Olog Cookbook
DRAFT February 22, 2022

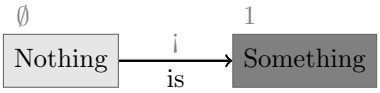
Darrell Lewis-Sandy
Maana, Inc.

February 22, 2022

This paper provides a summary of several design patterns that you might encounter while encoding knowledge in an olog; it is not intended as a primer on Ologs and assumes a basic understanding of types, aspects and facts. For an excellent introduction to ologs, I recommend the paper "OLOGS: A Categorical Framework for Knowledge Representation" by David Spivak.

This paper extends the original olog formalism in two ways: First, we introduce the use of color for identifying various limit and colimit objects. Since these objects are uniquely defined by other types and aspects, we hope that the color cues will be a reminder that these types and their associated universal aspects can be used without requiring endorsement of instances.

Secondly, we extend the notational convention for labeling. Building upon the convention established in the "More Expressive Ologs" section of Spivak's paper, we allow both types and aspects to be assigned a identifying symbol. The symbols are shown in gray, and usually placed above the type or aspect that they are identifying. For example:



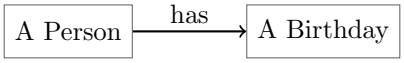
The use of the identifying symbol allows you to unambiguously refer to the same type or aspect in multiple diagrams, and to disambiguate between different drawing elements that might have the same human readable label.

1 Patterns Involving Aspects

1.1 Functional Relationship

An aspect between two types indicates that every element in the source of the aspect is related to an element in the target of the aspect. For example, the

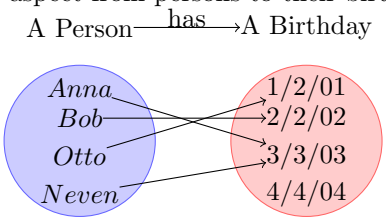
statement that "every person has a birthday" could be represented as a single aspect between person and date:



By using an aspect, you are specifying that the relation between the two types satisfies two important properties:

1. Given an element in the source of an aspect, there is a corresponding element in the target of the aspect.
2. Given an element in the source of an aspect, there is not more than one corresponding element in the target of the aspect.

To illustrate these properties, consider the one possible instance graph of the aspect from persons to their birthday:

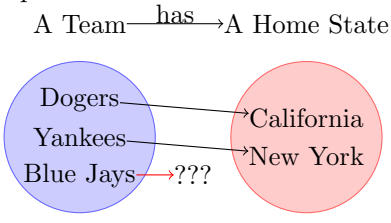


As indicated, each person is related to a date, and no person is related to more than one date.

Examples of statements that cannot be represented as aspects include:

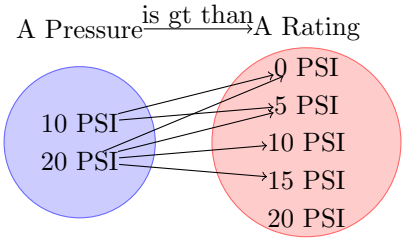
- "Some baseball teams have a home state"
- "The pressure rating is greater than a number"

Although the first statement is true, it cannot be expressed as an aspect because there are some baseball teams (e.g., Toronto Blue Jays, Washington Nationals) that do not have home states. The relation between baseball teams and home states is what is called a partial function because it only relates part of the source type to the target type. Partial functions cannot be represented as aspects.



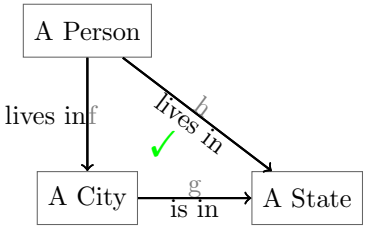
The second statement, although it is probably true, cannot be be expressed as an aspect because there are probably many (possibly infinite) numbers that

any given pressure rating is greater than. The relation between pressure ratings and the numbers that they are greater than is what is called a multiple (or non-functional) relation because it relates each element of the source to one or more elements in the target type. Multiple relations cannot be represented as aspects.



