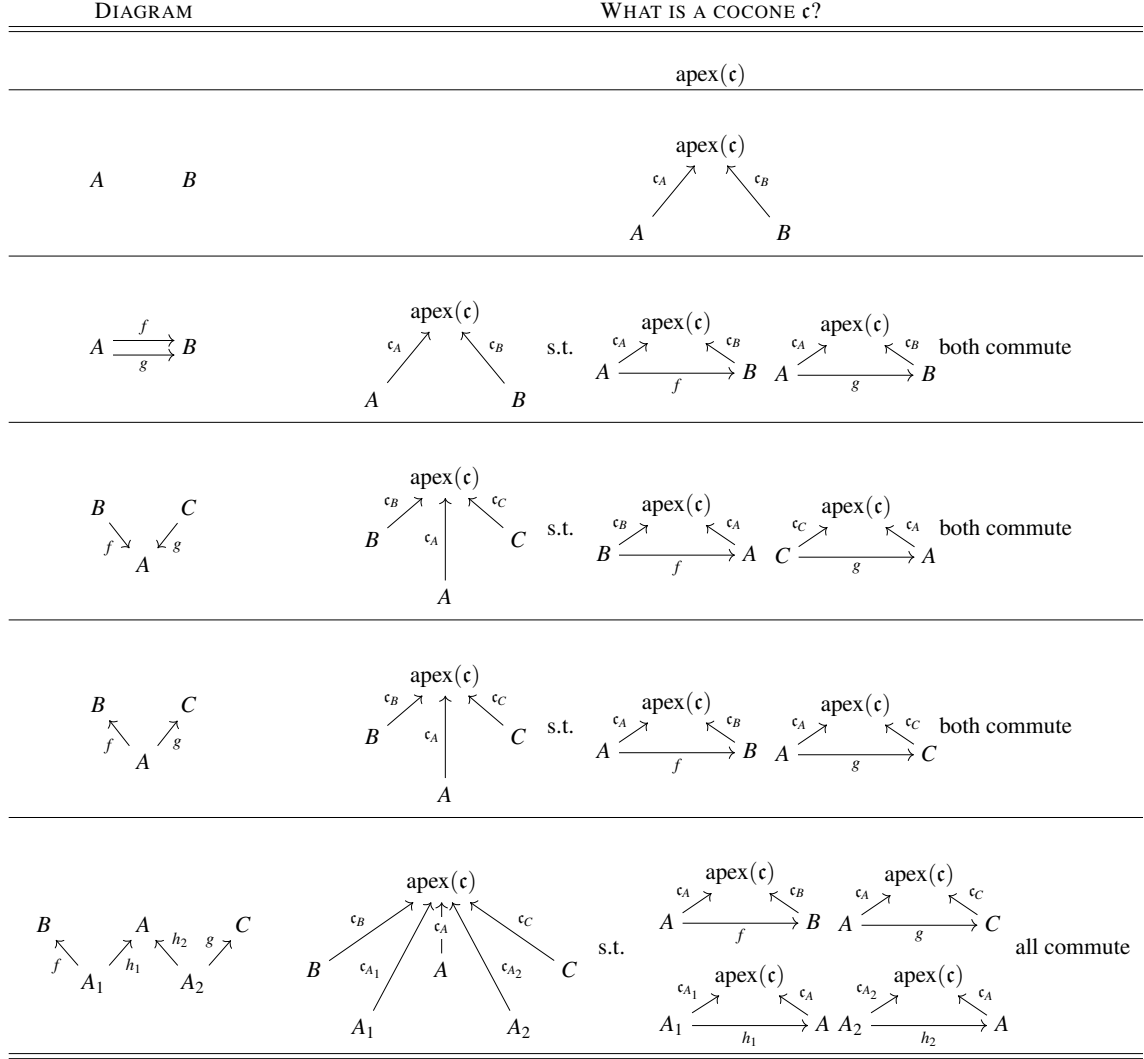


Figure 13: Some basic cocones

the case of a cocone of the cospan $B \xrightarrow{f} A \xleftarrow{g} C$. Here, it is enough to know the morphism c_A .

It is rather trivial noticing that every cocone \mathfrak{c} over a diagram \mathcal{D} induces a function $H_{\mathfrak{c}}$ defined by

$$H_{\mathfrak{c}} : \text{hom}(\text{apex}(\mathfrak{c}), -) \longrightarrow \text{Cocones}(\mathcal{D}, -)$$

$$f \longmapsto \mathfrak{c}; f$$

With the notation $\mathfrak{c}; f$ we obviously refer to the family $\{g; f\}_g$ is a morphism in \mathfrak{c} , i.e., $\{c_X; f\}_{X \in \text{Nodes}}$. These induced functions can be used to define that two cocones \mathfrak{c} and \mathfrak{d} (over the same diagram) are *isomorphic* when there is some isomorphism f in \mathbf{C} such that $\mathfrak{d} = H_{\mathfrak{c}}(f)$.

Definition 4.4 (Colimit). A cocone \mathfrak{c} over a diagram \mathcal{D} in a category \mathbf{C} is said to be a *colimit* if the function $H_{\mathfrak{c}}$ is a bijection. We write $\text{colim}(\mathcal{D}, \mathbf{C})$, or simply $\text{colim}(\mathcal{D})$, to refer to a colimit; and we will use $\text{colim}(\mathcal{D}, \mathbf{C})$ or $\text{colim}(\mathcal{D})$ for the pointed object in the cocone $\text{colim}(\mathcal{D}, \mathbf{C})$. \dashv


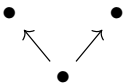
| LIMIT | DIRECTED GRAPH | COLIMIT |
|----------------|---|------------------|
| final object | | initial object |
| binary product | • • | binary coproduct |
| equaliser | • \rightrightarrows • | coequaliser |
| pullback |  | |
| |  | pushout |

Figure 14: Some famous (and simple) colimits and limits

It is worth noticing that [Definition 4.4](#) can be rephrased as claiming that every cocone over \mathcal{D} is of the form $\mathbf{c}; f$ for some unique morphism f . This remark allows us to rewrite the existence of a colimit as saying that: for every cocone over the same diagram, there is exactly one solution for a univariate system, using the cocone as parameters, of morphism equations. As an example, we illustrate this fact for the case of a colimit of a span, which is also called *pushout*.

Definition 4.5 (Pushout). Given a span $B \xleftarrow{f} A \xrightarrow{g} C$, a *pushout* of this span is a colimit

(see [Definition 4.4](#)), i.e., it is a cocone $\begin{array}{ccc} & \text{apex}(\mathbf{c}) & \\ \mathbf{c}_B \nearrow & \uparrow \mathbf{c}_A & \nwarrow \mathbf{c}_C \\ B & A & C \end{array}$ such that whenever $\begin{array}{ccccc} & D & & & \\ \mathfrak{d}_B \nearrow & \uparrow \mathfrak{d}_A & \nwarrow \mathfrak{d}_C & & \\ B & A & C & & \\ \nwarrow f & \uparrow & \nearrow g & & \end{array}$

commutes, it holds that the univariate system

$$\mathbf{c}_B; h = \mathfrak{d}_B \quad \mathbf{c}_A; h = \mathfrak{d}_A \quad \mathbf{c}_C; h = \mathfrak{d}_C$$

of morphism equations has a unique solution for h . \dashv

It is always the case that two colimits over the same diagram are isomorphic cocones, i.e., *colimits are unique up to isomorphism*. Indeed, if \mathbf{c} is a colimit, then the collection of all colimits is exactly $\{\mathbf{c}; f \mid f \text{ is isomorphism with } \text{src}(f) = \text{apex}(\mathbf{c})\}$. On the other hand, the existence of a colimit is in general not guaranteed; it depends very much on the diagram \mathcal{D} and the category \mathbf{C} .

Let us now mention two facts that restricts which cocones can be a colimit. The first fact is a trivial consequence of the injectivity of the function $H_{\mathbf{c}}$: all colimits \mathbf{c} have to be *jointly epimorphic*, which means that whenever h_1 and h_2 are two morphisms with source $\text{apex}(\mathbf{c})$ and

such that “ $c_X; h_1 = c_X; h_2$ for every node X ”, then $h_1 = h_2$.⁸ The second fact, also obvious from [Definition 4.4](#), is that for every object E , the set $\text{Cocones}(\mathcal{D}, E)$ (i.e., the collection of cocones over \mathcal{D} with apex E) must have the same cardinal than the set $\text{hom}(\text{apex}(\mathfrak{c}), E)$. These two facts are, in general, very powerful tools to recognise possible candidates as a colimit over a diagram. In the particular cases of **Set** and **Pfn** the second fact can be used to completely determine the possible apexes of colimits (since all objects with the same cardinal are isomorphic). [Remark 4.6](#) describes the method for the case of **Pfn**.

Remark 4.6 (Cardinality trick for **Pfn**). Consider the natural number m of cocones over \mathcal{D} with apex $\{1\}$ (i.e., a singleton set). Then, the cardinal of an object $\text{colim}(\mathcal{D}, \mathbf{Pfn})$ has to be the only natural number n such that $m = 2^n$. ■

For the colimits (and limits) of some concrete diagrams it is common to use an specific terminology. Some common examples of colimits are *binary coproducts*, *coequalisers* and *pushouts*, whose definition can be found in [Figure 14](#).⁹ A well-known result in the literature (see for instance [53, Section 4.6]) is, that the colimit of an arbitrary (finite) diagram \mathcal{D} in **C** can be always expressed as the coequaliser of an alternative diagram, such that the alternative diagram uses empty and binary coproducts of \mathcal{D} , whenever the involved coequaliser and coproducts exist. In consequence, we can consider coproducts and coequalisers as the basic pieces of the colimit construction. We are not giving the precise proof and formulation of this result here, but we provide an illustration in [Example 4.7](#) for the last two diagram cases given in [Figure 13](#). These two diagrams will be the important ones for the blending development to be explained in [Section 4.3](#).

In the sequel we will use the symbol \oplus to denote coproducts (in case they exist).¹⁰ To be more precise, for every $n \in \mathbb{N}$ the cocone

$$\begin{array}{c} A_1 \oplus \dots \oplus A_n \\ \swarrow \quad \searrow \quad \swarrow \quad \searrow \\ A_1 \quad A_2 \quad \dots \quad A_{n-1} \quad A_n \end{array}$$

(The diagram shows arrows labeled $i_{A_1}, i_{A_2}, \dots, i_{A_{n-1}}, i_{A_n}$ pointing from each A_i to the coproduct $A_1 \oplus \dots \oplus A_n$.)

is giving the n -ary coproduct of a diagram (without morphisms) given by the objects A_1, \dots, A_n . By the definition of colimit this means that there is a bijection between

$$\text{hom}(A_1 \oplus \dots \oplus A_n, -) \quad \text{and} \quad \text{hom}(A_1, -) \times \text{hom}(A_2, -) \times \dots \times \text{hom}(A_n, -);$$

to avoid any confusion we point out that \times refers, here and in the rest of the report, to the usual Cartesian product. We will use the notation $h_1 \oplus \dots \oplus h_n$ to denote the homomorphism in $\text{hom}(A_1 \oplus \dots \oplus A_n, -)$ with is mapped, under such bijection, to the ordered tuple (h_1, \dots, h_n) .

Example 4.7 (Colimits as coequalisers of coproducts). In this example, we will consider the last two diagrams in [Figure 13](#). Firstly, we focus on the colimit of $B \xleftarrow{f} A \xrightarrow{g} C$. In this case, it happens that this colimit coincides with the coequaliser of the diagram

⁸ It is worth pointing out that when **C** has coproducts, the following (i) and (ii) are equivalent. (i) $\{c_X\}_{X \in \text{Nodes}}$ is jointly epimorphic; (ii) the single morphism $\bigoplus \{c_X\}_{X \in \text{Nodes}}$ is epimorphic. This relationship explains the intuition behind this “jointly” terminology.

⁹ This standard terminology is chosen in such a way that products and coproducts generalise multiplication and addition of natural number.

¹⁰ The same symbol will also be used for the disjoint union, but this will not cause any misunderstanding as a consequence of the results given in [Examples 4.9](#) and [4.10](#).

$$A \oplus A \begin{array}{c} \xrightarrow{h} \\ \xrightarrow{l} \end{array} A \oplus B \oplus C$$

where $h := (\text{id}_A; i_A) \oplus (\text{id}_A; i_A)$ and $l := (f; i_B) \oplus (g; i_C)$.

Next we consider the colimit of $B \xleftarrow{f} A_1 \xrightarrow{h_1} A \xleftarrow{h_2} A_2 \xrightarrow{g} C$. For this diagram, the general theory tells us that the colimit is the coequaliser of the diagram

$$A_1 \oplus A_1 \oplus A_2 \oplus A_2 \begin{array}{c} \xrightarrow{h} \\ \xrightarrow{l} \end{array} A \oplus A_1 \oplus A_2 \oplus B \oplus C$$

where $h := (\text{id}_{A_1}; i_{A_1}) \oplus (\text{id}_{A_1}; i_{A_1}) \oplus (\text{id}_{A_2}; i_{A_2}) \oplus (\text{id}_{A_2}; i_{A_2})$ and $l := (f; i_B) \oplus (h_1; i_A) \oplus (h_2; i_A) \oplus (g; i_C)$. \square

Definition 4.8. A category is said to be *cocomplete* in case that for all diagrams in \mathbf{C} there is a colimit. Analogously, *complete* refers to the existence of all limits; and *bicomplete* refers to being both completeness and cocompleteness. \dashv

The categories **Set** and **Pfn** introduced in Example 4.2 are well-known to be bicomplete. Moreover, it is also known that if all morphisms of a diagram \mathcal{D} in **Pfn** are total functions (i.e., the diagram lives inside **Set**) then $\text{colim}(\mathcal{D}, \mathbf{Set}) = \text{colim}(\mathcal{D}, \mathbf{Pfn})$, i.e., it does not matter whether one computes the colimit in **Set** or in **Pfn**. Let us mention that this last remark is known to be false for the case of limits.¹¹

The general result illustrated in Example 4.7 (see [53, Section 4.6]) allows us to compute colimits over finite diagrams (in an arbitrary category) once it is known how to compute initial objects (i.e., empty coproducts), binary coproducts and coequalisers. Thus, in the following two examples we only focus on how computing such basic colimits for the categories **Set** and **Pfn**.

Example 4.9 (Finite colimits in **Set**). In **Set**, the initial object is the empty set \emptyset , and the binary

¹¹ An easy counterexample can be obtained considering the categorical product of two singleton sets, for example, $A := \{\rightarrow\}$ and $B := \{\bullet\}$. A quick way to convince oneself that the categorical product computed in **Set** is different than in **Pfn** is to use the cardinality trick described in Remark 4.6 (but dualised, in order to use it for limits instead of colimits). The fact that there are exactly 4 cones in **Pfn** with apex $\{\odot\}$ (i.e., a singleton) forces that the coproduct in **Pfn** must have 3 elements; on the other hand, using that there is exactly 1 cone in **Set** with apex $\{\odot\}$ one deduces that the coproduct in **Set** must have 1 element.

Indeed, the content of the previous paragraph is generalised in the following well-known statement (see [63, p. 20]):

- the product in **Set** of A and B is given by the cone $\begin{array}{ccc} A & & B \\ & \nwarrow \pi_A & \nearrow \pi_B \\ & O & \end{array}$ where O is the Cartesian product of A and B

(i.e., $O := A \times B$), and the morphisms π_A and π_B are the “projections” from the Cartesian product.

- the product in **Pfn** of A and B is given by the cone $\begin{array}{ccc} A & & B \\ & \nwarrow c_A & \nearrow c_B \\ & O & \end{array}$ where $O := (A \times B) \oplus A \oplus B$ (here \oplus refers,

as above, to the disjoint union), the morphism c_A is $\pi_A \oplus \text{id}_A \oplus \emptyset$, and the morphism c_B is $\pi_B \oplus \emptyset \oplus \text{id}_B$.

The last statement is providing the intuition that for the product in **Pfn** of two sets one needs to consider the ordered pairs in the Cartesian product, but also add those ordered pairs that are missing one element of the pair.

coproduct of objects A and B is the cocone

$$\begin{array}{ccc} & O & \\ i_A \nearrow & & \nwarrow i_B \\ A & & B \end{array}$$

where O is the disjoint union of A and B , which can be explicitly defined, among many other ways, as $O := (A \times \{0\}) \cup (B \times \{1\})$. The morphisms i_A and i_B are the ‘inclusions’ in the disjoint union.

