Chapter 9
A Compositional Framework

A couple in love walking along the banks of the Seine are, in real fact, a couple in love walking along the banks of the Seine, not mere particles in motion.

Stewart A. Kauffman [168]

The system architecture presented in Chap. 8 controls (i.e., sustains and constrains) the invocation of the inference methods introduced in Chap. 7. In this chapter, we describe the methods of higher level inference in more detail. There is increasing conviction that methods of *category theory* are well-suited for providing generic descriptions for cognitive science, general intelligence [258], and control [42], the latter being newly christened as 'categorical cybernetics'. In this chapter, we describe how to leverage the power of selected category-theoretic constructions to realize SCL operations in a compositional manner.

9.1 Categorical Cybernetics

Category theory has served as a formidable unifying mechanism in mathematics. It was devised in the 1940s by Eilenberg and MacLane [209] in order to provide a higher-order vocabulary for algebraic topology and defines a principled setting for the study of structurally-informed transformations. Amongst applied category theorists there is increasing interest in machine learning applications, and it is becoming apparent (e.g. [78, 90, 318]) that much can be done to unify and generalize existing methods. Following a brief overview of category theory, we describe how SCL operations may be implemented in terms of specific category-theoretic constructions. Below, we describe the essential concepts; more detail is available in various excellent texts (e.g. [196, 264, 290]). Mathematical approaches to formalizing compositionality can be broadly divided into 'syntactic' and 'semantic'. While syntactic

J. Swan et al., *The Road to General Intelligence*,
Studies in Computational Intelligence 1049,
https://doi.org/10.1007/978-3-031-08020-3_9

approaches such as those in linguistics [361] and process algebra [224] might be better known, the semantic approach via category theory has become increasingly popular in diverse fields due to its flexibility and mathematical elegance [89].

A category is a two-sorted algebraic structure rather like a graph, consisting of 'objects' and 'morphisms': every morphism has a source object and a target object, written $f : X \to Y$. In addition to the graph-like structure is a way to compose morphisms: given any morphisms $f : X \to Y$ and $g : Y \to Z$ with the target of one agreeing with the source of the other, there is a composite morphism $fg : X \to Z$ (typically written $g \circ f$). This must satisfy the 'associativity property' familiar from algebraic structures such as groups, i.e., given three composable morphisms f, g, h the two different ways of composing them must agree: $(fg)h = f(gh)$. Every object must also have an assigned 'identity morphism' $1_X : X \to X$, and they must act as the identity element for composition: $1_X f = f = f 1_Y$ for all $f : X \to Y$.

There are many examples of categories, and we will name only a handful:

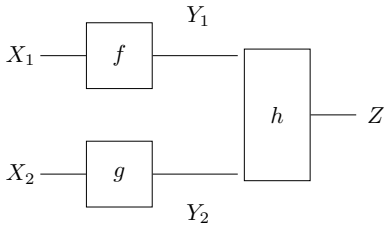
1. The category **Set** of sets, whose objects are sets and morphisms are functions.
2. The category **FinVec** of finite-dimensional vector spaces, whose objects are finite dimensional vector spaces and morphisms are linear maps.
3. The category **Rel** of relations, whose objects are sets and morphisms are binary relations.
4. For any graph G, there is the 'free category on G', whose objects are nodes of G and morphisms are paths in G. Composition is concatenation of paths, and identity morphisms are paths of length zero.
5. Any monoid M can be viewed as a category with a single object $*$, where every element $m \in M$ is viewed as a morphism $m : * \to *$.

Of particular relevance is the fact that typed programming languages (e.g. the simply-typed lambda-calculus [47] or intuitionistic type theory [217]) give rise to corresponding categories: it is possible[1] to consider a category with types as objects and functions as morphisms [54].

When studying compositionality, it is typical to work in the context of 'monoidal categories', which have additional structure: there is a monoid-like structure on objects, with a binary operation $\otimes$ and a unit I, in a way that is also compatible with morphisms, so if $f : X_1 \to Y_1$ and $g : X_2 \to Y_2$ are morphisms, then so is $f \otimes g : X_1 \otimes X_2 \to Y_1 \otimes Y_2$, satisfying various laws. A category typically has several different monoidal structures. For example, in the category of sets we could take $\otimes$ to be Cartesian product of sets (whose unit is a 1-element set) or disjoint union of sets (whose unit is the empty set). In the category of finite-dimensional vector spaces we could take $\otimes$ to be the direct product (whose unit is the 0-dimensional space) or the tensor product (whose unit is the 1-dimensional space).

Morphisms in a monoidal category are often represented using the graphical notation of *string diagrams* [216]. For example, if we have morphisms $f : X_1 \to Y_1$, $g : X_2 \to Y_2$ and $h : Y_1 \otimes Y_2 \to Z$, then the composite morphism $(f \otimes g)h : X_1 \otimes X_2 \to Z$ is represented by the diagram:

[1] Modulo certain technical considerations.



The other basic concept required is a 'functor', which is a structure-preserving map between categories. If $\mathcal{C}$ and $\mathcal{D}$ are categories than a functor $F : \mathcal{C} \to \mathcal{D}$ is an assignment sending every object X in $\mathcal{C}$ to an object $F(X)$ in $\mathcal{D}$, and every morphism $f : X \to Y$ in $\mathcal{C}$ to a morphism $F(f) : F(X) \to F(Y)$ in $\mathcal{D}$, in a way that preserves identities and composition. If our categories are monoidal then we consider 'monoidal functors', which also preserve the monoidal structure.

9.2 Hypothesis Generation

One of the most important requirements for general intelligence is that the hypotheses which are generated are *salient*, i.e., pertinent to the task at hand. In the case of the simple pendulum, the closed form expression was obtained by virtue of observation that e.g. the color of the pendulum bob was not relevant, but the angle of displacement was, and so on. However, this does not imply that all features of the pendulum were given equal attention. Humans have sufficiently strong priors about force and motion that it is hard to imagine an experimenter ever consciously entertaining color as a factor. It is therefore evident that scientific hypothesis generation enjoys a degree of subtlety which is absent from traditional ML approaches.