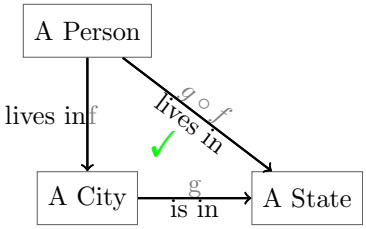
1.2 Composition

Sometimes you would like to create a new aspect that is the equivalent to a path. For example, given an olog that has an aspect that represents a person's hometown, and an aspect that relates a city to the state that it is in, you might want to create a new aspect that represents the relationship between a person and their home state. You could accomplish this by explicitly adding the new aspect from person to state and creating a fact to enforce the equivalence:



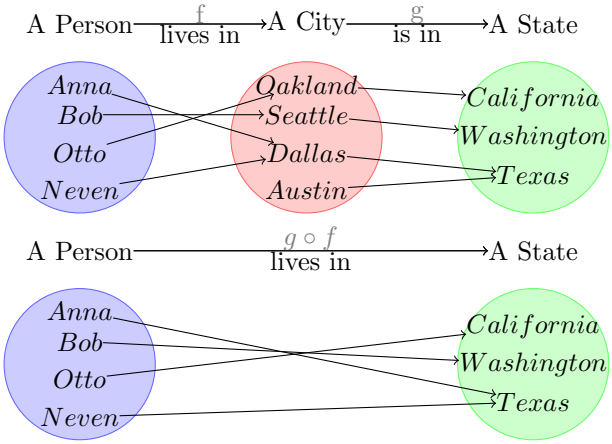
This is somewhat undesirable; by creating an aspect in this way, a subject matter expert must endorse instances for that aspect and ensure that it meets the requirements of the fact. However, the meaning of aspect h is uniquely defined by the instances that are endorsed for f and g.

Composite aspects solve this problem. Given any pair of aspects in a path, $A \xrightarrow{f} B \xrightarrow{g} C$ you can define a unique aspect $g \circ f$, called the composite. The composite represents the result of applying g after applying f, and automatically satisfies the fact in the diagram below:



By the composite of two aspects, the fact is implicitly understood, and does not need to explicitly stated.

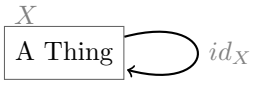
To illustrate the composition of two aspects, consider the instance graph below:



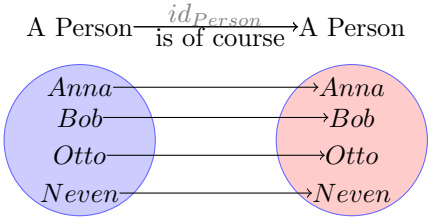
The primary value of using the composite of aspects is that the instances can be automatically inferred from its component aspects, thereby eliminating some of the burden in endorsement. You should consider using composite aspects whenever you are intending to provide a name for an existing path in your olog.

1.3 The Identity Aspect

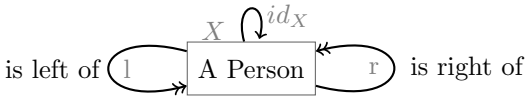
Every type in an olog has a special aspect, called the identity, which has that type as both its source and its target:



The identity aspect maps each value of its type back to exactly the same value. The instance diagram below depicts an identity aspect on people:



The identity aspect can be very useful in declaring facts. For example, consider the case where you are describing the seating arrangement of guests sitting around a table using the following olog:



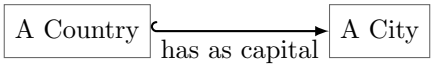
$$id_X = l \circ r$$
$$id_X = r \circ l$$

The aspects r and l map a person to the person sitting on their left, and on their right respectively. Both of the facts of the olog make use of the identity aspect on person to capture the relationship between the people at the table. Specifically, the facts capture the idea that every person is sitting to the right/left of the person sitting to the left/right of them is themselves.