The coequaliser (in **Set**) of a diagram $A \xrightarrow[f]{g} B$ is the cocone

$$\begin{array}{ccc} & O & \\ c_A \nearrow & & \nwarrow c_B \\ A & & B \end{array}$$

where

- O is the quotient¹² of B under the equivalence relation \equiv , defined as ‘the smallest equivalence relation of B extending $\{(f(a), g(a)) \mid a \in A\}$ ’ (i.e., O is B/\equiv),
- the morphism c_B is the projection under \equiv (i.e., it sends an element x of B to its equivalence class x/\equiv),
- the morphism c_A is $f; c_B$ (which also coincides with $g; c_B$). □

Example 4.10 (Finite colimits in **Pfn**). The initial object in **Pfn** is the empty set \emptyset . The binary coproduct (in **Pfn**) of objects A and B is the same cocone given in [Example 4.9](#). Now, we determine the cocone

$$\begin{array}{ccc} & O & \\ c_A \nearrow & & \nwarrow c_B \\ A & & B \end{array}$$

that is the coequaliser (in **Pfn**) of a diagram $A \xrightarrow[f]{g} B$. Before describing the coequaliser, it is convenient to introduce some auxiliary notation: we consider \equiv to be the equivalence relation of B defined like in [Example 4.9](#), and we also consider the following partial function

$$h : B \longrightarrow B/\equiv$$

$$x \longmapsto \begin{cases} \text{undefined} & \text{if there exists } y \in A \text{ s.t. } f(y) = x \text{ and } g(y) = \text{undefined} \\ \text{undefined} & \text{if there exists } y \in A \text{ s.t. } f(y) = \text{undefined and } g(y) = x \\ x/\equiv & \text{otherwise} \end{cases}$$

Then, the coequaliser happens to be the cocone where

¹² We remind the reader that, as it is customary, the quotient under an equivalence relation is the collection of all its equivalence classes.

- O is the (set-theoretical) image of h ,
- the morphism $c_B : B \rightarrow O$ is the partial function given by the same ordered pairs than in h ,
- the morphism c_A is $f; c_B$ (which also coincides with $g; c_B$).

The only possible difference between h and c is in the target object; but unfortunately, such construction cannot in general be simplified because there are cases¹³ where h and c_B are really different, i.e., O is really a proper subset of B/\equiv . Indeed, for finite sets the cardinality trick illustrated in [Remark 4.6](#) gives a quick method to determine whether O is or is not a proper subset of B/\equiv . \square

In this last example, it is worth noticing that the partial functions c_B and c_A are indeed total functions when f and g are total. Moreover, in such a case the coequalisers given in [Example 4.9](#) and in [Example 4.10](#) coincide. As a trivial consequence of this fact one gets that finite diagrams given by total functions have the same colimit in **Set** and in **Pfn**.

The categorical approach to blending developed by Goguen is particularly interested on computing colimits over diagrams $B \xleftarrow{f} A \xrightarrow{g} C$ and $B \xleftarrow{f} A_1 \xrightarrow{h_1} A \xleftarrow{h_2} A_2 \xrightarrow{g} C$, i.e., V-shaped and W-shaped diagrams. Next we focus on them for the particular cases of **Set** and **Pfn**. It is obvious how they can be computed by combining [Example 4.7](#) with [Examples 4.9](#) and [4.10](#); but in the literature one can find explicit methods for such computations. To be more specific let us at least mention that

- the pushout of $B \xleftarrow{f} A \xrightarrow{g} C$ in **Set** has as apex the set $(B \oplus C)/\equiv$, where \equiv is the smallest equivalence relation over $B \oplus C$ extending $\{(f(a), g(a)) : a \in A\}$.
- the pushout of $B \xleftarrow{f} A \xrightarrow{g} C$ in **Pfn** has as apex the set $(B \setminus f[A]) \oplus (C \setminus g[A]) \oplus Q$ where Q is will be soon defined. To do so, we consider \equiv to be the the smallest equivalence relation over A extending

$$\{(a_1, a_2) \in A \times A : f(a_1) = f(a_2)\} \cup \{(a_1, a_2) \in A \times A : g(a_1) = g(a_2)\},$$

and \hat{A} to be the largest subset of $\text{Dom}(f) \cap \text{Dom}(g)$ that is closed under \equiv . Then, Q is defined as the quotient of \hat{A} under the equivalence relation \equiv . For more details on such pushout construction one can look at [65, p. 4]. Nevertheless, it is worth saying that in most occasions the cardinal of Q can be directly determined using the cardinality trick illustrated in [Remark 4.6](#).¹⁴

¹³ One of such cases is the diagram given by $A := \{\blacktriangleright\}$, $B := \{\blacktriangleright\}$, $f := \{(\blacktriangleright, \blacktriangleright)\}$ and $g := \emptyset$. In such a case the apex of its coequaliser is the empty set.

¹⁴ For example, let us consider the diagram given by $A := \{\bullet, \blacksquare\}$, $B := \{\textcircled{1}, \textcircled{2}\}$, $C := \{\textcircled{1}, \textcircled{2}\}$, $f := \{(\bullet, \textcircled{1}), (\blacksquare, \textcircled{1})\}$ and $g := \{(\blacksquare, \textcircled{2})\}$. Using that it exactly has 4 cocones (in **Pfn**) over a set with only one element, one deduces that the set Q in the above description has to be the empty set. Thus, the following cocone

$$\begin{array}{ccccc} & & \{\textcircled{2}, \textcircled{1}\} & & \\ & \nearrow & \uparrow & \nwarrow & \\ \{(\textcircled{2}, \textcircled{2})\} & & \emptyset & & \{(\textcircled{1}, \textcircled{1})\} \\ \nearrow & & \downarrow & & \nwarrow \\ \{\textcircled{1}, \textcircled{2}\} & & \{\bullet, \blacksquare\} & & \{\textcircled{1}, \textcircled{2}\} \end{array} .$$

is a pushout.

- the colimit of a W-shaped diagram $B \xleftarrow{f} A_1 \xrightarrow{h_1} A \xleftarrow{h_2} A_2 \xrightarrow{g} C$ can be calculated (in any category) using three consecutive pushouts. Firstly, one computes, respectively, the pushouts

$$\begin{array}{ccc}
 & D & \\
 \nearrow & \uparrow & \nwarrow \\
 \partial_B & \partial_{A_1} & \partial_A \\
 \swarrow & \downarrow & \searrow \\
 B & A_1 & A
 \end{array}
 \quad \text{and} \quad
 \begin{array}{ccc}
 & E & \\
 \nearrow & \uparrow & \nwarrow \\
 \epsilon_A & \epsilon_{A_2} & \epsilon_C \\
 \swarrow & \downarrow & \searrow \\
 A & A_2 & C
 \end{array}$$

of the V-shaped diagrams $B \xleftarrow{f} A_1 \xrightarrow{h_1} A$ and $A \xleftarrow{h_2} A_2 \xrightarrow{g} C$. Next, one computes the pushout

$$\begin{array}{ccc}
 & F & \\
 \nearrow & \uparrow & \nwarrow \\
 f_D & f_A & f_E \\
 \swarrow & \downarrow & \searrow \\
 D & A & E
 \end{array}$$

of the V-shaped diagram $D \xleftarrow{\partial_A} A \xrightarrow{\epsilon_A} E$. Then, by an standard argument in category theory it happens that the cocone

$$\begin{array}{ccccc}
 & & F & & \\
 & \nearrow & \uparrow & \nwarrow & \\
 \partial_B; f_D & \partial_{A_1}; f_D & f_A & \epsilon_{A_2}; f_E & \epsilon_C; f_E \\
 \swarrow & \downarrow & \downarrow & \searrow & \searrow \\
 B & A_1 & A & A_2 & C
 \end{array}$$

is indeed the colimit of the initially considered W-shaped diagram.

To finish this section about category theory preliminaries we introduce several categories (and bicategories) that will play a role in [Section 4.3](#), where we present our theory of colimits and amalgams.

Definition 4.11. Let \mathbf{C} be a category that is closed under pullbacks, i.e., the limits of all cospans exist.

- The category **Rel** has sets as objects, and a morphism from A to B is a subset of the direct product $A \times B$.
- The bicategory¹⁵ **Span**(\mathbf{C}) has the same objects than \mathbf{C} , and a morphism from an object A to an object B is a span $A \xleftarrow{f} D \xrightarrow{g} B$ in \mathbf{C} . Composition of spans $A \xleftarrow{f} D \xrightarrow{g} B$

¹⁵ It is not a category because the composition defined here does not satisfy the associativity $f; (g; h) = (f; g); h$. Nevertheless, this property holds when replacing equality with isomorphism, which corresponds to claiming to be a bicategory (see [7, 50]).

and $B \xleftarrow{h} E \xrightarrow{l} C$ is defined (up to isomorphism) using the cone

$$\begin{array}{ccc} D & & B \\ & \swarrow \scriptstyle c_D & \uparrow \scriptstyle c_B \\ & \text{apex}(\mathfrak{c}) & \\ & \nwarrow \scriptstyle c_E & \end{array} \quad E$$

obtained as the pullback of $D \xrightarrow{g} B \xleftarrow{h} E$. The result of the composition is by definition the span $A \xleftarrow{c_D;f} \text{apex}(\mathfrak{c}) \xrightarrow{c_E;l} C$.

- The bicategory $\mathbf{MSpan}(\mathbf{C})$ is defined as $\mathbf{Span}(\mathbf{C})$ except for only considering as morphisms from A to B the *mono spans* from A to B , which are defined to be the spans $A \xleftarrow{f} D \xrightarrow{g} B$ where f is a monomorphism in \mathbf{C} .
- The category $\mathbf{Rel}(\mathbf{C})$ has the same objects than \mathbf{C} , and a morphism from an object A to an object B is the isomorphic class¹⁶ of a span $A \xleftarrow{f} D \xrightarrow{g} B$ in \mathbf{C} . Composition is defined using the same pullback construction than before.¹⁷
- The category $\mathbf{Pfn}(\mathbf{C})$ is defined as $\mathbf{Rel}(\mathbf{C})$ except for only considering as morphisms the isomorphic classes of mono spans. A *partial morphism* from A to B is defined as the isomorphic class of a mono span $A \xleftarrow{f} D \xrightarrow{g} B$. Thus, the morphisms in $\mathbf{Pfn}(\mathbf{C})$ are nothing else than the partial morphisms. \dashv

It is well-known that $\mathbf{Pfn}(\mathbf{Set})$ is (categorically) equivalent to the category \mathbf{Pfn} (and also equivalent to the category of pointed sets). Even more, $\mathbf{Pfn}(\mathbf{Set})$ and \mathbf{Pfn} are isomorphic categories: there is an obvious bijection between partial morphisms in \mathbf{Set} and morphisms in \mathbf{Pfn} . Thus, $\mathbf{Pfn}(\mathbf{C})$ can be considered as a natural candidate for generalising the category \mathbf{Pfn} of partial functions. On the other hand, $\mathbf{Rel}(\mathbf{Set})$ does not exhibit so clearly the same behaviour than \mathbf{Rel} . For example, two non-isomorphic spans in \mathbf{Set} like

$$A \xleftarrow{\pi_1} A \times C \times B \xrightarrow{\pi_3} B \quad \text{and} \quad A \xleftarrow{\pi_1} A \times D \times B \xrightarrow{\pi_3} B \quad (4.1)$$

¹⁶ In other words, the spans $A \xleftarrow{f} D \xrightarrow{g} B$ and $A \xleftarrow{f'} D' \xrightarrow{g'} B$ are considered equal in $\mathbf{Rel}(\mathbf{C})$ when there is

an isomorphism $h : D \rightarrow D'$ such that

$$\begin{array}{ccccc} & & D & & \\ & f & \downarrow & g & \\ A & \swarrow & h & \searrow & B \\ & f' & \downarrow & g' & \\ & & D' & & \end{array} \quad \text{commutes.}$$

¹⁷ In the literature it is quite common (e.g., [23, p. 38] and [36, Section 2]) to consider *relations* as a proper subclass of the one we have just introduced: consider only the subobjects of the terminal span from A to B . This alternative proposal has the advantage of avoiding the problems remarked around Equation (4.1). The reason to not consider this approach is simply not introducing the technical details behind it; since our goal will be to deal with partial functions, and not relations, this difference is not so important.

induce the same binary relation as a subset of $A \times B$ (at least if C and D are non-isomorphic non-empty sets).

Among partial morphisms from A to B there are some outstanding ones which we call *total*. They are, by definition, the isomorphic classes of mono spans $A \xleftarrow{f} D \xrightarrow{g} B$ where f is an isomorphism. It is obvious that the total morphisms form a subcategory (i.e., total morphisms are closed under composition and the identities are total) of $\mathbf{Pfn}(\mathbf{C})$, and such subcategory is equivalent to \mathbf{C} .

The categories $\mathbf{Pfn}(\mathbf{C})$ of partial morphisms are well-known in the literature. They were firstly considered in [66] using an even more general setting, there the authors introduce for every class \mathcal{M} of monomorphisms satisfying certain constraints (see [35, Definitions 6 and 7] for a modern presentation) a category $\mathbf{Pfn}(\mathbf{C}, \mathcal{M})$. Our category $\mathbf{Pfn}(\mathbf{C})$ corresponds to choosing \mathcal{M} as the class of all monomorphisms. We have decided to avoid this more general framework just for the aim of simplicity.

4.1.2 Colimits in Ordered Categories

The aim of this section is to explain Goguen's framework for the blending theory developed by Fauconnier and Turner. This framework is developed in [24] (mainly in its Section 5 and its Appendix B), and instead of using plain categories it is based on the following enriched version.

Definition 4.12 (Ordered category). An *ordered category* is a category \mathbf{C} such that

- for every two objects A and B , there is a partial order $\sqsubseteq_{A,B}$ on the set $\text{hom}(A, B)$;
- composition is monotonic with respect to \sqsubseteq in both arguments (i.e., if $f_1 \sqsubseteq g_1$ and $f_2 \sqsubseteq g_2$, then $f_1; f_2 \sqsubseteq g_1; g_2$).

Concerning notation it is customary to omit indices and simply use \sqsubseteq (see second item), i.e., \sqsubseteq can be considered to be $\bigcup \{ \sqsubseteq_{A,B} \mid A, B \in \text{obj}(\mathbf{C}) \}$. \dashv

Ordered categories are a simple case of so-called 2-categories (see [51, 38, 45]). Here, there is at most one 2-cell between two 1-cells (i.e., morphisms). Thus, ordered categories lie between plain 1-categories and 2-categories. For this reason, Goguen [24] introduces the term $\frac{3}{2}$ -categories to refer to ordered categories.¹⁸ Other names have also been used in the literature to refer to ordered categories: e.g., locally partially ordered categories, locally posetal categories, Pos-enriched categories, etc. A reference where one can find a detailed approach to ordered categories, but without considering all the difficulties due to dealing with general 2-categories, is [40].

Example 4.13. The categories $\mathbf{Pfn}(\mathbf{C})$ are ordered categories in the following sense: consider two partial morphisms from A to B , given respectively by the isomorphic classes of the mono spans

$$A \xleftarrow{f} D \xrightarrow{g} B \quad \text{and} \quad A \xleftarrow{f'} D' \xrightarrow{g'} B \quad .$$

¹⁸ The definition given in [24, Definition 6] also states that the identity morphism id_A has to be maximal in $\text{hom}(A, A)$. We do not require this last condition in the definition we have finally decided to adopt, but this property will also hold for the most natural examples of ordered categories (see Example 4.13).

We say that the first partial morphism is *below* the second one (denoted \sqsubseteq) if there is a morphism $h : D \rightarrow D'$ such that

$$\begin{array}{ccc} & D & \\ f \swarrow & \downarrow h & \searrow g \\ A & & B \\ f' \swarrow & \downarrow & \searrow g' \\ & D' & \end{array}$$

commutes. In such a case it must hold that h is also a monomorphism. Under these assumptions it holds that \sqsubseteq is a partial order: antisymmetry is obtained using the cancellativity property given by monomorphisms.¹⁹ Moreover, the partial morphisms that are total are the maximal elements of the partial order \sqsubseteq just defined. We will refer to this partial order \sqsubseteq as the *extension partial order*. \square

Example 4.13 tells us in particular that $\mathbf{Pfn}(\mathbf{Set})$ is an ordered category; for this case it holds that

$$f \sqsubseteq g \quad \text{iff} \quad \text{whenever } f \text{ is defined, } g \text{ is also defined and it agrees with } f.$$

Moreover, the structure of the partial order \sqsubseteq resembles a lot (but is not) a lattice because:

- for every two partial morphisms f_1 and f_2 (with the same sources and targets), there is also a partial morphism $f_1 \sqcap f_2$ which is the infimum in \sqsubseteq ;
- for every two partial morphisms f_1 and f_2 , if they are *compatible* (i.e., if there is some g such that $f_1 \sqsubseteq g$ and $f_2 \sqsubseteq g$) then there is also a partial morphism $f_1 \sqcup f_2$ which is the supremum in \sqsubseteq .

It is also worth noticing that the partial orders $\sqsubseteq_{A,B}$ are *directed-complete partial orders* (*dcpo*), which means that every directed subset has a supremum (which we will denote using the symbol \sqcup). And the composition function can be checked to be *Scott-continuous*, which means that: for every directed family $\{g_i \mid i \in I\}$ of partial functions and every partial function f ,

- $\{f; g_i \mid i \in I\}$ is also directed and its supremum is $f; \sqcup \{g_i \mid i \in I\}$;
- $\{g_i; f \mid i \in I\}$ is also directed and its supremum is $\sqcup \{g_i \mid i \in I\}; f$.