Previous such work on non-quantitative generation of alternative hypotheses can be found in Mitchell and Hofstadter's 'Copycat' [146]. Copycat is proposed as a cognitively-plausible generator of proportional analogies between letter-strings and operates without requiring a 'top-down, a priori' objective function. In the abstract, Copycat can be considered as an interpreter for expressions that describe (potentially partially constructed) analogies, in which top-down and bottom-up perspectives interact. At any point in construction, structures can be interpreted to yield what Hofstadter describes as their 'counterfactual halo', i.e., to suggest alternative expressions that tend to be more self-consistent. Copycat avoids explicit combinatorial search via a combination of attention heuristics (which share a great deal of commonality with the scheduling mechanism described in Sect. 8.2) and interacting hierarchical constraints. Salient actions are indexed 'on demand' by local updates to hypotheses and no form of global truth maintenance is required. These local updates act to greatly prune the space of possible alternatives, being biased by design in the general direction of 'more consistent, hence more compelling' hierarchies.

More generally, a number of previous works in cognitive architectures argue that the frame problem is an artifact of an overly rigid perspective on hypothesis repre-

sentation [97, 146]. Specifically, the claim is that hypotheses should be both causally grounded (via participation in a sensor-effector mapping that receives feedback from the environment) and 'reified on demand' via the data-driven context of task features. It is claimed that the arguments of a priori representationalism that lead to the frame problem are then no longer applicable. Such context-specific hypothesis-chaining is an intrinsic aspect of SCL: salience is facilitated via the joint action of fine-grained scheduling and resource bounds on both time and space [325]. In the following section, we describe a general mechanism for interpreting the structure of hypotheses, in which alternative hypotheses are generated via a specific form of interpretation which yields a modified hypothesis as a result.

Compositional Interpretation

Compositional interpretation of SCL-expressions is achieved via *denotational semantics*: the meaning of a composite expression is interpreted as as a function of the meaning of its component parts. In a 'closed-world' setting, such an interpretation is fixed at the point of deployment. In an open world setting, it remains forever possible that there are surprising latent interactions between components and the interpretation process must therefore incorporate online learning.

Marcus [210, 214] has previously contrasted deep learning with human capabilities (as inferred via observations from neuroscience), and notes that the former typically lacks generic mechanisms for recursion and variable binding. He proposes a hierarchical structured representation ('treelets') for addressing this, but leaves open the question of how they should be manipulated for efficient inference. As regards efficiency, Marcus gives desiderata which are evocative of the 'active symbols' of Mitchell and Hofstadter's 'Fluid Analogies' architecture [146], the direct analog of which in SCL is provided via the matching and transformation processes described in the beginning of this chapter.

It has previously been proposed [259] that the category theoretic mechanism of *initial F-algebras* provides an appropriate and parsimonious means of modeling these aspects of human cognition and we describe below a concrete application that supports such recursion and variable binding. F-algebras provide a universal mechanism for the generic interpretation of expressions. By 'generic', we mean that a wide class of typed expression languages can be compositionally interpreted. As we describe below, 'universal' has a specific technical meaning in category theory, but can for practical purposes be taken to mean that the interpreter is parameterized by the target datatype and can approximate (e.g. via learning [333, 334]) any primitive recursive semantic interpretation.[2] Moreover, the interpreter can both accommodate recursive expression languages and be stateful (and hence perform variable-binding, so meeting both of Marcus's requirements, above).

[2] Strictly, the expressiveness is more general than primitive recursive in that it includes e.g. Ackermann's function [154].

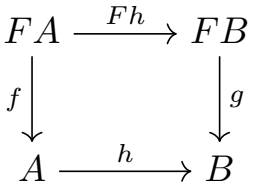


Fig. 9.1 h as a homomorphism

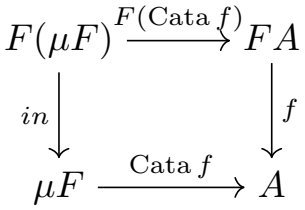


Fig. 9.2 Cata as a unique homomorphism

Technically, the universiality property arises as follows: for category C and functor $F : C \to C$, an *algebra*[3] consists of a pair (A, f) consisting of an object A and a morphism $f : FA \to A$. A *homomorphism* $h : (A, f) \to (B, g)$ between algebras is a morphism $h : A \to B$ such that the square in Fig. 9.1 commutes.

In the category that has algebras as objects and homomorphisms as morphisms, an *initial algebra* is an algebra that is unique (up to isomorphism) and *initial*, i.e., it can be transformed into any other algebra in the category. We write $(\mu F, in)$ for an initial algebra, and Cata f for the unique homomorphism $h : (\mu F, in) \to (A, f)$ from the initial algebra to any other algebra (A, f). That is, Cata f is defined as the unique arrow that makes the diagram of Fig. 9.2 commute.

The universal interpreter property of Cata then arises by virtue of this initiality [154]. Cata is an abbreviation of 'catamorphism' (from Greek: $\kappa\alpha\tau\alpha$ 'downwards' and $\mu o\rho\varphi\eta$ 'shape'); informally, a generic method of transforming some typed source expression into a target expression (of whatever desired type). Hence, in a category of expressions Ex in which the objects are types and the morphisms are functions, Cata thus provides an *algorithm template* for the interpretation of expressions. Hence predicates on Ex are represented as a transformation of some source expression which has target type bool and alternative hypotheses as having target type Ex.