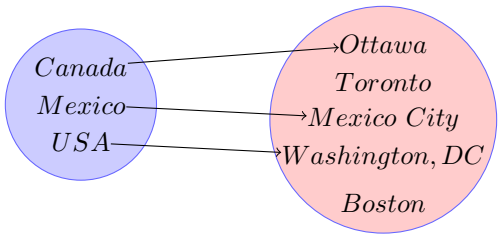
1.4 Injective Aspects

Sometimes you would like to capture the belief that an aspect must map each element of its source to a different value in its target. For example, an aspect that maps countries to their capital cities must ensure that no two countries have the same capital.

An injective aspect is a special kind of aspect that can be used to denote the distinctness of the mapped values. An injective aspect is drawn using an arrow with a hooked tail:



A possible instance diagram for this olog is depicted below:



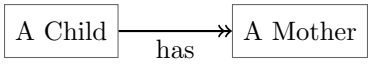
As indicated, each country has a different capital city.

An equivalent way of thinking about injective aspects is that they provide proof that the target of the aspect has at least as many elements as the source.

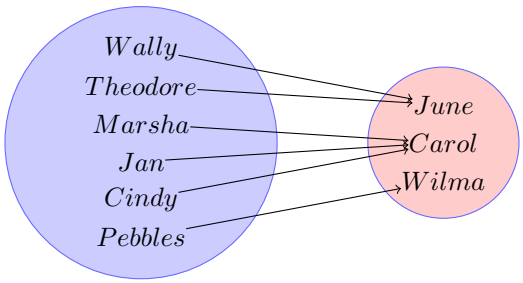
1.5 Surjective Aspects

Sometimes you would like to capture the belief that every value in the target of an aspect is related to some value in the source. For example, if you believe that a mother is a person that has at least one child, then an aspect that maps children to their mother must map at least one child to each mother.

A surjective aspect is a special kind of aspect, that can be used to denote that an aspect covers all the possible values of its target. A surjection is drawn using an arrow with two heads:



A possible instance diagram for this olog is depicted below:

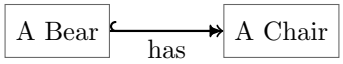


As indicated, each child has a mother, and each mother has at least one child.

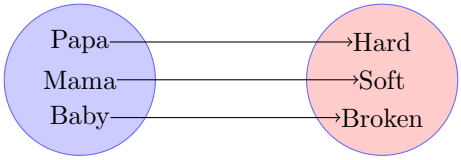
An equivalent way of thinking about surjective aspects is that they provide proof that the target of the aspect does not have more elements than the source.

1.6 Bijective Aspects

Sometimes you want to capture the belief that an aspect puts each element of its source and target into a unique correspondence with each other. That is each value in the target of the aspect is related to a different element of the source. For example, you might use a aspect to relate the bears in the classic children's story "Goldilocks and the three bears" to the chairs that belong to them:



A possible instance diagram for this olog is depicted below:

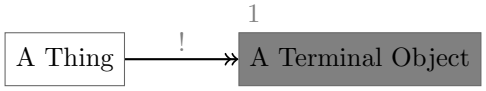


as indicated, each bear is related to a different chair, and each chair is related to a different bear.

An equivalent way of thinking about bijective aspects is that they provide proof that the source and target of an aspect have the same number of values.

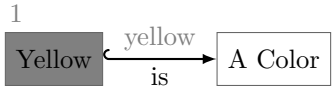
2 Patterns Involving the Terminal Object

A terminal object generalizes on the concept of singleton sets. There is always exactly one way to map any type to a terminal object. The terminal object is denoted 1, and the unique map from a type !:



2.1 Points

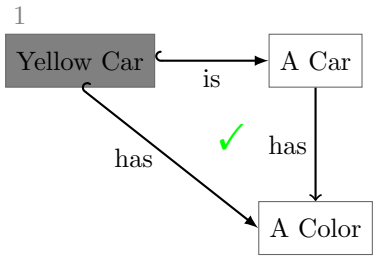
Aspects from the terminal object can be used to label (or point to) elements of a type. For example the olog:



states that "yellow is a color". To improve readability, we have followed the convention established by Spivak and named the terminal object's box with the name of the instance "Yellow". To make clear that this type is the terminal object, we have labeled it with the symbol "1".