Notice also that \mathbf{Set} is equivalent to the subcategory of $\mathbf{Pfn}(\mathbf{Set})$ given by total morphisms.

By means of Equation (4.1), we point out that $\mathbf{Rel}(\mathbf{Set})$ is not ordered using the extension partial order. Nevertheless, the category \mathbf{Rel} introduced in Definition 4.11 is obviously ordered using the set theoretical inclusion. Thus, while working with (isomorphic classes of) mono spans is a right approach to deal with partial functions and orderability, working with spans is not the right approach to deal with binary relations and orderability. Approaches that work well for such

¹⁹ In order to illustrate the necessity of this hypothesis, let us note that in the category $\mathbf{Rel}(\mathbf{Set})$ the relation \sqsubseteq just defined is not antisymmetric (and so is not a partial order). This can be seen, for example, using the isomorphic classes of the spans given in Equation (4.1).

case is considering *allegories* (and/or *collagories*) [23, 39, 40] or *Cartesian bicategories* [11, 10], and the price to pay is essentially the introduction, respectively, of a converse operation or a tensor product. For the sake of simplicity we will avoid the introduction of such frameworks, and keep this report to deal only with $\mathbf{Pfn}(\mathbf{C})$.

In the context of ordered categories there are, at least, two very natural alternative possibilities concerning colimits (see [40, Chapter 4]). One of them produces an strengthening of the plain notion of colimits, and we will refer to them as *ordered colimits*. The other one accepts a more general class of diagrams, which instead of considering functor one considers so-called *lax functors*, where commutativity is replaced with semicommutativity). The latter follows a very similar pattern than the one given for colimits in Definition 4.4, and respective colimits are called *lax colimits*.

Definition 4.14 (Ordered colimit, see [40, Definition 4.1.2]). A cocone \mathfrak{c} over a diagram \mathcal{D} in an ordered category \mathbf{C} is said to be an *ordered colimit* in case that the function $H_{\mathfrak{c}}$ introduced in Page 31 is an order-isomorphism (and therefore also a bijection) between the partial orders $\langle \text{hom}(\text{apex}(\mathfrak{c}), -), \sqsubseteq \rangle$ and $\langle \text{Cocones}(\mathcal{D}, -), \sqsubseteq^* \rangle$. The ordered \sqsubseteq^* considered among cocones is the one defined component-wise, that is, given two cocones $\mathfrak{c} := \{c_X\}_{X \in \text{Nodes}}$ and $\mathfrak{d} := \{d_X\}_{X \in \text{Nodes}}$ with the same object it holds that

$$\mathfrak{c} \sqsubseteq^* \mathfrak{d} \quad \text{iff} \quad c_X \sqsubseteq d_X \text{ for every node } X. \quad \dashv$$

From the very Definition 4.14 it is obvious that if \mathfrak{c} is an ordered colimit, then it holds that whenever h_1 and h_2 are two morphisms with source $\text{apex}(\mathfrak{c})$ and such that “ $c_X; h_1 \sqsubseteq c_X; h_2$ for every node X ”, then $h_1 \sqsubseteq h_2$. We will refer to such condition as being *jointly semiepipomorphic*.²⁰ It is obvious that jointly semiepipomorphic is an strengthening of the jointly epimorphic property.

In the particular case of the ordered category \mathbf{Pfn} (with the extension partial order described in Example 4.13), one can check that the colimits given in Example 4.2 are also ordered colimits.

Next, in order to introduce lax colimits we need to firstly introduce lax diagrams and lax cocones. The only difference between a functor $\mathcal{D} : \mathbf{J} \longrightarrow \mathbf{C}$ and a lax functor $\mathcal{D} : \mathbf{J} \longrightarrow \mathbf{C}$ is that instead of equality one only requires

$$\text{id}_{\mathcal{D}(A)} \sqsubseteq \mathcal{D}(\text{id}_A) \quad \text{and} \quad \mathcal{D}(f); \mathcal{D}(g) \sqsubseteq \mathcal{D}(f; g).$$

The second condition is known as *semicommutativity*, and it is common to represent it graphically as follows:

$$\begin{array}{ccc} & & \mathbf{C} \\ & \nearrow \mathcal{D}(g) & \uparrow \mathcal{D}(f; g) \\ \mathbf{B} & \sqsubseteq & \\ & \nwarrow \mathcal{D}(f) & \downarrow \\ & & \mathbf{A} \end{array}$$

²⁰ In the case there are ordered coproducts (in the sense of Definition 4.14) it is obvious that this definition also follows the same intuition explained in Footnote 8. That is, $\{c_X\}_{X \in \text{Nodes}}$ is jointly semiepipomorphic iff the single morphism $\bigoplus \{c_X\}_{X \in \text{Nodes}}$ is so.

Notice that if the ordered category satisfies that the identity morphisms are maximal, then the first condition $\text{id}_{\mathcal{D}(A)} \sqsubseteq \mathcal{D}(\text{id}_A)$ can be rewritten as saying $\text{id}_{\mathcal{D}(A)} = \mathcal{D}(\text{id}_A)$. A *lax diagram* in an ordered category \mathbf{C} is defined to be a lax functor $\mathcal{D} : \mathbf{J} \longrightarrow \mathbf{C}$. Here \mathbf{J} is just a category (no necessity to consider an ordered category).

A *lax cocone* c over a lax diagram \mathcal{D} in a category \mathbf{C} is an object O in \mathbf{C} together with a family (indexed by the nodes in the graph associated with \mathcal{D}) $\{c_X\}_{X \in \text{Nodes}}$ of morphisms in \mathbf{C} such that:

- c_X has source X (for every node X),
- c_X has target O (for every node X),
- $f; c_Y \sqsubseteq c_X$ (for every edge f from node X to node Y)

Thus, lax cocones are capturing the intuition of *semicommutative cocones*. As expected we will refer to the apex object as $\text{apex}(c)$. The collection of all lax cocones over \mathcal{D} will be denoted by $\text{laxCocones}(\mathcal{D}, -)$.

It is rather trivial noticing that every lax cocone c over a lax diagram \mathcal{D} induces a function²¹ H_c defined by

$$H_c : \begin{array}{ccc} \text{hom}(\text{apex}(c), -) & \longrightarrow & \text{laxCocones}(\mathcal{D}, -) \\ f & \longmapsto & c; f \end{array}$$

Definition 4.15 (Lax colimits, see [40, Definition 4.3.2]). A lax cocone c over a lax diagram \mathcal{D} in an ordered category \mathbf{C} is said to be a *lax colimit* when the recently introduced function H_c is an order-isomorphism (and so a bijection) between the partial orders $\langle \text{hom}(\text{apex}(c), -), \sqsubseteq \rangle$ and $\langle \text{laxCocones}(\mathcal{D}, -), \sqsubseteq^* \rangle$. The ordered \sqsubseteq^* considered among cocones is the one defined component-wise (see Definition 4.14). \dashv

It is again obvious that lax colimits must be jointly semiepipimorphic. Notice also that in case of considering a diagram \mathcal{D} (instead of an arbitrary lax diagram), the notions of lax colimit and ordered colimit collapse (up to isomorphism) if and only if all cocones are lax cocones. Thus, whenever semicommutativity is not trivially reduced to commutativity, the two recently introduced notions of colimits must be different.

In the particular case of the ordered category **Pfn** it happens that there are diagrams without lax colimits. For example, the lax colimit (indeed pushout) over the V-shaped diagram given in Footnote 14 does not exist.²²

It is well-known that the cocone of an ordered colimit is unique up to isomorphism. And the same happens for the lax cocone of a lax colimit. Goguen considers these facts to cause difficulties for the formalisation of blending, since one expects more than one way to blend concepts. For this reason he proposes the following alternative notion.²³

²¹ We use the same notation H_c than for the case of plain categories and colimits, but this is not a trouble because the context always clarifies which one we refer to.

²² For a proof of such a claim one can use a quick cardinality argument. Firstly, check that there exactly 32 lax cocones (in **Pfn**) over such diagram \mathcal{D} with apex $\{\otimes\}$. By Definition 4.15 this forces that any possible lax colimit must have cardinality 5. To finish the proof, it is enough to realise that there is no lax cocone with a 5-element apex that is jointly epimorphic.

²³ In [25, Section 3.1] it is used the expression “lax pushouts” in a naive way: this has not to be understood as a particular case of lax colimits in ordered categories.

Definition 4.16 ($\frac{3}{2}$ -Colimits, see [24, Definition 12]). A lax cocone c over a lax diagram \mathcal{D} in an ordered category \mathbf{C} is said to be a $\frac{3}{2}$ -colimit in case that for every lax cocone d (with object D) over \mathcal{D} it holds that the set

$$\{f \mid H_c(f) \sqsubseteq^* d\} \quad (\text{which is a subset of } \text{hom}(\text{apex}(c), D))$$

has a maximum element on \sqsubseteq . ⊣

Notice that this last definition is equivalent to just saying that the function

$$\begin{array}{ccc} H_c : \langle \text{hom}(\text{apex}(c), -), \sqsubseteq \rangle & \longrightarrow & \langle \text{laxCocones}(\mathcal{D}, -), \sqsubseteq^* \rangle \\ f & \longmapsto & c; f \end{array}$$

fulfills that the antiimage of principal downsets (i.e., downsets of an element) are also principal downsets. This last restatement of the notion of $\frac{3}{2}$ -colimits has the advantage of providing an easier comparison with Definition 4.15. In particular, it results obvious that if c is a lax colimit over \mathcal{D} , then it is also a $\frac{3}{2}$ -colimit.

When the ordered category involves partial orders that are dcpos and composition is Scott-continuous, then it is worth noticing that the following statements are equivalent:²⁴

1. The set $\{f \mid H_c(f) \sqsubseteq^* d\}$ has a maximum element on \sqsubseteq .
2. The set $\{f \mid H_c(f) \sqsubseteq^* d\}$ is directed, i.e., whenever $H_c(f_1) \sqsubseteq^* d$ and $H_c(f_2) \sqsubseteq^* d$ then there is some g such that $f_1 \sqsubseteq g$, $f_2 \sqsubseteq g$ and $H_c(g) \sqsubseteq^* d$.

Notice that the first condition is the one involved in Definition 4.16, and also that **Pfn** satisfies the hypotheses for such equivalence.

For the case of the diagram $B_1 \xleftarrow{f_1} A \xrightarrow{f_2} B_2$, Definition 4.16 provides the notion of $\frac{3}{2}$ -pushouts, which is Goguen's proposal for a formalisation of blending. We restate his proposal in Definition 4.17.

Definition 4.17 ($\frac{3}{2}$ -pushouts). A $\frac{3}{2}$ -pushout of a span $B_1 \xleftarrow{f_1} A \xrightarrow{f_2} B_2$ is given by a lax cocone

$$\begin{array}{ccccc} & & C & & \\ & g_1 \nearrow & \uparrow g & \nwarrow g_2 & \\ B_1 & \sqsubseteq & & \sqsupseteq & B_2 \\ & f_1 \nwarrow & \downarrow & \nearrow f_2 & \\ & & A & & \end{array}$$

²⁴ The assumptions just stated are only necessary to prove the implication $2 \Rightarrow 1$; the reverse implication always holds.

satisfying that whenever

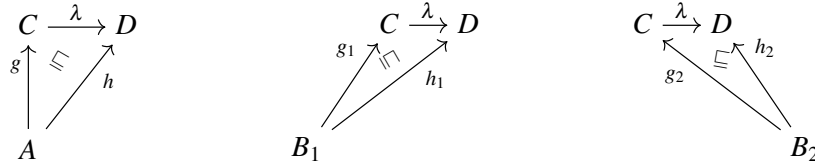
$$\begin{array}{ccccc}
 & & D & & \\
 & h_1 \nearrow & \uparrow h & \nwarrow h_2 & \\
 B_1 & \sqsubseteq & & \sqsupseteq & B_2 \\
 & f_1 \nwarrow & \downarrow & \nearrow f_2 & \\
 & & A & &
 \end{array}$$

semicommutates, it holds that the univariate system

$$g; \lambda \sqsubseteq h \qquad g_1; \lambda \sqsubseteq h_1 \qquad g_2; \lambda \sqsubseteq h_2$$

of morphism equations has a maximum solution for the indeterminate λ . \dashv

The formulation given in [Definition 4.17](#) for presenting $\frac{3}{2}$ -pushouts exhibits an obvious relationship with the one given in [Definition 4.5](#); the main difference is that instead of looking for unique solutions to a family of morphism equations one looks for best (i.e., largest) solutions to a family of morphism inequations. For the particular inequations given in [Definition 4.17](#), the family of morphism inequations is the one saying that the three triangles



semicommute.

It is worth saying that whenever the category \mathbf{C} has ordered coproducts (in the sense of [Definition 4.14](#)) the system $\{c_X; f \sqsubseteq d_X \mid X \in \text{Node}\}$ of morphism inequations (that is, the one which appears in [Definition 4.16](#)) is equivalent to the following single inequation: $(\bigoplus \{c_X \mid X \in \text{Node}\}); f \sqsubseteq \bigoplus \{d_X \mid X \in \text{Node}\}$.

Let us assume now that c is a $\frac{3}{2}$ -colimit (with object C) over a lax diagram \mathcal{D} and that $f \in \text{hom}(C, D)$. Then, by monotonicity it holds that $c; f$ is also a lax cocone (with object D). Therefore, by definition of $\frac{3}{2}$ -colimit the univariate inequational system $c; \lambda \sqsubseteq^* c; f$ has a maximum solution for λ . In other words, the inequational system

$$c_X; \lambda \sqsubseteq d_X; f \quad (\text{for every node } X)$$

has a maximum solution for λ . We denote such a maximum solution g . Considering that f is also trivially a solution to the very system, we obtain that $f \sqsubseteq g$. Thus, by monotonicity it must hold that $c_X; f \sqsubseteq c_X; g$ for every node X . Therefore, g is also the largest solution to the equational system $c; \lambda = c; f$.

Concludingly, we have demonstrated that for every $\frac{3}{2}$ -colimit c (with object C) over a lax diagram \mathcal{D} and every $f \in \text{hom}(C, -)$, there exists $\max_{\sqsubseteq} \{g \mid H_c(g) = H_c(f)\}$ that coincides with $\max_{\sqsubseteq} \{g \mid H_c(g) \sqsubseteq H_c(f)\}$. Thus, for every $\frac{3}{2}$ -colimit c over a lax diagram \mathcal{D} , we can define the *expansion* function

$$\begin{aligned}
 \text{xpan}_c : \text{hom}(\text{apex}(c), -) &\longrightarrow \text{hom}(\text{apex}(c), -) \\
 f &\longmapsto \text{xpan}_c(f) := \max_{\sqsubseteq} \{g \mid H_c(g) = H_c(f)\} = \\
 &\quad \max_{\sqsubseteq} \{g \mid H_c(g) \sqsubseteq H_c(f)\}
 \end{aligned}$$

It is obvious that $H_c(f) = H_c(\text{xpan}_c(f))$. Moreover, this function xpan_c is

- extensive, i.e., $f \sqsubseteq \text{xpan}_c(f)$;
- increasing, i.e., if $f_1 \sqsubseteq f_2$ then $\text{xpan}_c(f_1) \sqsubseteq \text{xpan}_c(f_2)$;
- idempotent, i.e., $\text{xpan}_c(\text{xpan}_c(f)) = \text{xpan}_c(f)$.

Consequently, every $\frac{3}{2}$ -colimit c induces a closure operator (or closure system) [8, Section I.5] on the set $\text{hom}(\text{apex}(c), -)$.

On Page 32 we point out that colimits are jointly epimorphic. Unfortunately, in the arbitrary case²⁵ it is not so clear whether this property also holds for $\frac{3}{2}$ -colimits. However, as obvious from the definitions of xpan_c , it holds that

$$\text{if } h_1 \text{ and } h_2 \text{ satisfy that } H_c(h_1) = H_c(h_2), \text{ then } \text{xpan}_c(h_1) = \text{xpan}_c(h_2).$$

In other words, the following property (which resembles the definition of jointly epimorphic) holds for $\frac{3}{2}$ -colimits c :

$$\text{if } h_1 \text{ and } h_2 \text{ satisfy that 'c}_X; h_1 = c_X; h_2 \text{ for every node } X\text{'}, \text{ then } \text{xpan}_c(h_1) = \text{xpan}_c(h_2).$$

It is worth noticing that $\text{xpan}_c(h_1) = \text{xpan}_c(h_2)$ implies in particular that h_1 and h_2 are compatible.