As previously mentioned, in the wider context of general intelligence, such a 'Closed World Assumption' is insufficient: reality may always intrude to defeat prior assumptions.[4] Such exceptions arise as a function of the difference between expected and observed states—whether because an action fails to yield the anticipated state, or else because some state of interest arises in an unanticipated manner. In the context of SCL, then, the term hypothesis recovers its traditional meaning as a tentative proposition about reality that remains forever potentially subject to revision. When

[3] This definition subsumes the familiar usage of the term.

[4] As discovered by Bertrand Russell's unfortunate fictional chicken.

the distributed transition function of the world model is determined to be in error in this manner, a corresponding 'repair' must be applied to the current denotational semantics. Those repairs can either widen the constraints or suitably constrain the transfer function.

In SCL, constraints are expressed via the predicate of a refinement type. As types are aggregated into composites (via sum and product), so the hierarchical structure of predicates becomes more complex. In the particular case of hypothesis generation, the catamorphic traversal of expressions accumulates proposed alternatives and performs 'conflict resolution' on them in order to propose a hypothesis that better meets its constraints. Any inference mechanism could conceivably be applied in the process, depending on the intermediate mappings between types that are required. Fortunately, the repair process is considerably facilitated by the granular nature of inference: it is typical that repairs require only local changes to inference rules.

It is also interesting to note the overlap here with the contemporaneous work of Goertzel [116], which proposes *chronomorphisms* as a potential unifying mechanism for the OpenCog framework. While we certainly share the belief that the zoo of recursion schemes (including anamorphisms, futuromorphisms, etc.) have a role to play in open-ended inference, we are not of the opinion that their role lies in solving optimization problems, as discussed in Sect. 5.1 on a priori rewards.

9.3 Abstraction and Analogy

Consensus on the value of abstraction in AI dates back to the inception of the field [220]. Various forms of analogical reasoning have similarly been widely argued, by cognitive and computer scientists alike [84, 144], to play a vital role. There are a wide variety of proposed definitions for both. For example, Gentner [107] defines abstraction as 'the process of decreasing the specificity of a concept'. Cremonini et al. [53] define abstraction as 'the process of mapping between problem representations, so as to simplify reasoning while preserving the essence of the problem'. Definitions of abstraction and analogy can overlap considerably. For example, the formal and general abstraction framework of Giunchiglia and Walsh [111] describes abstraction in terms of properties that are provably preserved under the source to target mapping. This definition could also be said to be applicable to *predictive analogy* [271], which is concerned with inferring properties of a target object as a function of its similarity to a source object, the oft-cited example of which is the similarity between the solar system and the Rutherford model of the atom. Given the perceived richness and complexity of abstraction and analogy, this overlap is unsurprising. Indeed, it seems possible that the processes are recursively interleaved in a nontrivial and data-driven manner. Hence, whilst in this Section we propose a concrete mechanism for abstraction that can then be used as a basis for analogy, this should be considered as a pedagogical device rather than any attempt at a definitive statement.

The interpretation of SCL expressions via catamorphisms already provides an elementary abstraction mechanism: the algorithm skeleton for the interpreter abstracts over the generation of alternative types and values from a given expression. However,

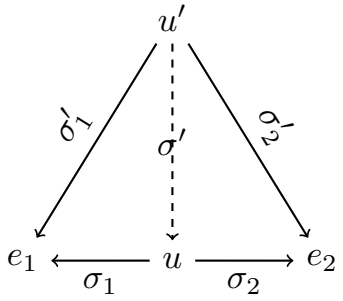


Fig. 9.3 Anti-unification as a categorical product. The anti-unifier of expressions e_1 and e_2 is an expression u, together with two substitutions σ_1 and σ_2. For $s \rightarrow_\sigma t$, we say that t is a *specialization* of s, i.e. it has been obtained from s via the instantiation of one or more variables. The *least general* anti-unifier (u, σ_1, σ_2) is the *unique* u such that, for any other candidate $(u', \sigma_1', \sigma_2')$, u is a specialization of u' via some substitution σ'

since expressions in SCL are first-class objects, we may also perform abstraction via other means, such as *anti-unification*. Anti-unification has a variety of uses in program analysis, including invariant generation and clone detection [39]. The various forms of unification [266] can be described categorically [119] via a category in which the objects are terms (i.e. SCL expressions) and the morphisms are *substitutions*, i.e., the binding of one or more variables to subexpressions. Anti-unification is the categorical dual of unification [162], representing the factorization of substructure common to two expressions. The discovery of such 'abstract patterns' is analogous to the induction of subroutines, which can be instantiated across a range of parameter values. More generally, abstraction is applicable across different dimensions of the state space—see Sect. 11.2.2 for a discussion of wider prospects in this regard.

Figure 9.3 depicts anti-unification as a *categorical product*, a construction that generalizes the familiar notion of Cartesian product. The diagram denotes that anti-unifier (u, σ_1, σ_2) is more specialized than all other candidates $(u', \sigma_1', \sigma_2')$ because the latter can be recovered from the former via σ'.

Analogical reasoning in humans appears to afford unbounded creative potential [95]. We share the belief that analogy is a dominant mechanism in human cognition [148, 228] and envisage that computational models of analogy will be a key research area for general intelligence. It should be clear that such research is completely open-ended. The categorical approach we describe below is therefore for pedagogical purposes; the further incorporation of heuristics is a more realistic prospect for practical use.

We give a categorical construction for proportional analogies which builds upon the method of abstraction defined above.

As previously described in Sect. 7.4, the example application domain we consider here is that of letter-string analogy (LSA) problems (e.g., `abc : abd :: ijk : ???`). Although the domain may appear simple, it has been remarked that it can require considerable sophistication to obtain solutions that appear credible to humans

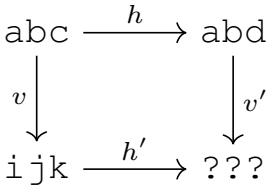


Fig. 9.4 Proportional analogy as a commutative diagram: `abc : abd :: ijk : ???`

[95], not least because the domain is not readily modeled as an optimization problem. It is therefore reasonable to assume that mechanisms for solving LSA problems have high relevance and are applicable for implementing cognitive processes at many levels. Notable approaches to LSA problems include Hofstadter and Mitchell's Copycat [146] and the 'E-generalization' approach of Weller and Schmid [360], although the latter is not cognitively plausible for reasons of scale.