2.2 Existential Quantification

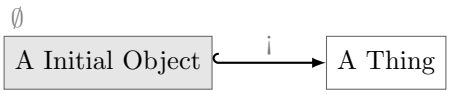
By declaring an aspect from the terminal object to another type, you are declaring that there is an element in that type. You can create a fact using two points that captures the concept of existential quantification:



Loosely translated, this olog reads "There exists at least one car which is yellow". In order to reify the olog, the subject mater expert must provide proof that there is a yellow car by endorsing instances so that the two paths of the fact commute.

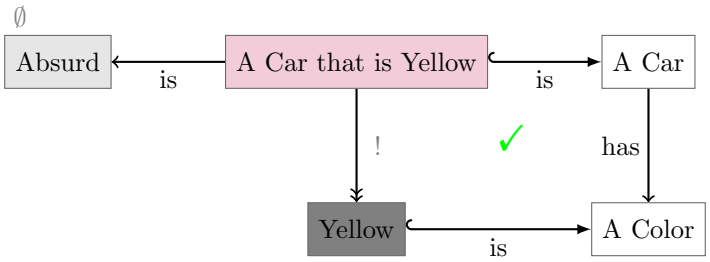
3 Patterns Involving the Initial Object

A initial object generalizes on the concept of empty sets. There is always exactly one way to map from the initial object to any type. The initial object is denoted $\emptyset$, and the unique map to a type ¡:



3.1 Non-Existence

In the category of Sets and Functions, any map from the initial object is (trivially) surjective. You can use this to construct an olog that captures the idea that a type has no instances. For example:

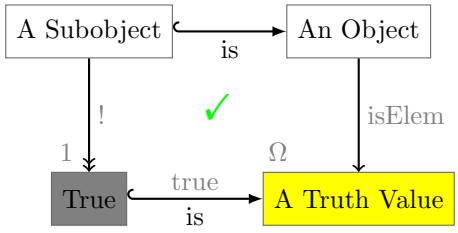


The pullback "A Car that is Yellow" is the subtype of cars that are yellow (more on this pattern in section 7). Loosely translated, this olog reads "The existence of a car that is yellow is absurd.". In order to reify the olog, the subject mater expert must provide proof that there is not a yellow car by endorsing instances x, $yellow$, and $colorOf$ so that the two paths of the fact commute and for which the pullback is empty.

Note that this pattern does not generalize to categories other than SET. It is preferable to use the universal quantification pattern in section 4.

4 Patterns Involving the Subobject Classifier

The subobject classifier is a type that generalizes the concept of truth values. The subobject classifier is a type, denoted "Ω", and an aspect "true" such that for any injective aspect, there exists a unique aspect that makes the fact of the olog below true:

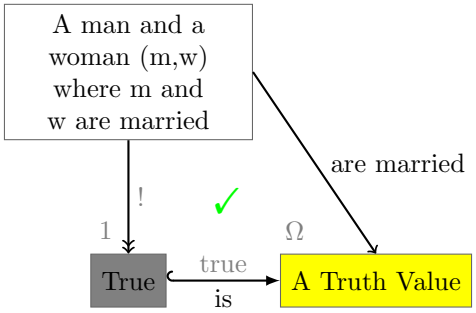


In this diagram, the function $isElem$ is a characteristic aspect that returns a value of true only for objects that are also subobjects. To make clear that an type is the subobject classifier, we labeled it with the symbol "Ω".

Unlike classical logic truth values, the subobject classifier may contain additional structure besides the element true.