Goguen's proposal is to use $\frac{3}{2}$ -pushouts as a computational method for finding blendings. In the easiest case (i.e., the blending of two concepts), this framework assumes that we have previously chosen

- a morphism f_1 from the generic space G into the first conceptual space I_1 (i.e. $f_1 : G \rightarrow I_1$), and also
- a morphism f_2 from the generic space G into the second conceptual space I_2 (i.e. $f_2 : G \rightarrow I_2$).

Furthermore, Goguen suggests to consider all $\frac{3}{2}$ -pushouts of the span $I_1 \xleftarrow{f_1} G \xrightarrow{f_2} I_2$ as candidates for blending of the two initial concepts. In the examples provided in [24]²⁶ this is done using ordered categories whose objects are algebraic theories (using the formal specification language OBJ), morphisms correspond to partial functions preserving the structure, and the partial order corresponds to being an extension.

There are several difficulties in order to provide a computational framework to blending following Goguen's categorical proposal. Some of them are as follows.

- While there are several available software packages for dealing with “algebraic theory” categories and colimits (like Hets [55, 12]) this is not the case in the context of ordered categories.

²⁵ Notice that in Appendix A it is shown that in **Pfn** it really holds.

²⁶ It is also worth looking at <http://cseweb.ucsd.edu/~goguen/papers/blend.html> because there are more recent examples.

- Although [24] contains a first theoretical study of $\frac{3}{2}$ -colimits, the theoretical framework still needs to be improved before considering computational implementations. For example, can we characterise all $\frac{3}{2}$ -pushouts in the ordered category **Pfn**? What about more complex diagrams that are still in **Pfn**? What about considering other well-known ordered categories? Can we get rid of the ordered category **C** appealing to some particular plain category built from **C**?

We hope to contribute to the second point in a near future with research done inside the COIN-VENT project, but currently most of these questions remain open. In [Appendix A](#) we completely characterise $\frac{3}{2}$ -colimits in **Pfn**.

Related Works. The papers [42] and [43] use Goguen’s categorical framework, but without ordered categories, i.e., only plain categories are considered.

The proposed framework uses the category of CASL theories, which is known to be cocomplete [54], and whose computation of colimits is supported in Hets.²⁷ Besides this, the authors of [42, 43] also advocate for using the distributed ontology language DOL as a metalanguage for specifying categorical diagrams (i.e., families of morphisms). When computing colimits, they point out (indeed Goguen already did) that in some case it might be interesting (for blending purposes) to ignore some of the morphisms in the diagram, and considering them just as auxiliary morphisms.

A crucial difference between [42] and [43] is, that in [43] the authors only focus on input diagrams given by total functions, while in the previous version [42] the same authors consider a more general setting allowing for partial morphisms. This simplification has deep consequences, because the colimits of diagrams formed by total functions are, in most cases, although computed in categories of partial morphisms, formed only by total functions (see [Page 34](#)).

4.2 Blending as Amalgams

An amalgam is a description that combines parts of two other descriptions as a new coherent whole. There are notions that are related to amalgams in addition to conceptual blending, notions such as merging operation or information fusion. They all have in common that they deal with combining information from more than one ‘source’ into a new integrated and coherent whole; their differences reside on the assumptions they make on the sources characteristics and the way in which the combination of the sources take place.

The notion of amalgams was developed in the context of Case-based Reasoning (CBR) [57], where new problems are solved based on previously solved problems (or cases, residing on a case base). Solving a new problem often requires more than one case from the case base, so their content has to be combined in some way to solve the new problem. The notion of amalgam of two cases (two descriptions of problems and their solutions, or situations and their outcomes) is a proposal to formalise this process of the ways in which they can be combined to produce a new, coherent case.

²⁷ Colimits are available in Hets without problems in the homogeneous case of reasonable institutions (which include most cases: first-order logic, description logics, etc), but things are not so simple in the heterogeneous case; for such a case only the colimits of certain diagrams (the ‘connected thin inf-bounded’ ones) [13] are computed.

Formally, the notion of amalgams can be defined in any representation language \mathcal{L} for which a subsumption relation \sqsubseteq between the terms (or descriptions) of \mathcal{L} can be defined. We say that a term ψ_1 subsumes another term ψ_2 ($\psi_1 \sqsubseteq \psi_2$) when ψ_1 is more general (or equal) than ψ_2 ²⁸. Additionally, we assume that \mathcal{L} contains the infimum element \perp (or ‘any’), and the supremum element \top (or ‘none’) with respect to the subsumption order.

Next, for any two terms ψ_1 and ψ_2 we can define their *unification*, $(\psi_1 \sqcup \psi_2)$, which is the *most general specialisation* of two given terms, and their *anti-unification*, defined as the *least general generalisation* of two terms, representing the most specific term that subsumes both. Intuitively, a unifier (if it exists) is a term that has all the information in both the original terms, and an anti-unifier is a term that contains only all that is common between two terms. Also, notice that, depending on \mathcal{L} , anti-unifier and unifier might be unique or not.

4.2.1 Amalgams

The notion of *amalgam* can be conceived of as a generalisation of the notion of unification over terms. The unification of two terms (or descriptions) ψ_a and ψ_b is a new term $\phi = \psi_a \sqcup \psi_b$, called unifier. All that is true for ψ_a or ψ_b is also true for ϕ ; e.g. if ψ_a describes ‘a red vehicle’ and ψ_b describes ‘a German minivan’ then their unification yields the description ‘a red German minivan.’ Two terms are not unifiable when they possess contradictory information; for instance ‘a red French vehicle’ is not unifiable with ‘a blue German minivan’. The strict definition of unification means that any two descriptions with only one item with contradictory information cannot be unified.

An *amalgam* of two terms (or descriptions) is a new term that contains *parts from these two terms*. For instance, an amalgam of ‘a red French vehicle’ and ‘a blue German minivan’ is ‘a red German minivan’; clearly there are always multiple possibilities for amalgams, since ‘a blue French minivan’ is another example of amalgam. The notion of amalgam, as a form of ‘partial unification’, was formally introduced in [57].

For the purposes of this paper, we will introduce a few necessary concepts.

Definition 4.18 (Amalgam). The set of *amalgams* of two terms ψ_a and ψ_b is the set of terms such that:

$$\psi_a \curlyvee \psi_b = \{\phi \in \mathcal{L} \setminus \{\top\} \mid \exists \alpha_a, \alpha_b \in \mathcal{L} : \alpha_a \sqsubseteq \psi_a \wedge \alpha_b \sqsubseteq \psi_b \wedge \phi = \alpha_a \sqcup \alpha_b\} \quad \dashv$$

Thus, an amalgam of two terms ψ_a and ψ_b is a term that has been formed by unifying two generalisations α_a and α_b (whenever this unification is not inconsistent, i.e. $\alpha_a \sqcup \alpha_b \neq \top$) such that $\alpha_a \sqsubseteq \psi_a$ and $\alpha_b \sqsubseteq \psi_b$ —i.e. an amalgam is a term resulting from combining some of the information in ψ_a with some of the information from ψ_b . Formally, $\psi_a \curlyvee \psi_b$ denotes the set of all possible amalgams; however, whenever it does not lead to confusion, we will use $\psi_a \curlyvee \psi_b$ to denote one specific amalgam of ψ_a and ψ_b .

In [57] a slightly different definition of amalgam is given, for which not all generalisations are taken into account, only those that are less general than $\psi_a \sqcap \psi_b$ (the anti-unification of the inputs). We rephrase this definition here introducing the notion of *bounded amalgam*:

²⁸In machine learning terms, $A \sqsubseteq B$ means that A is more general than B , while in description logics it has the opposite meaning, since it is seen as ‘set inclusion’ of their interpretations.

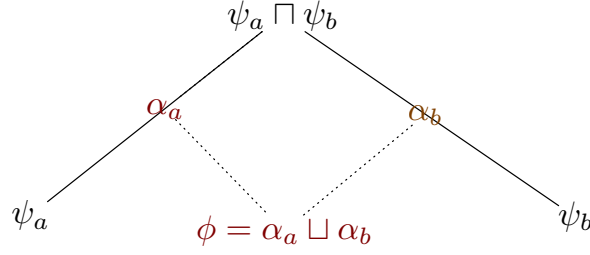


Figure 15: A diagram of an amalgam ϕ from inputs ψ_a and ψ_b where $\chi = \alpha_a \sqcap \alpha_b$.

Definition 4.19 (Bounded amalgam). Let $\chi \in \mathcal{L}$. The set of χ -bounded amalgams of two terms ψ_a and ψ_b is the set of terms such that:

$$\psi_a \curlyvee_{\chi} \psi_b = \{\phi \in \mathcal{L} \setminus \{\top\} \mid \exists \alpha_a, \alpha_b \in \mathcal{L} : \chi \sqsubseteq \alpha_a \sqsubseteq \psi_a \wedge \chi \sqsubseteq \alpha_b \sqsubseteq \psi_b \wedge \phi = \alpha_a \sqcup \alpha_b\} \quad \dashv$$

A particularly interesting case (the one studied in [57]) is when $\chi = \psi_a \sqcap \psi_b$, the anti-unification of the inputs, as illustrated in Figure 15. The intuitive reason is that the anti-unification represents what is common or shared between the two inputs and, thus, generalising more than $\psi_a \sqcap \psi_b$ would eliminate information that is already in both inputs and is compatible.

The terms α_a and α_b are called the *transfers* or *constituents* of an amalgam $\psi_a \curlyvee \psi_b$. α_a represents all the information from ψ_a which is *transferred* to the amalgam, and α_b is all the information from ψ_b which is transferred into the amalgam. As we will see later, this idea of transfer is akin to the idea of *transferring* knowledge from the source to target in CBR, and also in computational analogy [17].

Usually we are interested only on maximal amalgams of two input terms, i.e., those amalgams that contain maximal parts of their inputs that can be unified into a new coherent description. Formally, an amalgam $\phi \in \psi_a \curlyvee \psi_b$ is maximal if there is no $\phi' \in \psi_a \curlyvee \psi_b$ such that $\phi \sqsubset \phi'$. In other words, if more properties of an input were added the combination would be no longer consistent. The reason what we are interested in maximal amalgams is very simple: consider an amalgam ϕ' such that $\phi' \sqsubset \phi$; clearly ϕ' , being more general than ϕ , has less information than ϕ and thus combines less information from the inputs ψ_a and ψ_b . Since ϕ has more information while being consistent, ϕ' or any amalgam that is a generalisation of ϕ , are trivially derived from ϕ by generalisation.

Definition 4.20 (Asymmetric amalgam). The χ -bounded *asymmetric amalgams* $\psi_s \xrightarrow{\chi} \psi_t$ of two terms ψ_s (*source*) and ψ_t (*target*) is the set of terms such that:

$$\psi_s \xrightarrow{\chi} \psi_t = \{\phi \in \mathcal{L}^+ \mid \exists \alpha_s \in \mathcal{L} : \chi \sqsubseteq \alpha_s \sqsubseteq \psi_s \wedge \phi = \alpha_s \sqcup \psi_t\} \quad \dashv$$

In an asymmetric amalgam, the target term is transferred completely into the amalgam, while the source term is generalised. The result is a form of partial unification that conserves all the information in ψ_t while relaxing ψ_s by generalisation and then unifying one of those more general terms with ψ_t itself. As before, we would be usually interested only on the asymmetric amalgams that are maximal.

This model of analogy as asymmetric amalgam can be used to model the case-based inference in CBR, as explained in [58]. Essentially, this model clarifies what knowledge is transferred from source description to target, namely the transfer term α_s is what case-based inference conjectures is applicable (is constant with) the target. In the case of a maximal amalgam, α_s represents as much information as can be transferred from the source to the target ψ_t such that $\alpha_s \sqcup \psi_t$ is consistent.

4.2.2 Comparing Amalgams and Blends

Amalgams and blends are similar in that they combine (parts of) two inputs into a new coherent whole. They diverge however on the assumptions they make about those inputs. Amalgams approach the problem of combining inputs from the viewpoint of Artificial Intelligence, assuming some representation language is used to specify these inputs (that can be understood as cases, problems, situations, depending on the context). Although amalgams makes very few demands on the representation language (only that some kind of subsumption/generalisation relation can be defined among the formulas of the language), the fact is that it takes very syntactic approach on what the inputs are and what consistency is (since consistency is itself implicitly defined by the way a given language defines unification).

Conceptual integration/blending, on the other hand, coming from cognitive psychology, considers blends as a combination of inputs conceived as *mental spaces*. Mental spaces have been described as ‘*small conceptual packets constructed as we think and talk, for purposes of local understanding and action*’ by Fauconnier and Turner [22]. The nature of the assumptions made about what the inputs are, therefore, quite disparate. Although we make the additional hypothesis that mental spaces can be formalised as a formula in a representation language, the properties that should be preserved in that formalisation remains an open research issue. Probably, a good formalisation of mental spaces would add more constraints to the syntactic notions of consistency in unification and amalgams.

As an example of this issue, we present now an characterisation mental spaces in which a situation, syntactically described with a formula in a language, is augmented by interpreting it using an image schema (e.g. the CONTAINER schema). In other words, given an input situation $\psi \in \mathcal{L}$, and a library of image schemas, we select an image schema η with which to interpret ψ . This interpretation generates new knowledge about the input, so the augmented input can be seen as $\psi \sqcup \eta$, assuming image schemas are represented in the same representation language.

Let us revise the notion of amalgam when we also have a library of image schemas. An amalgam, such as introduced above can be characterised as an amalgam tuple $\langle \psi_a, \psi_b, \phi, \alpha_a, \alpha_b \rangle$, as shown in Figure 15. Now, let us assume that we will use two image schemas η_a and η_b , where a term η can be a single image schema or a particular combination of image schemas to be used in this interpretation. The addition of an image schema based interpretation is defined as follows:

Input Spaces $\bar{\psi}_a = \psi_a \sqcup \eta_a$ and $\bar{\psi}_b = \psi_b \sqcup \eta_b$

Antiunification $\bar{\psi}_a \sqcap \bar{\psi}_b$

Amalgam $\bar{\phi} \in \bar{\psi}_a \vee \bar{\psi}_b$ with constituent terms $\bar{\phi} = \bar{\alpha}_a \sqcup \bar{\alpha}_b$

Constituents $\bar{\alpha}_i = \alpha_i \sqcup \eta'_i$ where $\eta'_i \sqsubseteq \eta_i$, for $i \in \{a, b\}$

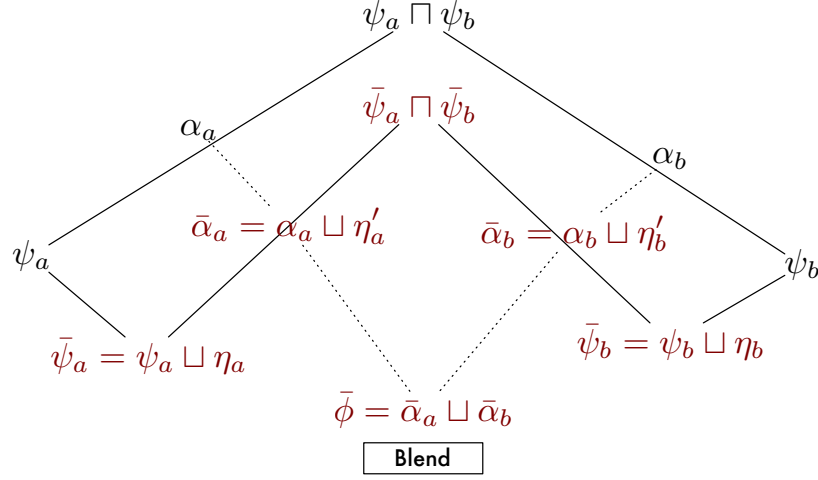


Figure 16: A diagram of a blend $\bar{\phi}$ based on amalgam ϕ from inputs ψ_a and ψ_b and image schemas η_a and η_b where $\chi = \alpha_a \sqcap \alpha_b$.

See Figure 16 for a diagram of the relations among this terms and compare it to the unrefined amalgam of Figure 15.

In this approach, a blend $\bar{\phi}$ seen as an amalgam plus an image schema based interpretation is defined as follows:

$$\bar{\phi} = \bar{\alpha}_a \sqcup \bar{\alpha}_b = \alpha_a \sqcup \alpha_b \sqcup \eta'_a \sqcup \eta'_b = \phi \sqcup \eta'_a \sqcup \eta'_b$$

Thus, the amalgam now has new content based on the properties contributed by the image schemas being used in a particular interpretation of the situation inputs. The inputs augmented with the image schemas are intense to model the ‘mental spaces’ of conceptual integration/blending and, in that sense, the result is a blend of the input spaces $\bar{\psi}_a$ and $\bar{\psi}_b$.

A short example may be helpful in clarifying these ideas.

4.2.3 An Example: Computer Icons

Computer icons are a graphical semiotic system used in GUIs for different purposes in human-computer interaction: identification, selection, action, etc. From our experience in using icons, we humans learnt that some patterns can be interpreted as signs that have a (contextual) meaning: therefore icons can be seen as a semiotic system. Consider the two icons²⁹ shown in Figure 17: a) may be interpreted as sign of a music mobile device or smartphone (i.e. identifying a class of entities), while b) may be interpreted as a sign representing data synchronisation (i.e. identifying a class of processes).