As can be seen in Fig. 9.4, proportional analogy problems can also be considered to form a commutative diagram. The 'abstraction via anti-unification' approach described above can be used as a building block for constructing analogies, for example as is done in 'Heuristic Driven Theory Projection' [308]. In particular, abstraction can be combined with the powerful category theoretic constructions of *pushouts* and *pullbacks* to construct, express, and understand such analogies in a computationally automatable manner across a wide range of expression languages. Specifically, we can use these constructions to determine the possible relationships between our objects $A = \texttt{abc}$, $B = \texttt{abd}$ and $C = \texttt{ijk}$ such that $D = \texttt{ijl}$ is uniquely determined through commutative diagrams.[5]

Pushouts

A pushout may be understood as an 'abstract gluing' of a pair of morphisms $b : A \to B$ and $c : A \to C$. The construction of a pushout involves the construction of some fourth object D alongside morphisms $c' : B \to D$ and $b' : C \to D$ such that the resulting diagram commutes, i.e., $c'(b(A)) = b'(c(A)) \cong D$, where $\cong$ denotes equality up to isomorphism. Further, the resultant commuting diagram must satisfy the 'pushout condition', that for all D' with $e : B \to D'$ and $f : C \to D'$ and $e(b(A)) = f(c(A)) \cong D'$, there exists a unique morphism $d : D \to D'$ such that $d(c'(B)) = d(b'(C)) = e(B) = f(C) \cong D'$.

The meaning of these conditions will become immediately clear through an example of a pushout in the category of sets (typically denoted **Set**). In **Set**, objects are sets and morphisms are functions. Below we given an example of a pushout in **Set**, with objects $A = \{a, b, c\}$, $B = \{1, 2, 3, 4\}$ and $C = \{W, X, Y, Z\}$, and morphisms $h = \{a \to 1, b \to 2, c \to 3\}$ and $v = \{a \to X, b \to Y, c \to Z\}$. For each object in B, C and D, we denote its pre-image in A in parenthesis.

[5] The possibility $D = \texttt{ijd}$ is not treated here; a more fully-featured approach to analogy would include nondeterminism and preference heuristics, e.g. as in Goguen's 'sesqui-categories' [117].

$$\begin{array}{ccc} A = \{a, b, c\} & \xrightarrow{\quad h \quad} & B = \{1(a), 2(b), 3(c), 4\} \\ \downarrow v & & \downarrow v' \\ C = \{W, X(a), Y(b), Z(c)\} & \xrightarrow{h'} & D = \{W, 1X(a), 2Y(b), 3Z(c), 4\} \end{array}$$

The resultant object $D = \{W, 1X, 2Y, 3Z, 4\}$ with morphisms $v' = \{1 \rightarrow 1X, 2 \rightarrow 2Y, 3 \rightarrow 3Z, 4 \rightarrow 4\}$ and $h' = \{X \rightarrow 1X, Y \rightarrow 2Y, Z \rightarrow 3Z, W \rightarrow W\}$ are the unique pushout of morphisms h and v.

The uniqueness is a result of the pushout condition; if we instead had $D' = \{1X, 2Y, 3Z, 4W\}$ and $v'(4) = 4W$ and $h'(W) = 4W$ then we would have elements in B and C, with no common pre-image in A, being mapped to the same element in D'. As a result, there would be no morphism $d : D' \rightarrow D$, such that the resultant diagram commutes as the offending element $4W$ would need to be mapped to two separate elements in $D, 4$ and W. Thus we can see that the pushout condition prevents 'confusion' such that, if an element in D has pre-images in both B and C, then those pre-images must themselves have a common pre-image in A.

Similarly, if we have $D' = \{W, 1X, 2Y, 3Z, 4, Q\}$, such that the element Q has no pre-image in B or C, then it may be mapped to any element in D by some morphism $d' : D' \rightarrow D$, such that d' is no longer unique. Thus we can see that the pushout condition prevents 'junk' such that all elements in D must have a pre-image in at least one of B or C. A common phrase describing this property is that v' and h' are *jointly surjective*.

Another relevant concept related to pushouts is that of pushout complements. A pushout complement is the completion of a pair of arrows $h : A \rightarrow B$ and $v' : B \rightarrow D$ to a pushout, and is given by an object C with morphisms $v : A \rightarrow C$ and $h' : C \rightarrow D$ such that the resulting diagram of h, h', v and v' is a pushout.

Pullbacks

If pushouts are 'abstract gluings' of morphisms, then 'pullbacks' may be understood as 'abstract intersections' of morphisms. A pullback is given over a pair of morphisms $h : B \rightarrow D$ and $v : C \rightarrow D$. The construction of the pullback of h and v yields a fourth object A with morphisms $v' : A \rightarrow B$ and $h' : A \rightarrow C$ such that the resulting diagram commutes and satisfies the 'pullback' condition. This condition requires that for all A' with $e : A' \rightarrow B$ and $f : B' \rightarrow C$ and $e(h(A')) = f(v(A')) \cong D$, there exists a unique morphism $a : A' \rightarrow A$ such that $v'(a(A')) = B$ and $h'(a(A')) = C$.