4.1 Universally Quantified Predicates as Facts

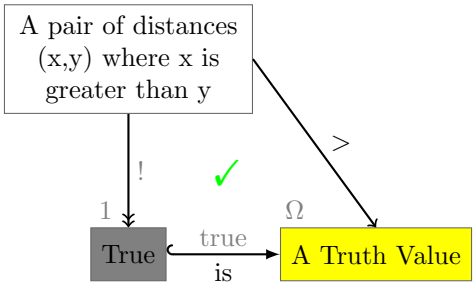
The subobject classifier can be used indicate that every element in a type satisfies a predicate. For example:



Here the predicate "are married" is an aspect which returns true only when the given pair is married. The path which factors through the terminal object takes every pair to the truth value "true". The fact equates those two paths and can only be valid when the pair are married.

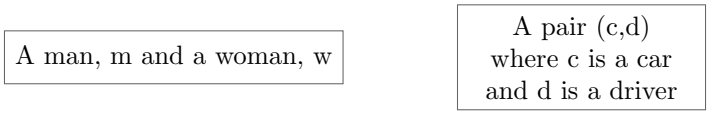
4.2 Comparisons of Size

The relational operators ($<$, $>$, $\geq$, and $\leq$) can be considered predicates which take a pair of values and return true when the inequality holds. You can universally quantify statements of size in the same way that you would any other predicate. For example:



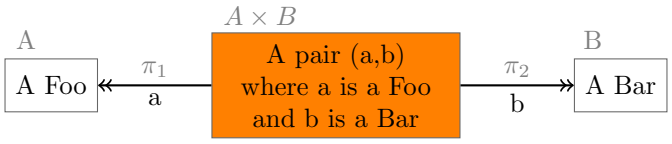
5 Patterns Involving Products

In section 2.1.1 of Spivak's paper he describes types with compound structure, for example:



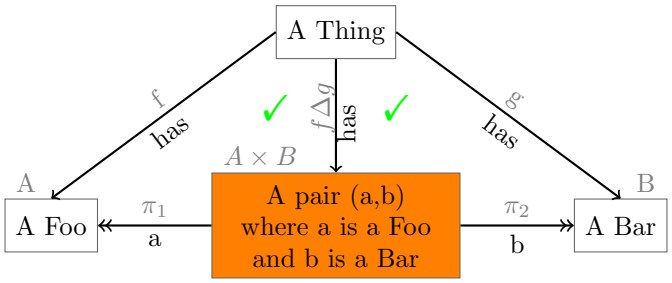
Each instance of these compound types must contain a pair, however it is up to an expert to select which instances are included. In the case of the first type, the pairs might represent married couples, dance partners in a competition, or mothers and their oldest sons. Sometimes you need to ensure that a type contains all possible tuples of two or more types; for this you need the categorical product.

The categorical product generalizes on the concept of compound types. A product of of two types A, and B, is a type $A \times B$. It contains *all the possible pairs* that can be constructed so that the first element from type A, and the second element is taken from type B. Every product also comes with a pair of aspects ($A \times B \xrightarrow{\pi_1} A$, and $A \times B \xrightarrow{\pi_2} B$) which project the first and second elements of the product:



5.1 Universal Aspect of Products

Products have a universal aspect. Given any type X and any pair of aspects $X \xrightarrow{f} A$ and $X \xrightarrow{g} B$, then there is a unique aspect $X \xrightarrow{f \Delta g} A \times B$ that the facts in the following olog hold:

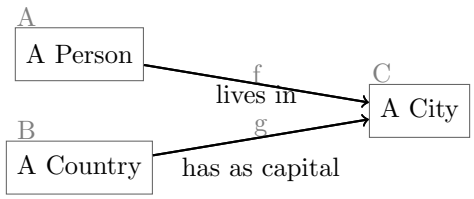


The universal aspect $X \xrightarrow{f \Delta g} A \times B$ takes each element, x that is in X to the pair (f(x), g(x)).

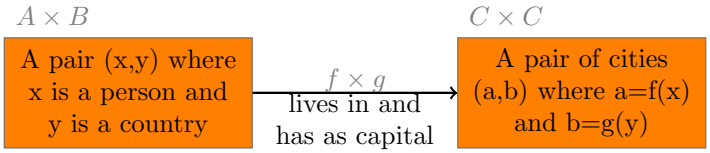
Whenever you use the universal aspect of a product, the fact is implicitly understood and does not need to be explicitly stated on the olog.