Existing icon images can be analysed by image understanding techniques. In this work on icon blending we select a technique that analyses images to produce a qualitative model of that

²⁹These icons have been adapted, for our purposes here, from icons in an existing library of icons such as <http://www.iconshock.com/flat-icons>.

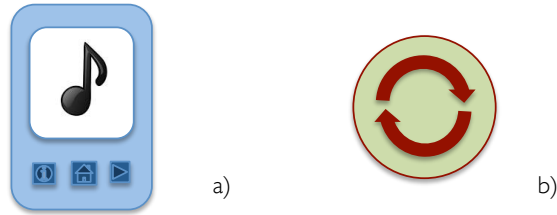


Figure 17: Two examples of computer icons: a) music mobile device and b) data synchronisation.



Figure 18: An icon example built as an amalgam of the two icons in Figure 17.

image that can be described in natural language³⁰. The result of the analysis is a formula relating the component objects using a propositional vocabulary of qualitative spatial relations; there are relative position predicates (e.g. $\text{Above}(x,y)$ or $\text{LeftOf}(y,z)$) and general position predicates (e.g. an object is situated on the upper middle half, or is centered). Moreover the objects being recognised are assigned classes, that can be either generic geometric classes (e.g. ‘octagon’) or pertain to a domain-dependent vocabulary (e.g. ‘arrow’, or ‘screen’).

Applying an image schema, such as the CONTAINER schema, means that we interpret the inputs, e.g., the icons in Figure 17, adding the predicates that characterise the CONTAINER schema. For instance, we can say that icon a) is an octagon that contains three squares (buttons) and an octagon (screen) that contains a mixed-shape object (quaver). The application of an image schema, as a high level cognitive pattern, seems to imply also that the predicates belonging to the schema are more significant than the other —i.e. predicate $\text{Contains}(x,y)$ seems to be more significant than $\text{Above}(x,y)$. Consequently, the blends that include more images schema predicates should be preferred to others that do not³¹.

Now consider the icon shown in Figure 18 built as an amalgam of the two icons in Figure 17: a mobile device that shows in its screen that it is synchronising data. Form the point of view of the CONTAINER schema the new icon is represented as an octagon that contains three squares (buttons) and an octagon (screen) that contains two curved arrows (sync sign). The meaning or interpretation of the new icon seems straightforward: a mobile device in the process of data synchronisation. Clearly, this is one of many possible amalgams (or blends in conceptual blending), but the process of creating it is straightforward, as shown in Figure 19.

³⁰This unpublished work on icon analysis and blending is done in cooperation with Lledó Museros Cabedo and Ismael Sanz, from Jaume I University, developers of the image understanding technique [18].

³¹At this point, however, we have not incorporated any mechanism for taking into account this preference but is scheduled to be analysed as future work.

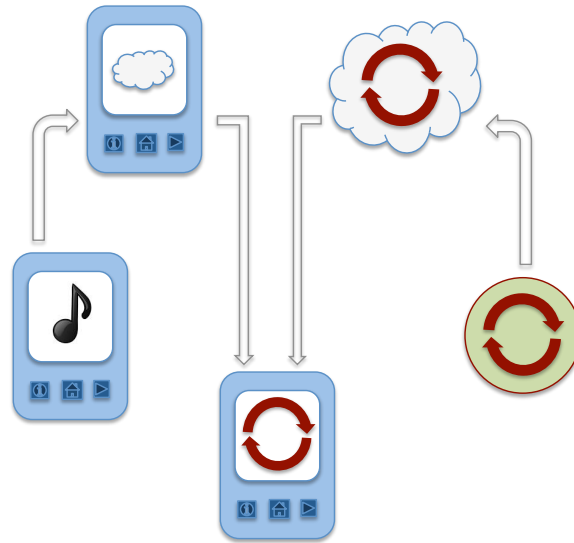


Figure 19: An example of the process creating the amalgam icon of Figure 18 from the two input icons. The ‘cloud’ indicates the parts that are abstracted away by generalisation (upward arrows); the two downward arrows indicate the unification of the two generalisations creating the final amalgam.

The two inputs augmented with the CONTAINER schema are generalised in a very simple way. The icon a) has generalised the mixed-shape object (quaver) to an object without any particular form (shown as a cloud in Figure 19), while the icon b) has generalised the circle into an object without any particular form (shown as a cloud). These two generalisations are the *constituents* of the amalgam in Figure 19. This amalgam is built unifying the constituents, a process that binds the formless object contained the screen with the arrows contained in the formless object of icon b). The result is an icon that contains the two arrows in the screen contained in the octagon that forms the icon; since with the CONTAINER schema considers the ‘contains’ relation transitive, the new icon contains the two arrows, since it contains the screen that contains the arrows.

This added information in the mental spaces can be considered equivalent to the notion of *completion* that builds the *emergent structure* in Conceptual Blending Theory. Recall that the emergent structure is informational structure added to the blend and that does not come from the intuit spaces, and completion is a process that uses a person background knowledge to infer new information that is relevant to or needed in the blend. The main difference is that in Conceptual Blending Theory, the way it is explained, seems this process takes place *after* the selective projection of the input spaces into the blend spaces. However, since Conceptual Blending Theory does not offer a computational model, we think this temporality is a matter of exposition, and does not imply a temporality order in computational processes. Thus, in general terms, our proposal of augmenting inputs and amalgams with image schemas is compatible with the notion of a completion process that adds relevant knowledge from what we call a *rich background*—containing image schemas and past example episodes of the same domain (in this scenario, previous icons and interpretations of their meaning).

Other possible amalgams or blends are possible, depending on the generalisation steps taken,

but only some of the syntactically legal combinations seem ‘meaningful’ to us. For instance, another amalgam may be that the object sign contained one of the buttons of icon a) is generalised away, and the new blended icon has the two arrows contained in a button; this amalgam can have again a straightforward interpretation: a button that controls data synchronisation. However, other possible amalgams have no worthy interpretation: it is possible to generalise a way the arrows in icon b) and we have then a green circle that can be blended into objects contained in the icon a) like the screen or one button. These less meaningful combinations are no less amalgams or blends, they are simply less interesting or significant with respect to some context-dependent optimality criteria.

4.3 Relating Colimits and Amalgams

In Section 4.1 we mention that it is very appealing to model blending as a colimit in some category \mathbf{C} of conceptual spaces and their structure-preserving mappings. When blending two input spaces, however, not everything is included into the blend because there may be incompatibilities between the input spaces. In general, conceptual blending is based on selective projections from the input spaces into the blend (see Section 2.2).

Consequently, the classical colimit construct in \mathbf{C} is inadequate for modelling blending. Goguen suggested $\frac{3}{2}$ -colimits in ordered categories instead, where structure-preserving mappings between conceptual spaces are based on partial functions. We discuss this approach thoroughly in Section 4.1.2.

In Definition 4.11 we introduce several alternative ways in which selective projections can be modelled categorically, without getting into the subtlety of dealing with partial morphisms and an order between them. In this section we shall focus on $\mathbf{MSpan}(\mathbf{C})$ — the category of mono spans in \mathbf{C} — and show that the cocone constructs in $\mathbf{MSpan}(\mathbf{C})$ can be seen as an abstraction, into the category-theoretical setting, of amalgams as introduced in Section 4.2.1. Furthermore, this construct might be also suitable for modelling and computing conceptual blends, as we shall illustrate in Section 5. First, however, we recall some basic notions of category theory not introduced in Section 4.1 that we are going to need in this section, and we introduce also some additional notation.

4.3.1 Preliminaries

Let \mathbf{C} be a category and $f: A \rightarrow C$ be a morphism in \mathbf{C} . We say that f *factors* through some morphism $g: B \rightarrow C$ if there exists $h: A \rightarrow B$ such that $f = h;g$. If g is a monomorphism, then h is the pullback of f along g . Following there is a proof of this claim: Let $m: D \rightarrow A$ and $n: D \rightarrow B$ such that $m;f = n;g$. The morphism m is also the unique morphism from D to the apex A of the pullback such that $m;id_A = m$ and $m;h = n$. The first equality is trivial. For the second, we know that $m;f = n;g$ and $f = h;g$, consequently $m;h;g = n;g$. But g is a monomorphism, so $m;h = n$. And if k is any other morphism from D to the apex A satisfying these properties we would have that $k;id = m$, hence $k = m$.

Let $A \xrightarrow{f} C \xleftarrow{g} B$ be a diagram in \mathbf{C} . If there is a pullback over this diagram we shall write \tilde{f} for the pullback of morphism f along morphism g .

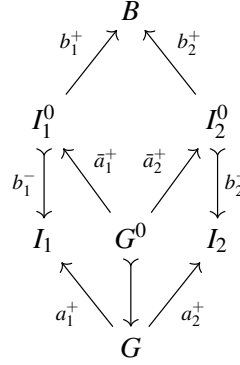


Figure 20: Representation in \mathbf{C} of a cocone in $\mathbf{MSpan}(\mathbf{C})$ over $I_1 \xleftarrow{a_1} G \xrightarrow{a_2} I_2$

Remember from [Definition 4.11](#) that a morphism $f : A \rightarrow B$ in $\mathbf{MSpan}(\mathbf{C})$ is in particular a span in \mathbf{C} . We will represent this span in \mathbf{C} with $A \xleftarrow{f^-} A^0 \xrightarrow{f^+} B$. Recall that f^- is a monomorphism, i.e., the span is a mono span.

4.3.2 A Category-Theoretical Account of Amalgams

A poset $\langle \mathcal{L}, \sqsubseteq \rangle$ as the one considered in [Section 4.2](#) can be seen as a category such that objects are the elements of \mathcal{L} , and there is a unique morphism from ϕ to ψ whenever $\phi \sqsubseteq \psi$. Consequently, we can propose a category-theoretical account of the notion of amalgam as given in [Definitions 4.18](#) and [4.19](#).

Let \mathbf{C} be a category and let A be an object in \mathbf{C} . We will say that the *generalisations* of A are all monomorphisms with target A . Let $f : A \rightarrow B$ be a morphism in \mathbf{C} . We will say that the *f-bounded generalisations* of A are all monomorphisms $g : C \rightarrow B$ such that f factors through g .

Now, let \mathbf{C} be a category with pullbacks, and let $I_1 \xleftarrow{a_1} G \xrightarrow{a_2} I_2$ be a V-shaped diagram in the bicategory $\mathbf{MSpan}(\mathbf{C})$ such that $a_1^- = a_2^- = id_G$. (Note that we can see it also as a V-shaped diagram $I_1 \xleftarrow{a_1^+} G \xrightarrow{a_2^+} I_2$ in \mathbf{C} .) Recall that for $I_1 \xleftarrow{b_1} B \xrightarrow{b_2} I_2$ to be a cocone over this V-shaped diagram in $\mathbf{MSpan}(\mathbf{C})$ we need that $a_1; b_1 \simeq a_2; b_2$. This amounts to say that, in the \mathbf{C} -diagram of [Figure 20](#), the pullbacks of $I_i^0 \xleftarrow{b_i^-} I_i \xrightarrow{a_i} G$ are isomorphic (G^0 denotes the apex of these isomorphic objects, without loss of generality), and $\bar{a}_1^+; b_1^+ = \bar{a}_2^+; b_2^+$. This brings us to the categorical notion of amalgam.

Definition 4.21 (Amalgam). Let $a_1^+ : G \rightarrow I_1$ and $a_2^+ : G \rightarrow I_2$ be two morphisms in a category \mathbf{C} with pullbacks. An *amalgam* $\langle b_1^+, b_2^+ \rangle$ of a_1^+ and a_2^+ is a cocone with apex B over

$I_1^0 \xleftarrow{\bar{a}_1^+} G^0 \xrightarrow{\bar{a}_2^+} I_2^0$, where \bar{a}_i^+ are the pullbacks of a_i^+ along generalisations $b_i^- : I_i^0 \rightarrow I_i$ of I_i (for $i \in \{1, 2\}$), such that G^0 is the common (up to isomorphism) apex of these pullbacks (see [Figure 20](#)). \dashv

In the particular case when \mathbf{C} is the poset $\langle \mathcal{L}, \sqsubseteq \rangle$ of [Section 4.2](#) the definition above amounts to [Definition 4.18](#) (taking as G the infimum element \perp). If we focus on a_i -bounded generalisations of I_i instead, we get [Definition 4.19](#), where G plays the role of the element χ . This is so because in this case the apex G^0 of the pullback is isomorphic to G .

[Definition 4.21](#) provides us a way to characterise conceptual blending in a manner that is faithful to the description given by Fauconnier and Turner (see [Section 2.2](#)) and is independent of any particular choice of representation formalism for conceptual spaces and of any implementation thereof. Furthermore, the definition points to a possible way to compute blends via the classical colimit construct as implemented in Hets. The next section illustrates this with the Buddhist Monk Riddle.

5 The Theory at Work

As a proof-of-concept for our theory we provide an in-depth example where blending is used for creative problem solving. Towards this, we first state the informal description of the problem to solve, namely the Buddhist Monk Riddle. Then we provide some background information concerning our implementation machinery, and finally we present our implementation.

5.1 The Buddhist Monk Riddle

The Buddhist Monk Riddle was first related to creativity by [Koestler](#), who gives the following informal description (see also [Figure 21](#)):

“One morning, exactly at sunrise, a Buddhist monk began to climb a tall mountain. The narrow path, no more than a foot or two wide, spiralled around the mountain to a glittering temple at the summit.

The monk ascended the path at varying rates of speed, stopping many times along the way to rest and to eat the dried fruit he carried with him. He reached the temple shortly before sunset. After several days of fasting and meditation, he began his journey back along the same path, starting at sunrise and again walking at variable speeds with many pauses along the way. His average speed descending was, of course, greater than his average climbing speed.

Prove that there is a single spot along the path the monk will occupy on both trips at precisely the same time of day.” [[41](#), p. 183–184]

[Koestler](#) further states how he experienced his mental process of solving the riddle:

“I tried this and that, until I got fed up with the whole thing, but the image of the monk in his saffron robe walking up the hill kept persisting in my mind. Then a moment came when, superimposed on this image, I saw another, more transparent one, of the monk walking *down* the hill and I realised in a flash that the two figures must meet at some point some time - regardless at what speed they walk and how often each of them stops. Then I reasoned out what I already knew: whether the monk descends two or three days later comes to the same; so I was quite justified in letting him descend on the same day, in duplicate so to speak.” [[41](#), p. 184]

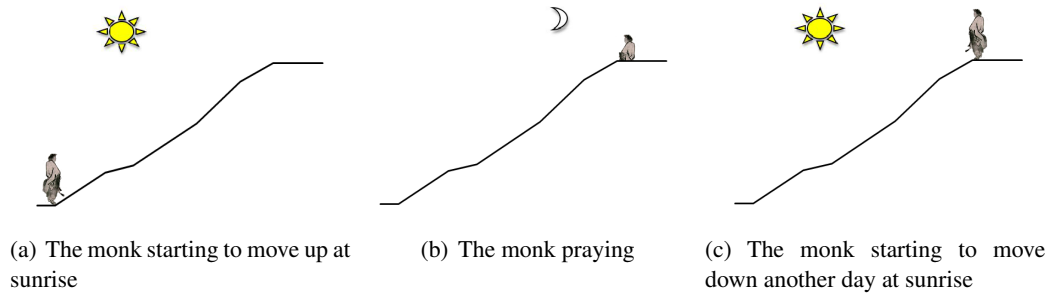


Figure 21: Illustration of the Buddhist Monk Riddle ³²

The Buddhist Monk Riddle has also been chosen by Fauconnier and Turner [22] to illustrate the constitutive elements of conceptual blending, because the superimposition that Koestler mentions can be seen as an act of blending, where the mental image of the monk walking up the mountain is overlaid with the image of the monk walking down the same mountain on the same path. The composition of those images is then completed and elaborated with *commonsense background knowledge*, i.e., knowledge that two objects that approach each other on the same path will necessarily meet at some point.

In order to illustrate our theory at work, we specify a categorical diagram that describes the conceptual integration network underlying this riddle. We further compute an amalgam as a possible solution to compose the blend. The formalisation presented here is an extension and elaboration of the one presented by Goguen [27].

Our concrete implementation of the riddle is based on formalisations of image schemas, which according to Fauconnier [19] participate in the framing of mental spaces. From an AI viewpoint, image schemas take the role of templates for data structures. In the particular example of the Buddhist monk, we build on the SOURCE-PATH-GOAL (PATH) image schema (see Section 2.4), which provides the structure to conceptualise the dynamic scene of the monk walking along a path from a source location (the foot of the mountain) to a goal location (the summit of the mountain) and back. However, before we provide the details of the implementation, in order to facilitate a better understanding, we first give a brief overview over the specification language we use, namely the Common Algebraic Specification Language (CASL) [2].