Analogy via Pushouts, Pullbacks, and Pushout Complements

We now demonstrate that these concepts give mechanisms for automatic derivation of analogies according to abstractions. For a more complete description of pushouts, pullbacks, and pushout complements in this context, see Taentzer et al. [76]. In Fig. 9.5, we give a sketch of a pushout-based solution to classic letter-string analogy problems. Letter-strings are represented as lists of natural numbers, with natural numbers themselves represented via Peano arithmetic. We are working in the category

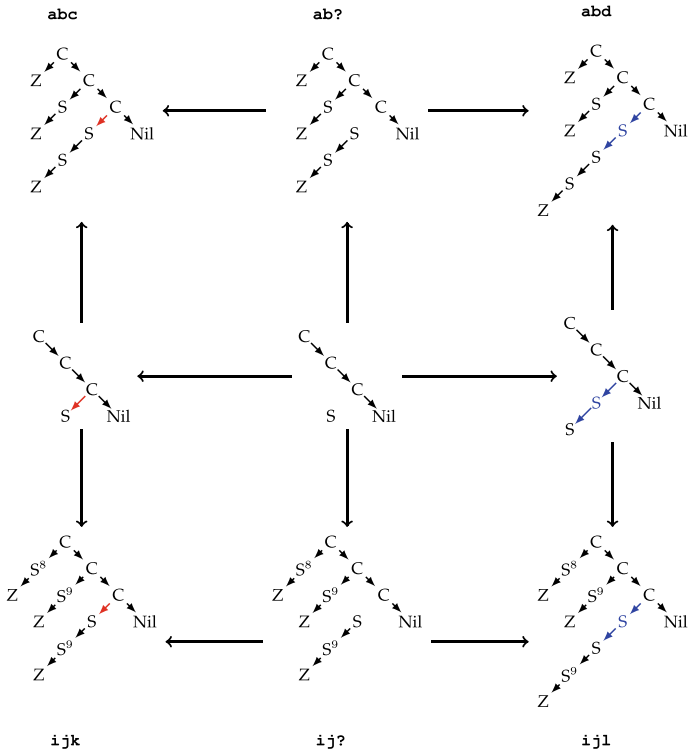


Fig. 9.5 A solution to the letter-string analogy question `abc : abd :: ijk : ???` via pushouts. Each letter string is represented by its corresponding tree in Peano arithmetic, with the letter 'a' corresponding to the number '0'. The shorthand 'S^x' is used to concisely represent a sequence of x consecutive successor nodes. Elements in red are 'deleted' by the transformation, whereas elements in blue are 'created'. The middle rule can be viewed as an abstract rule for transforming letter-strings, and interpreted in language as 'increment the final (non-`a`) character in the letter-string'

of labeled graphs (an 'adhesive category' [185]), along injective morphisms which are both label- and structure-preserving. Note that in this scenario, relabeling may be achieved in rewriting systems over labeled objects through the use of partially labeled objects as intermediaries [126].

The example works as follows. We are provided with the expression trees of the letter strings `abc`, `abd`, and `ijk`. We know that `abc` relates to `abd` in some manner and wish to induce the equivalent relation for `ijk` and some unknown letter string. Thus the task may be phrased "`abc` is to `abd` as `ijk` is to what?". The first step is to induce common substructures between the pairs `abc` and `abd` as well as `abc` and `ijk`. Two particularly promising substructures are shown in the top middle and left middle of the diagram. This first step is most critical; the two common substructures will deterministically define the rest of the transformation. If either common substructure is too specific, it may form too rigid a restriction for

the rest of the process to be successful. Alternatively, if the common substructures are too general, ambiguity may occur causing there to be several possible analogous letter-strings.

Given the two common substructures, a pullback may be constructed, yielding the graph in the center of the diagram. This may be understood as the common elements of both common substructures such that the top-left square commutes. With the central element, we may then construct the pushout complements in the middle-right and bottom-middle spaces, and finally the pushout of those elements with the central element to give the final analogous graph: `ijl`. A remarkable property of this process is that it actually yields an abstract rule $L \leftarrow K \rightarrow R$ (the elements of the middle row), which may then be applied to any other letter-string expression T according to the following process:

1. Find a graph morphism $f : L \rightarrow T$.
2. Construct the pushout complement of f and $L \leftarrow K$ to give intermediary D with morphisms $t : D \rightarrow T$ and $g : K \rightarrow D$.
3. Construct the pushout of g and $K \rightarrow R$, giving result expression S.

This process is only applicable if there exists a graph morphism $f : L \rightarrow T$, and this is only the case when the letter-string expression T is at least 3 letters long and its final letter is not `a`. Hence the rule $L \leftarrow K \rightarrow R$ may be interpreted in language as; 'increment the final (non-`a`) character in the letter-string'. When applied to the letter-strings `ddd` and `mmjjkk` it gives the unique results `dde` and `mmjjkl` respectively.

Although LSA is given as an example domain, the treatment is a general one for adhesive categories [185]. This means that the principles described above can be leveraged to induce analogies over labeled graphs [76] and their derivatives (e.g. forests and trees), hierarchical graphs [72], and port graphs [85, 267].

9.4 Abduction

There has recently been much interest in the applied category theory community in 'lenses', which provide a simple but powerful abstraction for hierarchical feedback. Originally appearing in database theory and functional programming for describing deeply nested destructive updates [91], they have later turned out to be of central interest in categorical approaches to both game theory [108] and machine learning [88, 90]. One perspective on lenses is that they are a general theory of 'things that compose by a chain rule'.