5.2 CASL and its Application for Blending

In order to represent input spaces for conceptual blending we follow Fauconnier and Turner's theory and use image schemas as structural templates for the formalisation of input spaces. Hence, for a computational model, we need an input language that is expressive enough to model image schemas, and also the complex relations between them. As an example for such relations, consider that Lakoff and Johnson [47] state that image schemas may be extended with knowledge represented through other image schemas. Hence, it is required to have a input language that allows for some form of inheritance. Inheritance is also required to model blend completion, i.e., the step of the blending process where additional background knowledge is added to the input spaces.

³²Images from <http://markturner.org/blending.html>, accessed Sept. 2014

Apart from expressiveness, the most crucial requirement for the input language we use is compatibility with theoretical model we develop in [Section 4](#). That is, we require an input language with a signature that can be understood as a cocomplete category.

Towards this, we use the Common Algebraic Specification Language (CASL) to describe input spaces. CASL is capable of expressing inheritance, and a lot of research on the categorical properties of CASL has been conducted in the past. In particular its cocompleteness property has been proven [54].

An excellent brief summary about CASL is given by Astesiano et al. [2]: “*The Common Algebraic Specification Language (CASL) is an expressive language for the formal specification of functional requirements and modular design of software. It has been designed by COFI, the international Common Framework Initiative for algebraic specification and development. It is based on a critical selection of features that have already been explored in various contexts, including subsorts, partial functions, First-order logic, and structured and architectural specifications. CASL should facilitate interoperability of many existing algebraic prototyping and verification tools.*”

In addition to its inherent support for inheritance and its categorical properties, CASL also allows one to infer emergent structure in a blend, in form of axioms, using a theorem prover. Towards this, we employ the Hets architecture [55], which is capable of translating CASL to various input languages understood by theorem provers. In [Section 5.3.5](#) we show for the specific case of the Buddhist monk how we use Hets and a theorem prover to infer additional emergent structure by emulating the elaboration of a blend via a proof.

CASL syntax. In order to understand our implementation of the Buddhist Monk Riddle it not necessary to be familiar with the deeper semantics of CASL (for details we refer to [2]). However, it will be helpful to have an intuitive understanding of the following syntactic elements:

- The pattern **spec ... end** is used to describe a theory, which consists of sorts, operators, predicates and axioms, where
 - **sorts** indicates a list of sorts;
 - **ops** indicates a list of operators that map objects of a certain sort to another sort;
 - **preds** indicates a list of predicates that map objects to Boolean values;
 - axioms are indicated by **•** symbols, possibly using variables defined via \forall and \exists quantifiers.
- The pattern “**spec** TCHILD = TPARENT **with** <mapping> **then** <defs>” states that a theory TCHILD extends a theory TPARENT and inherits all sorts, operators and predicates from TPARENT. An optional symbol mapping is defined after **with**. The definitions after **then** are additional sorts, operators, predicates and axioms. A child theory can extend more than one parent theory.
- **view** defines a morphism, i.e., in terms of CASL a symbolic mapping between sorts, operators and predicates.
- The keyword **combine** denotes the category theoretical colimit operation for views.

The category of CASL signatures. We build on the work by Mossakowski [54], who describe a category of CASL signatures. Signatures are defined as follows:

Definition 5.1 (CASL signature [54]). A CASL subsorted signature (S, F, TF, P, \leq) consists of

- a set S of *sorts*;
- a $S^* \times S$ -sorted family $F = (F_{w,s})_{w \in S^*, s \in S}$ of *function symbols*;
- a $S^* \times S$ -sorted family $TF = (TF_{w,s} \subseteq F_{w,s})_{w \in S^*, s \in S}$ of subsets indicating *total function symbols*;
- a S^* -sorted family $P = (P_w)_{w \in S^*}$ of *predicate symbols*, and
- a pre-order (i.e., a reflexive transitive relation) \leq of subsort embeddings on the set S of sorts. \dashv

Morphisms between CASL signatures are described in the following Definition 5.2.

Definition 5.2 (Morphisms between CASL signatures (adapted from [54])). Given signatures $\Sigma = (S, F, TF, P, \leq)$, $\Sigma' = (S', F', TF', P', \leq')$, a signature morphism $\sigma : \Sigma \rightarrow \Sigma'$ consists of

- a map $\sigma^S : S \rightarrow S'$
- a map $\sigma_{w,s}^F : F_{w,s} \rightarrow F'_{\sigma^{S^*}(w), \sigma^S(s)}$ for each $w \in S^*, s \in S$
- a map $\sigma_w^P : P_w \rightarrow P'_{\sigma^{S^*}(w)}$ for each $w \in S^*$

such that

- subsorting is preserved, i.e., $s_1 \leq s_2$ implies $\sigma^S(s_1) \leq \sigma^S(s_2)$ for $s_1, s_2 \in S$
- totality is preserved, i.e., $\sigma_{w,s}^F(TF_{w,s}) \subseteq TF'_{\sigma^{S^*}(w), \sigma^S(s)}$
- overloading relations are preserved, i.e., $f : w_1 \rightarrow s_1 \sim_F f : w_2 \rightarrow s_2$ implies $\sigma_{w_1, s_1}^F(f) = \sigma_{w_2, s_2}^F(f)$, and similarly for predicate symbols.³³ \dashv

Definition 5.2 directly yields the category of CASL signatures, which is commonly referred to as **CASLSig**. Theorem 3.1. in [54] states that **CASLSig** is cocomplete.

5.3 The Formalisation of the Riddle

The blend for the Buddhist Monk Riddle is based on the SOURCE-PATH-GOAL image schema (see Section 2.4), which in turn inherits structure from LINK image schema. We also include a temporal dimension to the schema to be able to talk about the trajectory being at the source or goal location of the path at certain time instances. Hence, in order to represent the SOURCE-PATH-GOAL schema, we first have to specify the LINK schema and a very simple theory of time, named TIME.

³³The symbol \sim_F denotes an overloading relation. For details we refer to [54].

The LINK schema says that for every object of sort *Link* there are exactly two objects of sort *Entity* such that they are linked.

```
spec LINK =
  sorts Link; Entity
  preds linked : Entity × Link
  ∀ l : Link, ∃!2 e : Entity
    • linked(e,l)
end
```

For the purpose of this exercise, we only specify a sort *Time* of time instances and a $<$ predicate to denote a total order of time instances for theory TIME, ignoring the axiomatisation of this theory.

```
spec TIME =
  sorts Time
  preds < : Time × Time
end
```

The SOURCE-PATH-GOAL schema—which we abbreviate to PATH—is based on the LINK schema and includes theory TIME as follows:

```
spec PATH = LINK
  with sorts Link ↦ Path, Entity ↦ Location,
    preds linked ↦ endpoint
and TIME
then sort Trajector
  preds on : Location × Path
  ops source : Trajector → Location; goal : Trajector → Location;
    startTime : Trajector → Time; endTime : Trajector → Time;
    position : Trajector × Time → Location
  ∀ p : Path; m : Trajector; t : Time
    • endpoint(source(m), p) ∧ endpoint(goal(m), p)
    • source(m) ≠ goal(m)
    • startTime(m) < endTime(m)
    • position(m, startTime(m)) = source(m) ∧ position(m, endTime(m)) = goal(m)
    • startTime(m) < t ∧ t < endTime(m) ⇒ on(position(m, t), p)
end
```

Recall that the notation “PATH = LINK **with** ...” denotes that PATH inherits all sorts, predicates, operators and axioms from LINK, using a mapping of sorts and predicates defined after the **with** keyword by \mapsto arrows. Therefore, the PATH schema describes a path as a link where the endpoints are locations. The axioms are defined within the scope of universal quantifiers over paths, trajectors and time points. The first three axioms represent commonsense knowledge that the source location and goal location are distinct endpoints of the path, and that the trajector starts the journey before ending it (in the temporal sense). The fourth and fifth axiom state the obvious relations between time and location during the trajector’s movement.

5.3.1 Input Spaces

The input spaces are two theories describing that a monk is moving up (resp. down) the mountain. To show that they are framed by the PATH image schema, they are represented as particularisations of this schema with the monks as path trajectors from sunrise to sunset.

```

spec  $I_1 = \text{PATH}$ 
then sorts Day
             TimeOfDay
    ops foot, summit : Location;
        trail : Path;
        monk : Trajector;
         $d_1$  : Day;
        sunrise, sunset : TimeOfDay;
         $\text{time} : \text{Day} \times \text{TimeOfDay} \rightarrow \text{Time}$ 
    •  $\text{endpoint}(\text{foot}, \text{trail}) \wedge \text{endpoint}(\text{summit}, \text{trail})$ 
    •  $\text{foot} \neq \text{summit}$ 
    •  $\text{source}(\text{monk}) = \text{foot} \wedge \text{goal}(\text{monk}) = \text{summit}$ 
    •  $\text{startTime}(\text{monk}) = \text{time}(d_1, \text{sunrise}) \wedge \text{endTime}(\text{monk}) = \text{time}(d_1, \text{sunset})$ 
end

```

The input space describes a situation where a monk is situated at the foot of a mountain at sunrise and at the summit of the mountain at sunset. Similarly, we define a second input space I_2 which is identical to I_1 , except that in I_2 we have a different day d_2 , where the start location is the summit and the end location is the foot of the mountain.

```

spec  $I_2 = \text{PATH}$ 
then sorts Day
             TimeOfDay
    ops foot, summit : Location;
        trail : Path;
        monk : Trajector;
         $d_2$  : Day;
        sunrise, sunset : TimeOfDay;
         $\text{time} : \text{Day} \times \text{TimeOfDay} \rightarrow \text{Time}$ 
    •  $\text{endpoint}(\text{foot}, \text{trail}) \wedge \text{endpoint}(\text{summit}, \text{trail})$ 
    •  $\text{foot} \neq \text{summit}$ 
    •  $\text{source}(\text{monk}) = \text{summit} \wedge \text{goal}(\text{monk}) = \text{foot}$ 
    •  $\text{startTime}(\text{monk}) = \text{time}(d_2, \text{sunrise}) \wedge \text{endTime}(\text{monk}) = \text{time}(d_2, \text{sunset})$ 
end

```

5.3.2 The Generic Space

According to Fauconnier and Turner [22, p. 41], “[a] generic mental space maps onto each of the inputs and contains what the inputs have in common: a moving individual and his position, a path linking foot and summit of the mountain, a day of travel, and motion in an unspecified direction.”

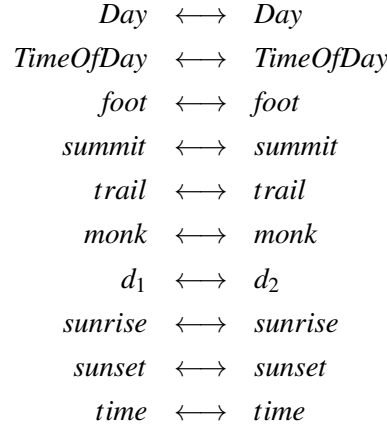


Figure 22: Cross-space mapping

A partial *cross-space mapping* [22] between the input spaces I_1 and I_2 gives hints to generate this generic space. The cross-space mapping connects counterparts in the input mental spaces, such as mountain, moving individual, day of travel, and motion (see Figure 22).

To generate the generic space for the Buddhist Monk Riddle we first pursue a naive approach and include each mapped symbol as an entity of the generic space. Furthermore, we keep only those axioms that are defined in both spaces.

```

spec  $G_{naive} = \text{PATH}$ 
then sorts  $Day;$ 
              $TimeOfDay$ 
    ops  $foot, summit : Location;$ 
         $trail : Path;$ 
         $monk : Trajectory;$ 
         $d : Day;$ 
         $sunrise, sunset : TimeOfDay;$ 
         $time : Day \times TimeOfDay \rightarrow Time$ 
    •  $endpoint(foot, trail) \wedge endpoint(summit, trail)$ 
    •  $foot \neq summit$ 
    •  $startTime(monk) = time(d, sunrise)$ 
    •  $startTime(monk) = time(d, sunrise) \wedge endTime(monk) = time(d, sunset)$ 
end

```

5.3.3 Composition of the Blend – Amalgamating the Input Spaces

A naive way to blend the input spaces would be to directly compute the colimit of the diagram formed by I_1 , I_2 and G_{naive} with morphisms mapping signature symbols of G to the counterparts in I_1 and I_2 , except for d , which is mapped to d_1 and d_2 , respectively (see Figure 23). However, this colimit computation does not keep the monks separate as in the description of the blend by

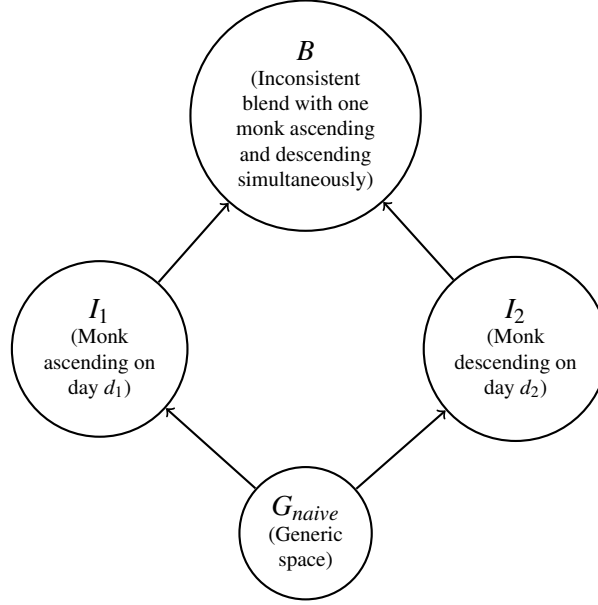


Figure 23: Naive categorical diagram of the Buddhist Monk Riddle

Fauconnier and Turner [22, p. 42]: “[...] composition of elements from the inputs makes relations available that do not exist in the separate inputs. In the blend but in neither of the inputs, there are two moving individuals instead of one. They are moving in opposite directions, starting from opposite ends of the path, and their positions can be compared at any time of the trip, since they are travelling on the same day”.

In our naive blend we get axioms $source(monk) = foot$ and $source(monk) = summit$, which together with $foot \neq summit$ form an inconsistent theory. Consequently, we need to generalise the generic space, so as to remove the entity *monk*, which generated the inconsistency in the blend. Applying a universal generalisation to the last two axioms we get the following generalised generic space:

```

spec  $G = \text{PATH}$ 
then sorts  $\text{Day};$ 
            $\text{TimeOfDay}$ 
  ops  $foot, summit : \text{Location};$ 
       $trail : \text{Path};$ 
       $d : \text{Day};$ 
       $sunrise, sunset : \text{TimeOfDay};$ 
       $time : \text{Day} \times \text{TimeOfDay} \rightarrow \text{Time}$ 
      •  $endpoint(foot, trail) \wedge endpoint(summit, trail)$ 
      •  $\forall m : \text{Trajectory}$ 
        •  $startTime(m) = time(d, sunrise)$ 
        •  $endTime(m) = time(d, sunset)$ 
end
  
```

To fully capture Fauconnier and Turner’s intuitions, we require also some sort of generalisa-

tion of the input spaces, so as to ignore the concrete day on which the monk's journey happens focussing only on the time instances of the day. These consideration lead to the more complicated diagram in Figure 24, which reflects the amalgamation cocone of Figure 20 in Section 4.3.2.

All this suggests that blending is an amalgamation where we first generalise the input spaces to relax some of their structure and then combine these relaxed spaces. In the case of the Buddhist Monk Riddle, we ignore the calendrical time of the journey and focus only on sunrise and sunset as times. This generalisation is indeed the key to solving the riddle because it implies that two monks approach each other simultaneously. We define the respective generalisations I_1^0, I_2^0 as follows:

```

spec  $I_1^0 = \text{PATH}$ 
then ops  foot, summit : Location;
           trail : Path;
           monk : Trajector;
           sunrise, sunset : Time
           • endpoint(foot, trail)  $\wedge$  endpoint(summit, trail)
           • foot  $\neq$  summit
           • source(monk) = foot  $\wedge$  goal(monk) = summit
           • startTime(monk) = sunrise  $\wedge$  endTime(monk) = sunset
end

```

Note that we generalise the input space I_1 in that we do not account anymore for the day on which the journey takes place. Instead, we just focus on time points *sunrise* and *sunset*, which are not connected anymore to a specific day. The second generalised input space I_2^0 , is analogous to I_1^0 , except that start location and end location are interchanged.

```

spec  $I_2^0 = \text{PATH}$ 
then ops  foot, summit : Location;
           trail : Path;
           monk : Trajector;
           sunrise, sunset : Time
           • endpoint(foot, trail)  $\wedge$  endpoint(summit, trail)
           • foot  $\neq$  summit
           • source(monk) = summit  $\wedge$  goal(monk) = foot
           • startTime(monk) = sunrise  $\wedge$  endTime(monk) = sunset
end

```

The morphisms from I_i^0 to I_i are the derived signature morphisms that map each symbol to its counterpart, except for *sunrise* and *sunset*, which are mapped to *time*($d_i, \text{sunrise}$) and *time*(d_i, sunset), respectively (for $i = 1, 2$).

A generalised generic space G^0 is computed with respect to each generalised input space as the pullback of the diagram $I_i^0 \rightarrow I_i \leftarrow G$, according to Definition 4.21 of amalgam.

```

spec  $G^0 = \text{PATH}$ 
then ops  foot, summit : Location;
           trail : Path;
           sunrise, sunset : Time

```

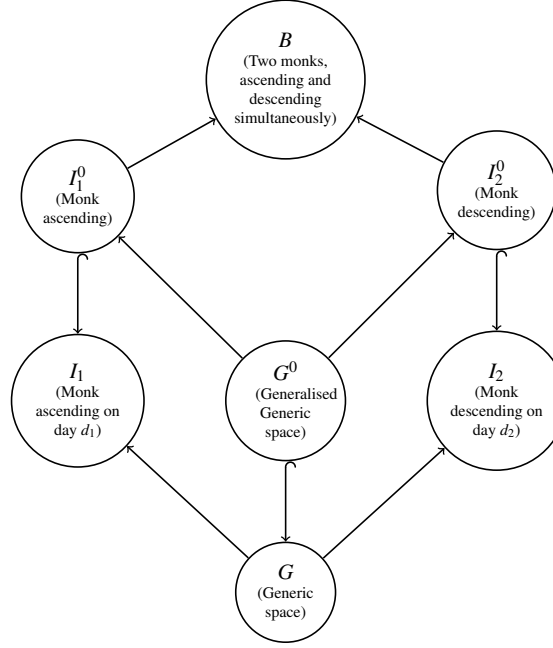


Figure 24: Categorical diagram of the Buddhist Monk Riddle amalgam

- $\text{endpoint}(\text{foot}, \text{trail}) \wedge \text{endpoint}(\text{summit}, \text{trail})$
- $\text{foot} \neq \text{summit}$
- $\forall m : \text{Trajector}$
 - $\text{startTime}(m) = \text{sunrise} \wedge \text{endTime}(m) = \text{sunset}$

end

The amalgamation is eventually completed with the combination of the (generalised) input spaces into a new conceptual space. In Hets, we use the **combine** operation to compute the colimit of two morphisms (called **views** in CASL) from G^0 to I_1^0 and from G^0 to I_2^0 .

```

view  $M_1^0 : G^0 \text{ to } I_1^0$  end
view  $M_2^0 : G^0 \text{ to } I_2^0$  end
spec BLEND_COMPOSITION = combine  $M_1^0, M_2^0$  end

```

The category theoretical diagram that represents this conceptual integration network is illustrated in Figure 24.

5.3.4 Completion of the Blend – Adding Background Knowledge

According to Fauconnier and Turner [22, p. 43], “[...] *completion brings additional structure to the blend. This structure of two people moving on the path can itself be viewed as a salient part of a familiar background frame: two people starting a journey at the same time from opposite ends of a path. [...] by means of completion, this familiar structure is recruited into the blended space. At this point, the blend is integrated: It is an instance of a particular familiar frame, the frame of two people walking on a path in opposite directions.*”

In terms of our approach, completion involves adding commonsense background knowledge about the familiar frame of people who walk in opposite directions. Of particular importance for solving the riddle is the commonsense background knowledge about the existence of a meeting point within the familiar frame. This knowledge is specified as a “MEETINGSPEACE” theory in CASL as follows:

```

spec MEETINGSPEACE = TIME
then sorts Trajector;
           Location
ops   source : Trajector → Location; goal : Trajector → Location;
       startTime : Trajector → Time; endTime : Trajector → Time;
       position : Trajector × Time → Location
       ∀ m1, m2 : Trajector
       • source(m1) = goal(m2) ∧ goal(m2) = source(m1)
         ∧ startTime(m1) = startTime(m2)
         ⇒ ∃ meetingLoc : Location; meetingTime : Time
           • position(m1, meetingTime) = meetingLoc ∧ position(m2, meetingTime) = meetingLoc
end

```

The knowledge about the meeting space is combined with the previously composed blend by simply extending the composition.

```

spec BLEND_COMPLETION = BLEND_COMPOSITION and MEETINGSPEACE end

```

Note that adding background knowledge coincides with extending image schemas, as described by Lakoff and Johnson [47, p. 27]: the authors state that image schemas are not fixed; extensions are possible and likely. In particular, the authors mention an extended SOURCE-PATH-GOAL image schema that involves several trajectors. This extended schema is a very familiar frame that allows one to immediately perceive a meeting point when two trajectors move in the opposite direction.

5.3.5 Elaboration of the Blend – Proving the Riddle

The riddle is proven by “running the blend”. According to Fauconnier and Turner [22, p. 44], “[t]his “running of the blend” is called elaboration. Running of the blend modifies it imaginatively, delivering the actual encounter of the two people. This is new structure: There is no encounter in either of the input mental spaces, even if we run them dynamically. But those two people in the blend are projected back to the “same” monk in the two input mental spaces. The meeting place projects back to the “same” location on the path in each of the inputs, and, of course, the time of day when they meet in the blend is the same as the time of day in the input spaces when the monk is at that location.”

Though Hets is not a dedicated tool to simulate dynamic narratives, we can specify narratives using PATH image schemas defined in CASL. Consequently, it allows one to draw conclusions of dynamic scenes, as a formal alternative to “mental” simulation, by proving predefined axioms. In the case of the Buddhist Monk Riddle, we want to prove that the two monks will

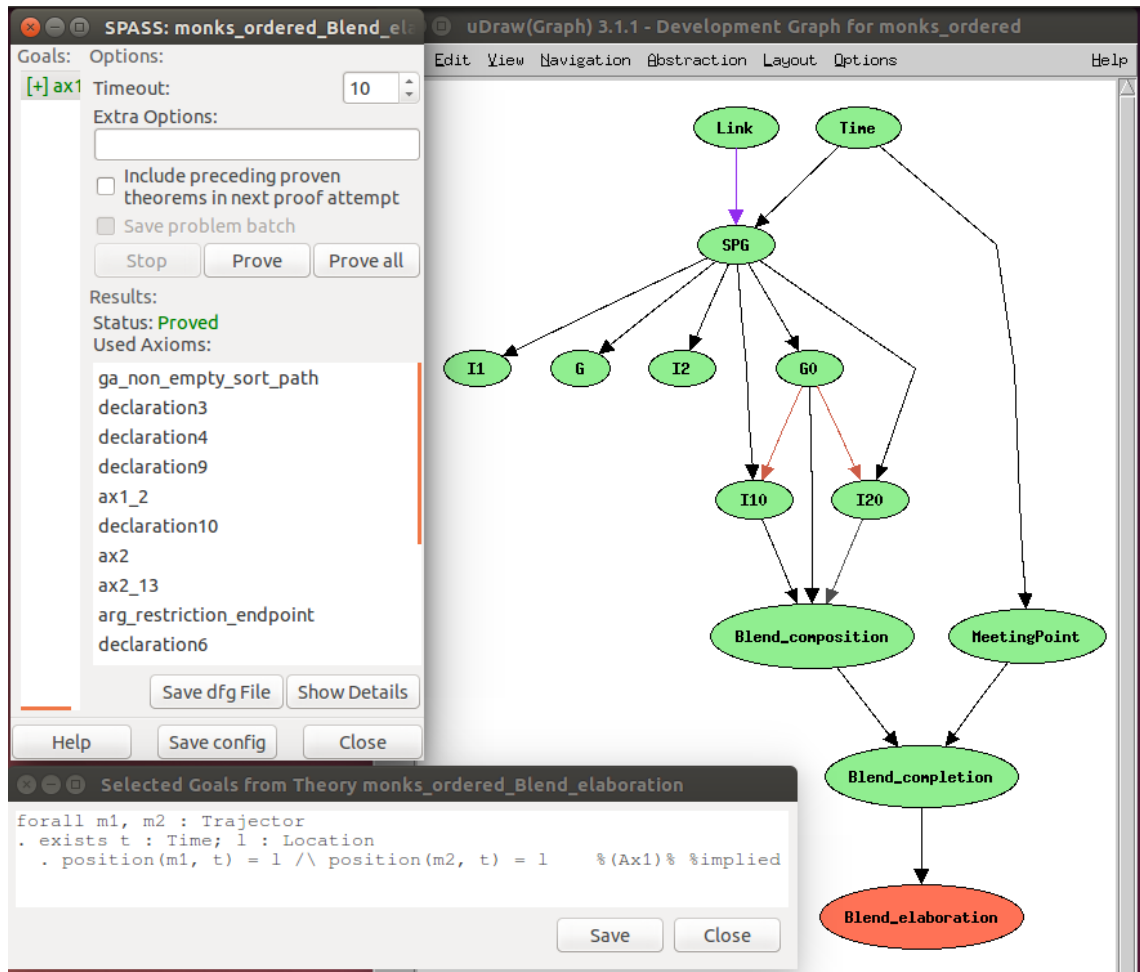


Figure 25: Specification and proof displayed in the Hets GUI

meet at a certain point. This is achieved by adding a respective axiom to the specification of BLEND_COMPOSITION, followed by the keyword **%implied**.

```

spec BLEND_ELABORATION = BLEND_COMPLETION
then  $\forall m1, m2 : \text{Trajectory}$ 
    •  $\exists t : \text{Time}; l : \text{Location}$ 
    •  $\text{position}(m1, t) = l \wedge \text{position}(m2, t) = l$  %implied
end

```

The Hets GUI allows us to graphically represent the entire theory and the proof that the axiom is correct (see Figure 25).

6 Concluding Thoughts

The theory of conceptual blending as put forward by Fauconnier and Turner in cognitive linguistics has been keenly adopted by researchers in the computing sciences for guiding the implementation of computational systems that aim at exhibiting creative capabilities, particularly when taking into consideration the creation of new concepts. In [Section 3](#) we have surveyed the most significant proposals of such systems.

As is common with these early adoptions, each system has made its own choices of interpretation of the core elements that constitute Fauconnier and Turner’s theory. They provide a formalisation of some fragment of theory that on one hand attempts to be as faithful as possible to the intuitions stated by Fauconnier and Turner, and on the other hand would be feasible to implement in a computational system.

What has become evident from these early implementations of conceptual blending is that they have been designed in a very system-specific manner, without a clear separation of system-independent issues from those that are more system-specific. This makes it difficult to gain a deeper insight into the computational aspect of conceptual blending and hence to favour the reuse of blending technology to domains other than those envisioned by the system implementors.

In this deliverable we have chosen to pursue a more domain- and system-independent approach to the development of a formal and computational theory of blending. In particular, we have taken the basic insight of Goguen that a blend might be adequately modelled as some kind of category-theoretical colimit, and in [Section 4](#) we have expounded on the details of this insight in order to fully grasp its relationship with Fauconnier and Turner’s theory.

Goguen himself proposed the framework of ordered categories to flesh out a mathematical account of conceptual blending, but he never fully worked out the implications of this proposal, nor did he show —other than with some small examples— how concrete acts of conceptual blending actually fit into his framework. The intuitions seemed convincing, but a thorough analysis was still missing. This is what we have started to do and what we have reported in this deliverable.

What has become clear of our analysis in [Section 4](#) is that dealing with Goguen’s framework is much more subtle than originally expected. His notion of $\frac{3}{2}$ -colimit as a way to model blending is quite complicated to grasp conceptually, not least to act as a guide for the implementation of computational blending systems. Although the notion of colimit is, in our view, still a powerful notion to be exploited theoretically for the purpose of giving a precise characterisation of conceptual blending, we have considered alternative ways to do so, for instance, exploiting the notion of colimit in a category of spans or of relations, a so-called allegory. The advantage of such an approach is that it nicely covers also a generalisation of the notion of amalgam, originally proposed as a method for knowledge transfer in case-based reasoning. Indeed, the notion of amalgam is very reminiscent to that of blending, and by modelling blending as colimits in a category of spans we have been capable of bringing blending and amalgamation of the same theoretical footing.

The in-depth theoretical exploration carried out in this deliverable will guide our subsequent work to carry out a computational realisation of blending that clearly distinguishes the domain-independent elements of blending such as amalgamation and colimit construction from the domain-specific realisations thereof. In [Section 5](#) we have given a glimpse of the potential of our theoretical analysis at work by specifying and implementing a well-known problem of creative thinking using the computational components we are bringing into the project.

References

- [1] ADÁMEK, J., HERRLICH, H., AND STRECKER, G. E. 2006. Abstract and concrete categories: the joy of cats. *Reprints in Theory and Applications of Categories* 17, 1–507. Reprint of the 1990 original [Wiley, New York; MR1051419].
- [2] ASTESIANO, E., BIDOIT, M., KIRCHNER, H., KRIEG-BRÜCKNER, B., MOSSES, P. D., SANNELLA, D., AND TARLECKI, A. 2002. CASL: the common algebraic specification language. *Theor. Comput. Sci.* 286, 2, 153–196. Current trends in algebraic development techniques (Lisbon, 1998).
- [3] BARR, M. AND WELLS, C. 1990. *Category theory for computing science*. Prentice Hall International Series in Computer Science. Prentice Hall International, New York.
- [4] BATEMAN, J. A., MAGNINI, B., AND FABRIS, G. 1995. The generalized upper model knowledge base: Organization and use. *Towards very large knowledge bases*, 60–72.
- [5] BESOLD, T. R., CAMBOUROPOULOS, E., GÓMEZ RAMÍREZ, D., KALIAKATSOS-PAPAKOSTAS, M., KÜHNBERGER, K.-U., MACLEAN, E., PLAZA, E., AND SMAILL, A. 2014. Specification of the representation formalism and of constraints and requirements of reasoning. Deliverable D1.1, COINVENT Project. October.
- [6] BODEN, M. A. 2004. *The Creative Mind: Myths and Mechanisms*, Second ed. Routledge.
- [7] BORCEUX, F. 1994. *Handbook of categorical algebra I. Basic category theory*. Encyclopedia of Mathematics and its Applications, vol. 50. Cambridge University Press, Cambridge.
- [8] BURRIS, S. AND SANKAPPANAVAR, H. P. 2012. *A course in Universal Algebra*, The Millennium, 2012 update ed. Electronically available at <http://www.math.uwaterloo.ca/~snburris/htdocs/ualg.html>.
- [9] CALUGAREANU, G. AND PURDEA, I. 2011. Examples in category theory. Unpublished book. Electronically available at <http://math.ubbcluj.ro/~calu/B00-0-14.pdf>.
- [10] CARBONI, A., KELLY, G. M., WALTERS, R. F. C., AND WOOD, R. J. 2008. Cartesian bicategories II. *Theory Appl. Categ.* 19, No. 6, 93–124.
- [11] CARBONI, A. AND WALTERS, R. F. C. 1987. Cartesian bicategories. I. *J. Pure Appl. Algebra* 49, 1-2, 11–32.
- [12] CODESCU, M., HOROZAL, F., KOHLHASE, M., MOSSAKOWSKI, T., RABE, F., AND SOJAKOVA, K. 2010. Towards logical frameworks in the heterogeneous tool set Hets. In *Recent Trends in Algebraic Development Techniques - 20th International Workshop, WADT 2010, Etelsen, Germany, July 1-4, 2010, Revised Selected Papers*, T. Mossakowski and H. Kreowski, Eds. Lecture Notes in Computer Science, vol. 7137. Springer, 139–159.
- [13] CODESCU, M. AND MOSSAKOWSKI, T. 2008. Heterogeneous colimits. In *Proceedings of the 2008 IEEE International Conference on Software Testing Verification and Validation Workshop*. ICSTW '08. IEEE Computer Society, Washington, DC, USA, 131–140.