Lenses

Lenses are the morphisms of a category **Lens** whose objects are pairs of sets, where we think of the first as 'forwards' and the second as 'backwards'. A lens

$$\lambda : (X^+, X^-) \rightarrow (Y^+, Y^-)$$

is a pair consisting of a 'forwards' function:

$$\lambda^+ : X^+ \to Y^+$$

and a 'backwards' function:

$$\lambda^- : X^+ \times Y^- \to X^-$$

If we have another lens $\mu : (Y^+, Y^-) \to (Z^+, Z^-)$ then the composite lens has forwards function:

$$(\lambda\mu)^+(x) = \mu^+(\lambda^+(x))$$

while the backwards function is given by the characteristic law

$$(\lambda\mu)^-(x, z) = \lambda^-(x, \mu^-(\lambda^+(x), z))$$

The category of lenses has a monoidal structure, which is given on objects by:

$$(X^+, X^-) \otimes (Y^+, Y^-) = (X^+ \times Y^+, X^- \times Y^-)$$

and on morphisms by:

$$(\lambda \otimes \mu)(x_1, x_2) = (\lambda^+(x_1), \mu^+(x_2))$$
$$(\lambda \otimes \mu)^-((x_1, x_2), (y_1, y_2)) = (\lambda^-(x_1, y_1), \mu^-(x_2, y_2))$$

While this definition is written in terms of sets and functions, it turns out that lenses can be more generally defined over (essentially) any monoidal category, although the correct general definition is not obvious [291].

Examples in Machine Learning

Backpropagation. An important class of lenses consist of a function paired with its first derivative, which is usually known as reverse-mode automatic differentiation. Specifically, let **Smth** be the category whose objects are Euclidean spaces and whose morphisms are differentiable functions. There is a functor $D : \mathbf{Smth} \to \mathbf{Lens}$ which is given on objects by $D(\mathbb{R}^n) = (\mathbb{R}^n, \mathbb{R}^n)$ where we think of the second $\mathbb{R}^n$ as the cotangent space at a point in the first. On morphisms the functor is given by:

$$D(f) = (f, f^\#), \text{ where } f^\#(x, v) = J_f(x)^\top v$$

where J_f is the Jacobian of f. In order for this to be a functor it must be the case that the derivate of a composite function fg can be determined from the derivatives of f and g using the lens composition law. This turns out to be essentially the chain rule: Given composable smooth functions f, g, we have

$$\begin{aligned}
(D(f)D(g))^{-}(x,v) &= f^{\#}(x, g^{\#}(f(x),v)) && \text{(lens composition law)}\\
&= J_f(x)^{\top} J_g(f(x))^{\top} v \\
&= [J_g(f(x))J_f(x)]^{\top} v && \text{(law of matrix transpose)}\\
&= J_{fg}(x)^{\top} v && \text{(multivariate chain rule)}\\
&= (fg)^{\#}(x,v)
\end{aligned}$$

From this it follows that $D(f)D(g) = D(fg)$, i.e., that D is a functor. This connection between lenses and the chain rule was explicitly observed by Zucker [376] and is also implicit in Fong et al. [90].

Variational inference. There is a category whose morphisms $X \to Y$ are probability distributions on Y conditional on X. There are several different ways to make this precise, for example to say that X, Y are measurable spaces and the conditional distribution is modeled as a measurable function $X \to \mathcal{G}(Y)$, where $\mathcal{G}(Y)$ is the measurable space of all probability measures on Y [98, 110]. Equivalently, one might say that objects are finite sets and morphisms are stochastic matrices. Composition of morphisms is by 'integrating out' the middle variable (sometimes called the Chapman-Kolmogorov equation [250]), which is simply matrix multiplication in the finite case. Call this category **Stoch**. There is a morphism **Stoch** $\to$ **Lens** that pairs a conditional distribution with the function that performs Bayesian inversion on it, namely $f \mapsto (f, f^{\#})$ where $f^{\#} : \mathcal{G}(X) \times Y \to \mathcal{G}(X)$ returns the posterior distribution $f^{\#}(\pi, y)$ given a prior π and an observation y. Bayesian inversion satisfies a 'chain rule' with respect to composition, meaning that the Bayesian inverse of a composite conditional distribution can be computed in terms of the Bayesian inverses of the components, and this fact precisely says that **Stoch** $\to$ **Lens** is a functor [318].

Dynamic programming. Consider a Markov chain with state space S, action space A, and (stochastic) transition function $P : S \times A \to S$. Suppose further that actions are controlled by an agent, who obtains utility $U : S \times A \to \mathbb{R}$ on each transition. For each policy $\pi : S \to A$ we obtain a function $f : S \to S$ given by $f(s) = P(s, \pi(s))$, and a function $f^{\#} : S \times \mathbb{R} \to \mathbb{R}$ given by $f^{\#}(s, c) = U(s, \pi(s)) + \gamma c$, where $0 < \gamma < 1$ is a fixed discount factor. The second input to $f^{\#}$ is known as the continuation payoff. These two functions constitute a lens $\lambda_\pi : (S, \mathbb{R}) \to (S, \mathbb{R})$, indexed by the policy. On the other hand, a lens $V : (S, \mathbb{R}) \to (1, 1)$ turns out to be just a function $V : S \to \mathbb{R}$, which we take to be the value function. If V is an initial value function and π is the appropriately optimal policy for it, the lens composition $\lambda_\pi V : (S, \mathbb{R}) \to (1, 1)$ performs a single stage of value function iteration. Thus value function iteration amounts to approximating the limit:

$$\cdots \longrightarrow (S, \mathbb{R}) \xrightarrow{\lambda_{\pi_2}} (S, \mathbb{R}) \xrightarrow{\lambda_{\pi_1}} (S, \mathbb{R}) \xrightarrow{V_0} (1, 1)$$

where each π_i is the optimal policy for the current value function at each stage.[6]

[6] This connection between dynamic programming and lenses is due to Viktor Winschel.

Hierarchical Symbolic-Numeric Lenses

All three examples of the 'lens pattern' we have described above for machine learning are notably 'low-level' and numerical. However, now that the common pattern has been identified, it is possible in principle to design systems which are structurally the same but which are semantically 'higher-level'. This allows the best of both worlds: logical languages embodying GOFAI principles such as abduction and generalization can be combined with the hierarchical feedback which has been enormously successful in numerical and connectionist approaches. One option is to construct a monoidal functor $\mathcal{C} \to \mathbf{Lens}$ where $\mathcal{C}$ is a suitable category for higher-level reasoning; another is to build additional structure into **Lens** itself using its more general definition.

The specific approach proposed here is to construct lenses in which the forwards map performs deductive reasoning, and the backwards map performs abductive reasoning. The idea is that the forwards map λ^+ will, given a hypothesis x, generate a deductive conclusion $\lambda^+(x)$, while the backwards map will, given an *initial* hypothesis x and an observation y, abductively generate an updated hypothesis $\lambda^-(x, y)$ in order to explain the observation in a way that is in some sense 'as close as possible' to the starting hypothesis.

Suppose now that from an initial hypothesis x we make a 2-step deduction $\mu^+(\lambda^+(x))$. If we then observe z, we can perform a 2-step abduction using the lens composition law to determine a new hypothesis. First, using the deduced hypothesis $\lambda^+(x)$ and the observation z, we use μ^- to abductively determine the new 'middle' hypothesis $\mu^-(\lambda^+(x), z)$. We then treat this as though it is an observation, which together with the initial hypothesis x abductively determines the final result:

$$\lambda^-(x, \mu^-(\lambda^+(x), z))$$

Another possibility is that forwards maps perform abstraction between different levels of representation of a state, and backwards maps are control *commands* (or *desires*). We will illustrate this with a simple worked example. Consider a factory robot moving in the region:

$$R = \{(x, y) \in \mathbb{R}^2 \mid 0 \leq x \leq 10, 0 \leq y \leq 10\}$$

Since we will only be describing the robot's movement with pseudocode it suffices to informally describe the map. The factory floor is divided into two *zones*, with a path running through both. In each zone there is a *goal* adjacent to the path, representing a place where the robot can pick up or deliver objects. An abstracted description of the robot's position is given by elements of `Zone` $\times$ `Chunk`, where

$$\texttt{Zone} = \{\texttt{zone1}, \texttt{zone2}\}$$

is the set of zones and

$$\texttt{Chunk} = \{\texttt{path}, \texttt{goal}, \texttt{nothing}\}$$

There is a function $\lambda^{+}_{\text{pos}} : R \to \texttt{Zone} \times \texttt{Chunk}$ that abstracts the robot's position. Going the other way, a command to move to a certain state at the more abstract level can be 'translated down' into a lower-level command to move in the space R. For this we also need to know the current position in the concrete space R, making the type of the backwards function:

$$\lambda^{-}_{\text{pos}} : R \times (\texttt{Chunk} \times \texttt{Zone}) \to R$$

λ^{-}_{pos} need not move the robot directly to a position that satisfies the goal, that is, the equation $\lambda^{+}_{\text{pos}}(\lambda^{-}_{\text{pos}}(x, g)) = g$ (known as the 'put-get law' of lenses) need not always hold. Rather, λ^{-}_{pos} can direct the robot through a series of 'waypoints' by treating the position variable as a state variable. What should be guaranteed is that holding the goal fixed and iterating $\lambda^{-}_{\text{pos}}(-, g) : R \to R$ from any starting position will after finitely many steps reach a position $x \in R$ satisfying $\lambda^{+}_{\text{pos}}(x) = g$ (provided the goal is physically reachable for the robot). For example, our $\lambda^{-}_{\text{pos}}(x, g)$ could be given by the following pseudocode:

- If the current position satisfies the goal ($\lambda^{+}_{\text{pos}}(x) = g$) then do nothing ($\lambda^{-}_{\text{pos}}(x, g) = x$).
- Otherwise, if the current position is within a fixed short distance of the goal, then move onto the center of the goal.
- Otherwise, if the robot is on the path, move along the path towards the goal.
- Otherwise, move directly onto the path.

Thus if we iterate $\lambda^{-}_{\text{pos}}(-, (\texttt{zone1}, \texttt{goal}))$ from a typical starting position then the robot will first move onto the path, then move along the path towards zone 1, and then move onto the goal. Together the functions λ^{+}_{pos} and λ^{-}_{pos} constitute a lens:

$$\lambda_{\text{pos}} : R \to \texttt{Chunk} \times \texttt{Zone}$$

We will now demonstrate how this lens can be a part of a hierarchy in which the next level is task-centric. Suppose the robot can carry an object, from the set:

$$\texttt{Object} = \{\texttt{widget}, \texttt{gizmo}, \texttt{nothing}\}$$

and sense its carry weight. A widget weighs 2, a gizmo weighs 7 and no object weighs 0, defining a function $W : \texttt{Object} \to [0, \infty)$.

We define a lens:

$$\lambda_{\text{wt}} : [0, \infty) \to \texttt{Object}$$

where

$$\lambda^{+}_{\text{wt}} : [0, \infty) \to \texttt{Object}$$

classifies any weight less than 1 as `nothing`, any weight between 1 and 5 as a `widget` and any weight greater than 5 as a `gizmo`. The backward function:

$$\lambda^{-}_{\text{wt}} : [0, \infty) \times \texttt{Object} \to [0, \infty)$$

ignores the current weight, and takes the desired object to its resulting desired weight, namely, $\lambda^{-}_{\text{wt}}(w, o) = W(o)$.

We can run these two lenses in parallel, yielding:

$$\lambda_{\text{pos}} \otimes \lambda_{\text{wt}} : R \times [0, \infty) \to \texttt{Chunk} \times \texttt{Zone} \times \texttt{Object}$$

The parallel composition of lenses

$$\lambda_1 : X_1 \to Y_1 \text{ and } \lambda_2 : X_2 \to Y_2$$

is the lens:

$$\lambda_1 \otimes \lambda_2 : X_1 \times X_2 \to Y_1 \times Y_2$$

given by:

$$v(x_1, x_2) = (v_1(x_1), v_2(x_2)) \text{ and}$$
$$u((x_1, x_2), (y_1, y_2)) = (u_1(x_1, y_1), u_2(x_2, y_2))$$

Unlike the sequential composition defined previously, this is a non-interacting composition.