- [14] DIACONESCU, R. 2008. *Institution-independent model theory*. Studies in Universal Logic. Birkhäuser Verlag, Basel.
- [15] ELIASMITH, C. 2004. Learning context sensitive logical inference in a in a neurobiological simulation. *Compositional connectionism in cognitive science*.
- [16] ELIASMITH, C. 2013. *Holographic Reduced Representation: Distributed Representation for Cognitive Structures*. Oxford University Press.
- [17] FALKENHAINER, B., FORBUS, K. D., AND GENTNER, D. 1989. The structure-mapping engine: Algorithm and examples. *Artificial Intelligence* 41, 1–63.
- [18] FALOMIR, Z., JIMÉNEZ-RUIZ, E., ESCRIG, M. T., AND CABEDO, L. M. 2011. Describing images using qualitative models and description logics. *Spatial Cognition & Computation* 11, 1, 45–74.
- [19] FAUCONNIER, G. 1985. *Mental Spaces: Aspects of Meaning Construction in Natural Language*. MIT Press.
- [20] FAUCONNIER, G. 1997. *Mappings in Thought and Language*. Cambridge University Press.
- [21] FAUCONNIER, G. AND TURNER, M. 1998. Conceptual integration networks. *Cognitive Science* 22, 2, 133–187. Reprinted in [“Cognitive Linguistics: Basic Readings” (edited by D. Geeraerts), pp. 303–371].
- [22] FAUCONNIER, G. AND TURNER, M. 2002. *The Way We Think: Conceptual Blending And The Mind’s Hidden Complexities*. Basic Books.
- [23] FREYD, P. J. AND SCEDROV, A. 1990. *Categories, allegories*. North-Holland Mathematical Library, vol. 39. North-Holland Publishing Co., Amsterdam.
- [24] GOGUEN, J. 1999. An introduction to algebraic semiotics, with application to user interface design. In *Computation for Metaphors, Analogy, and Agents*, C. L. Nehaniv, Ed. Vol. 1562. 242–291.
- [25] GOGUEN, J. 2001. 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, 298–303.
- [26] GOGUEN, J. 2004. Steps towards a design theory for virtual worlds. In *Developing Future Interactive Systems*, M. Sánchez-Segura, Ed. Idea Group Publishing, 116–152.
- [27] GOGUEN, J. 2006. Mathematical models of cognitive space and time. In *Reasoning and Cognition: Proc. of the Interdisciplinary Conference on Reasoning and Cognition*, D. Andler, Y. Ogawa, M. Okada, and S. Watanabe, Eds. Keio University Press, 125–128.
- [28] GOGUEN, J. A. 1991. A categorical manifesto. *Mathematical Structures in Computer Science* 1, 49–68.

-
- [29] GOGUEN, J. A. AND HARRELL, D. F. 2005. Foundations for active multimedia narrative: Semiotic spaces and structural blending. *Interaction Studies: Social Behaviour and Communication in Biological and Artificial Systems*.
 - [30] GOGUEN, J. A. AND HARRELL, D. F. 2010. Style: A computational and conceptual blending-based approach. In *The Structure of Style: Algorithmic Approaches to Understanding Manner and Meaning*, S. Argamon, K. Burns, and S. Dubnov, Eds. Springer, 291–316.
 - [31] GUHE, M., PEASE, A., SMAILL, A., MARTÍNEZ, M., SCHMIDT, M., GUST, H., KÜHNBERGER, K.-U., AND KRUMNACK, U. 2011. A computational account of conceptual blending in basic mathematics. *Cognitive Systems Research* 12, 3-4, 249–265.
 - [32] HAMPE, B. 2005. Image schemas in cognitive linguistics: An introduction. In *From Perception to Meaning: Image Schemas in Cognitive Linguistics*, B. Hampe and J. E. Grady, Eds. Cognitive Linguistic Research, vol. 29. Mouton de Gruyter, 1–12.
 - [33] HARRELL, D. F. 2005. Shades of computational evocation and meaning: The GRIOT system and improvisational poetry generation. *6th Digital Arts and Culture Conference*.
 - [34] HARRELL, F. 2007. Theory and technology for computational narrative: an approach to generative and interactive narrative with bases in algebraic semiotics and cognitive linguistics. Ph.D. thesis, University of California, San Diego.
 - [35] HAYMAN, J. AND HEINDEL, T. 2014. On pushouts of partial maps. In *Graph Transformations. Proceedings of the 7th Biannual International Conference (ICGT 2014)*, H. Giese and B. König, Eds. Lecture Notes in Computer Science, vol. 8571. Chapter 12, 177–191.
 - [36] HÖHLE, U. 2014. Categorical foundations of topology with applications to quantaloid enriched topological spaces. *Fuzzy Sets and Systems* 256, 166–210.
 - [37] JOHNSON, M. 1987. *The Body in the Mind*. University Of Chicago Press. The Bodily Basis of Meaning, Imagination, and Reason.
 - [38] JOHNSTONE, P. T. 2002. *Sketches of an elephant: a topos theory compendium*. 2 Volumes. Oxford Logic Guides, vol. 44. The Clarendon Press Oxford University Press, Oxford.
 - [39] KAHL, W. 2010a. Co-tabulations, bicolimits and van-Kampen squares in collagories. *Electronic Communications of the EASST* 29.
 - [40] KAHL, W. 2010b. Collagory Notes, version 1. SQRL Report 57. Electronically available at <http://www.cas.mcmaster.ca/sqrl/papers/SQRLreport57.pdf>.
 - [41] KOESTLER, A. 1964. *The Act of Creation*. Hutchinson & Co.
 - [42] KUTZ, O., MOSSAKOWSKI, T., HOIS, J., BHATT, M., AND BATEMAN, J. 2012. Ontological blending in DOL. In *Computational Creativity, Concept Invention, and General Intelligence - 1st International Workshop. International Workshop on Computational Creativity, Concept Invention, and General Intelligence (C3GI-12), First, located at ECAI 2012, August 27, Montpellier, France*. Publication Series of the Institute of Cognitive Science.

- [43] 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*.
- [44] LABOVE, W. 1972. The transformation of experience in narrative syntax. *Language in the inner city*.
- [45] LACK, S. 2010. A 2-categories companion. In *Towards higher categories*. IMA Vol. Math. Appl., vol. 152. Springer, New York, 105–191.
- [46] LAKOFF, G. 1987. *Women, Fire, and Dangerous Things*. University Of Chicago Press.
- [47] LAKOFF, G. AND JOHNSON, M. 1999. *Philosophy In The Flesh: The Embodied Mind And Its Challenge To Western Thought*. Basic Books.
- [48] LAKOFF, G. AND NÚÑEZ, R. 2000. *Where Mathematics Come From: How The Embodied Mind Brings Mathematics Into Being*. Basic Books.
- [49] LANE, S. M. 1998. *Categories for the working mathematician*, Second ed. Graduate Texts in Mathematics, vol. 5. Springer-Verlag, New York.
- [50] LEINSTER, T. 1998. Basic bicategories. Electronically available at <http://arxiv.org/abs/math/9810017>.
- [51] LEINSTER, T. 2002. A survey of definitions of n -category. *Theory Appl. Categ.* 10, 1–70.
- [52] MALCOLM, G. 2000. *Software Engineering with OBJ: algebraic specification in action*. Kluwer.
- [53] MCLARTY, C. 1992. *Elementary categories, elementary toposes*. Oxford Logic Guides, vol. 21. The Clarendon Press Oxford University Press, New York. Oxford Science Publications.
- [54] MOSSAKOWSKI, T. 1998. Colimits of order-sorted specifications. In *Recent trends in algebraic development techniques (Tarquinia, 1997)*. Lecture Notes in Computer Science, vol. 1376. Springer, Berlin, 316–332.
- [55] MOSSAKOWSKI, T., MAEDER, C., AND LÜTTICH, K. 2007. The heterogeneous tool set, Hets. In *Tools and Algorithms for the Construction and Analysis of Systems, 13th International Conference, TACAS 2007, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2007 Braga, Portugal, March 24 - April 1, 2007, Proceedings*, O. Grumberg and M. Huth, Eds. Vol. 4424. Springer, 519–522.
- [56] MURPHY, G. L. AND MEDIN, D. L. 1985. The role of theories in conceptual coherence. *Psychological Review*.
- [57] ONTAÑÓN, S. AND PLAZA, E. 2010. Amalgams: A formal approach for combining multiple case solutions. In *Case-Based Reasoning. Research and Development, 18th International Conference on Case-Based Reasoning, ICCBR 2010, Alessandria, Italy, July 19-22, 2010. Proceedings*, I. Bichindaritz and S. Montani, Eds. Lecture Notes in Computer Science, vol. 6176. Springer, 257–271.

-
- [58] ONTAÑÓN, S. AND PLAZA, E. 2012. Toward a knowledge transfer model of case-based inference. In *Proceedings of the Fifteenth International Florida Artificial Intelligence Research Society (FLAIRS)*. AAAI Press.
 - [59] PEREIRA, F. C. 2005. A computational model of creativity. Ph.D. thesis, Universidade de Coimbra.
 - [60] PEREIRA, F. C., Ed. 2007. *Creativity and artificial intelligence a conceptual blending approach*. Applications of Cognitive Linguistics, vol. 4. Mouton de Gruyter, Berlin.
 - [61] PIERCE, B. C. 1991. *Basic category theory for computer scientists*. Foundations of Computing Series. MIT Press, Cambridge, MA.
 - [62] PLATE, T. A. 2003. *Holographic Reduced Representation: Distributed Representation for Cognitive Structures*. CSLI Lecture Notes. Center for the Study of Language and Information.
 - [63] POIGNÉ, A. 1986. Elements of categorical reasoning: products and coproducts and some other (co-)limits. In *Category theory and computer programming (Guildford, 1985)*, D. Pitt, S. Abramsky, A. Poigné, and D. Rydeheard, Eds. Lecture Notes in Computer Science, vol. 240. Springer, Berlin, 16–42.
 - [64] QUILLIAN, M. 1968. Semantic memory. *Semantic Information Processing*.
 - [65] RAOULT, J. 1984. On graph rewritings. *Theoretical Computer Science* 32, 1-2, 1–24.
 - [66] ROBINSON, E. AND ROSOLINI, G. 1988. Categories of partial maps. *Information and Computation* 79, 2, 95–130.
 - [67] SANNELLA, D. AND TARLECKI, A. 2012. *Foundations of Algebraic Specification and Formal Software Development*. Monographs in Theoretical Computer Science. An EATCS Series. Springer.
 - [68] SCHMIDT, M., KRUMNACK, U., GUST, H., AND KÜHNBERGER, K. 2014. Heuristic-driven theory projection: An overview. In *Computational Approaches to Analogical Reasoning: Current Trends*, H. Prade, G. Richard, H. Prade, and G. Richard, Eds. Studies in Computational Intelligence, vol. 548. Springer, Chapter 7, 163–194.
 - [69] SCHWERING, A., KRUMNACK, U., KÜHNBERGER, K.-U., AND GUST, H. 2009. Syntactic principles of heuristic-driven theory projection. *Cognitive Systems Research* 10, 3, 251–269.
 - [70] TARLECKI, A. 2014. Category theory in foundations of computer science. Slides of a Winter Semester Course of 2013/2014. Electronically available at <http://www.mimuw.edu.pl/~tarlecki/teaching/ct/index.html>.
 - [71] THAGARD, P. 2010. *The Brain and the Meaning of Life*. Princeton University Press.
 - [72] THAGARD, P. AND STEWART, T. C. 2011. The AHA! experience: Creativity through emergent binding in neural networks. *Cognitive Science* 35, 1, 1–33.
 - [73] VEALE, T. AND KEANE, M. 1997. The competence of sub-optimal theories of structure mapping on hard analogies. In *IJCAI*. 232–237.

- [74] VEALE, T. AND O'DONOGHUE, D. 2000. Computation and blending. *Cognitive Linguistics 11*, 3-4, 253–282.
- [75] WIGGINS, G. A. 2006. A preliminary framework for description, analysis and comparison of creative systems. *Knowledge-Based Systems 19*, 7, 449–458.

A Characterisation of $\frac{3}{2}$ -colimits in **Pfn**

The aim of this section is to provide a characterisation of the notion of $\frac{3}{2}$ -colimits (see [Definition 4.16](#)) in the case of the ordered category **Pfn** (with the extension partial order). The content of such characterisation is stated in [Theorem A.4](#). It is worth saying that most of the argument here provided can be carried out in the more general setting of having an ordered category with partial orders that are dcpos and composition being Scott-continuous (cf. [Page 43](#)).

To start dealing with $\frac{3}{2}$ -colimits in **Pfn** it is convenient to realise that the system of inequations involved in the definition can be easily solved in the ordered category **Rel** of binary relations. Indeed, it is well-known that the following definition

$$\begin{aligned} \Rightarrow : \mathbf{Rel}(A, C) \times \mathbf{Rel}(A, D) &\longrightarrow \mathbf{Rel}(C, D) \\ (R, S) &\longmapsto R \Rightarrow S := \{(x, y) \in C \times D \mid \forall z \in A (\text{if } (z, x) \in R \text{ then } (z, y) \in S)\} \end{aligned}$$

fulfills that for every relation Z ,

$$R; Z \sqsubseteq S \quad \text{iff} \quad Z \sqsubseteq R \Rightarrow S.$$

Therefore, the previously defined relation $R \Rightarrow S$ happens to be the maximum element of $\{Z \mid R; Z \sqsubseteq S\}$. An immediate consequence of such fact is the following result.

Lemma A.1. *Let \mathcal{D} be a lax diagram in **Pfn** and let \mathfrak{c} be a lax cocone over \mathcal{D} . Then, the following statements are equivalent.*

1. \mathfrak{c} is a $\frac{3}{2}$ -colimit over \mathcal{D} .
2. For every lax cocone \mathfrak{d} over \mathcal{D} , it happens that the binary relation $\bigcap \{\mathfrak{c}_x \Rightarrow \mathfrak{d}_x \mid X \in \text{Node}\}$ is functional.

Therefore, from now on our purpose is to characterise the second condition in [Lemma A.1](#), and this is going to be possible using a surjectivity condition. Before doing so we provide a lot of conditions that happen to be equivalent in the context of **Pfn**. The following equivalences are stated dealing with only one morphism, but it is obvious it can also be stated for a family of morphisms following the same idea given in [Footnote 8](#). Let us also say that the first condition was seen in [Page 45](#) to hold for all $\frac{3}{2}$ -colimits.

Proposition A.2. *Let $f: A \rightarrow B$ be a morphism in **Pfn**. Then, the following statements are all equivalent.*

1. For every morphisms h_1 and h_2 , if $f; h_1 = f; h_2$ then h_1 and h_2 are compatible.
2. f is surjective.
3. f is an epimorphism (i.e., for every morphisms h_1 and h_2 , if $f; h_1 = f; h_2$ then $h_1 = h_2$).
4. f is a compatibly epimorphism (i.e., for every morphisms h_1 and h_2 , if $f; h_1$ and $f; h_2$ are compatible functions, then h_1 and h_2 are also compatible).
5. f is a semiepimorphism (i.e., for every morphisms h_1 and h_2 , if $f; h_1 \sqsubseteq f; h_2$ then $h_1 \sqsubseteq h_2$).

Proof. Some of these equivalences are very well-known in the literature; and the new ones can be proved using the same standard arguments (e.g., if f is not surjective then it is easy to provide two functions h_1 and h_2 such that $f;h_1 = f;h_2$ while h_1 and h_2 are not compatible). \square

Proposition A.3. *Let \mathcal{D} be a lax diagram in **Pfn** and let \mathfrak{c} be a lax cocone over \mathcal{D} which is jointly epimorphic. Then, for every family $\{R_X \mid X \in \text{Node}\}$ of binary functional relations it holds that the binary relation $\bigcap \{\mathfrak{c}_x \Rightarrow R_X \mid X \in \text{Node}\}$ is functional.*

Proof. Let us suppose that both (x, y) and (x, y') are elements in the binary relation $\bigcap \{\mathfrak{c}_x \Rightarrow R_X \mid X \in \text{Node}\}$. Using that \mathfrak{c} is jointly epimorphic we know that there is some node X and some element $a \in \text{src}(\mathfrak{c}_X)$ such that $\mathfrak{c}_X(a) = x$. Thus, $(a, x) \in \mathfrak{c}_X$. Using that $(x, y) \in \mathfrak{c}_X \Rightarrow R_X$ and $(x, y') \in \mathfrak{c}_X \Rightarrow R_X$, together with the definition of \Rightarrow , we deduce that $(a, y) \in R_X$ and $(a, y') \in R_X$. Since R_X is functional we conclude that $y = y'$, which finishes the proof. \square

Theorem A.4 (Characterization of $\frac{3}{2}$ -colimits in **Pfn**). *Let \mathcal{D} be a lax diagram in **Pfn** and let \mathfrak{c} be a lax cocone over \mathcal{D} . Then,*

$$\mathfrak{c} \text{ is a } \frac{3}{2}\text{-colimit over } \mathcal{D} \quad \text{iff} \quad \mathfrak{c} \text{ is jointly epimorphic.}$$

Proof. The rightwards implication is a consequence of what was seen in Page 45 (together with Proposition A.2). And the leftwards implication is a consequence of Lemma A.1 and Proposition A.3. \square

In the particular case of considering a V-shaped diagram we obtain the following corollary (cf. Definition 4.17).

Corollary A.5 (Characterization of $\frac{3}{2}$ -pushouts in **Pfn**). *A $\frac{3}{2}$ -pushout of a span $B_1 \xleftarrow{f_1} A \xrightarrow{f_2} B_2$ is given by a lax cocone*

$$\begin{array}{ccccc} & & C & & \\ & g_1 \nearrow & \uparrow g & \nwarrow g_2 & \\ B_1 & \sqsubseteq & & \sqsupseteq & B_2 \\ & f_1 \nwarrow & \downarrow & \nearrow f_2 & \\ & & A & & \end{array}$$

such that $\text{Im}(g_1) \cup \text{Im}(g) \cup \text{Im}(g_2) = C$.