The second level of the hierarchy will be described as a lens:

$$\lambda_{\text{task}} : \texttt{Chunk} \times \texttt{Zone} \times \texttt{Object} \to \texttt{Task}$$

where

$$\texttt{Task} = \{\texttt{task1}, \texttt{task2}, \texttt{nothing}\}$$

The backwards function takes the current state and the desired task, and returns the desired next state required to complete the task. Task 1 entails collecting a `widget` from the goal in zone 1 and delivering it to the goal in zone 2; task 2 entails collecting a `gizmo` from the goal in zone 2 and delivering it to the goal in zone 1. $\lambda^{-}_{\text{task}}(c, z, o, t)$, which returns the robot's desired next state given the current state and the task, is given by the following pseudocode:

- If the task is 1 and the held object is a `widget`, then proceed to the goal of zone 2 to deliver it:

$$\lambda^{-}_{\text{task}}(c, z, \texttt{widget}, \texttt{task1}) = (\texttt{goal}, \texttt{zone2}, \texttt{nothing})$$

- If the task is 1 and the held object is a `gizmo`, then proceed to the goal of zone 2 to return it:

$$\lambda^{-}_{\text{task}}(c, z, \texttt{gizmo}, \texttt{task1}) = (\texttt{goal}, \texttt{zone2}, \texttt{nothing})$$

- If the task is 1 and no object is held, then proceed to the goal of zone 1 to pick up a `widget`:

$$\lambda^{-}_{\text{task}}(c, z, \texttt{nothing}, \texttt{task1}) = (\texttt{goal}, \texttt{zone1}, \texttt{widget})$$

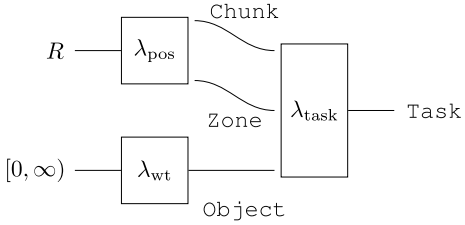
- If the task is 2, then there are three cases similar to the above.
- If there is no task, then remain in the current state:

$$\lambda^{-}_{\text{task}}(c, z, o, \texttt{nothing}) = (c, z, o)$$

We can now compose together the entire control system; it is the lens:

$$(\lambda_{\text{pos}} \otimes \lambda_{\text{wt}})\lambda_{\text{task}} : R \times [0, \infty) \to \texttt{Task}$$

This composite is commonly represented schematically by a 'string diagram' as follows:



The update function of this composite treats the output of $\lambda^{-}_{\text{task}}$, which is the next state desired by the high-level planner, as the input to the first level, which 'translates' it into the lower level of coordinates.

Around this composite system we must place an 'environment'. On the right-hand side sits the human controller or other top-level planner, which decides the top-level tasks given the top-level observations. On the left sits the 'physical environment', consisting of the real world (or a simulation thereof), together with actuators that implement the robot's bottom-level desires and sensors that produce the bottom-level observations. Crucially, this physical environment will typically have an internal state that cannot be observed by the robot.

In our example, for simplicity we take the top-level planner to be a constant task. The physical environment will store the robot's position and currently held object. The current position is continually updated with the desired position provided it is reachable in a straight line. The desired weight is ignored since the robot has no corresponding actuator. When the robot's position is in one of the goal areas, the carried object will change as an object is either picked up or delivered.

If we iterate the composite backward function:

$$((\lambda_{\text{pos}} \otimes \lambda_{\text{wt}})\lambda_{\text{task}})^{-}(-, \texttt{task1})$$

from any starting position, the robot will repeatedly navigate between the goals of `zone1` and `zone2`, picking up and delivering `widgets`. (If it is initially carrying a `gizmo` it will first return the `gizmo` before picking up its first `widget`.)

This setup has the feature that time 'naturally' moves slower the higher one goes up the hierarchy. Suppose the robot's initial position is in `zone2` and it is holding no object. If the task is `task1` then $\lambda^{-}_{\text{task}}$ will output $(\texttt{zone1}, \texttt{goal}, \texttt{widget})$. This will be used as the desired input to λ^{-}_{pos}. The robot will navigate through several stages towards the goal of `zone1`, during which time the output of $\lambda^{-}_{\text{task}}$ will not change. After the robot reaches the goal, the environment will update its held object to a `widget`, which will cause λ^{+}_{wt} to change its output to `widget`. This in turn will finally cause $\lambda^{-}_{\text{task}}$ to change its output to $(\texttt{zone2}, \texttt{goal}, \texttt{nothing})$, signaling a change in desire to move to the other goal to deliver the `widget`. This will again stay constant while the lower level λ_{pos} navigates the robot towards the new goal.

Here we have proposed to found abductive inference on the category-theoretic machinery of lenses. Besides abduction, we have also shown how lenses generalize backpropagation, variational inference, and dynamic programming. We then introduced novel 'symbolic-numeric' lenses, which allows hybrid structures, consisting of both symbols and these pre-existing lenses, to be hierarchically composed. This is important for implementing *scalable planning*: the general planning problem suffers from both branching and time horizon, which can be ameliorated by lower dimensionality as well as longer time jumps. This can be achieved by progressively building a hierarchy of 'concepts' and their affordances (cf. Sect. 11.2.2), and operationalizing planning as abductive reasoning at the highest available level, which, thanks to the hierarchical composition, will still be firmly anchored in the sensorimotor level. In the next chapter, we will see how control loops, which are considered from a lens perspective by emerging research in categorical cybernetics, provide a compositional vocabulary to identify and regulate control systems.