5.2 Applying aspects to the elements of pairs

Sometimes you would like to apply a different function to each element of a pair, to obtain a new pair. For example, if I had a pair of aspects, :



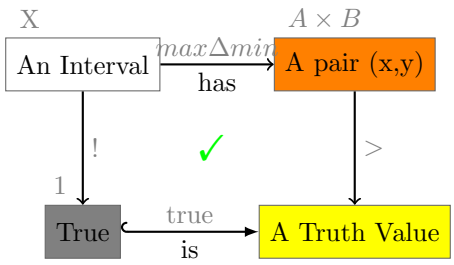
I might want to an aspect that, given a pair (x,y) where x is a person and y is a city, returns a new pair (f(x), g(x)).



In general, I can take any pair of aspects, and create a new aspect $f \times g$ is called the "cross" of f and g. It is uniquely determined by the functions f and g and satisfies the following path equivalence: $f \times g = (f \circ \pi_1)\Delta(g \circ \pi_2)$. By using the cross function, this fact is implicitly understood, and need not be explicitly stated.

5.3 Comparing Pairs of Attributes

The universal aspect of products can be used to compare pairs of attributes of the same type. For example, you might want to say that "Every interval has a maximum and a minimum where the maximum is greater than the minimum". This can be represented as the following olog:

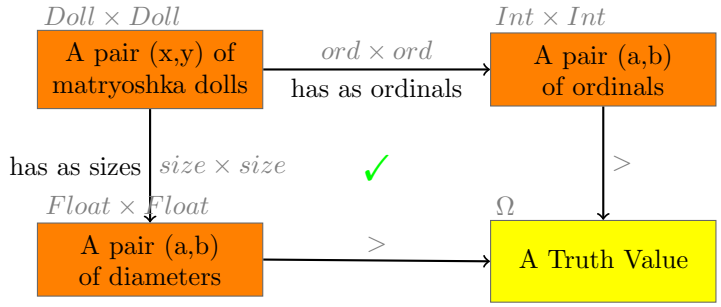


5.4 Comparing Monotone Properties

Sometimes the properties of an object are related to where it is in a sequence. For example, Russian matryoshka dolls are a traditional wooden toy comprised of a series of carved wooden dolls that fit one inside the other. You could number the dolls in order from smallest to largest. A basic olog that captures the belief that each doll has an ordinal and a size is depicted below:



You can extend this olog to capture the relationship between the ordinal (number in the sequence) and the diameter of the dolls in the following olog:



The use of the product (Dolls x Dolls) in the upper right universally quantifies the fact over all possible pairs of dolls. The path along the top tests if the ordinal for the first doll is greater than the ordinal for the second where $ord \times ord$ maps the pair of dolls to their pair of ordinals. The right path performs the same test for the sizes of the dolls. By equating the two paths, weyou encode a fact that might read "Given any two dolls, whenever the ordinal of the first is greater than the second, then the size of the first is also greater than the second".

5.5 Larger Products

Products can be extended to more than two types to generalize the concept of triples, quadruples, ... of values. For example $A \times B \times C \times D$ would be the product of four types. Products of more than two types will have the same number of projections $(\pi_1, \pi_2, \pi_3, ...)$ to access each element by its position in the tuple.

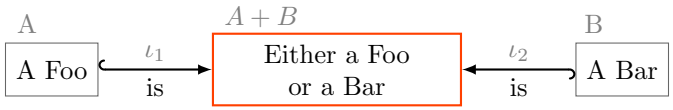
6 Patterns Involving Coproducts

Sometimes you would like to express an idea which involves an option between two or more types. For example:

- The colors of the rainbow are red, orange, yellow, green and blue.
- An animal, vegetable or mineral.
- A queue is either empty, or it is a person standing at the head of a (shorter) queue

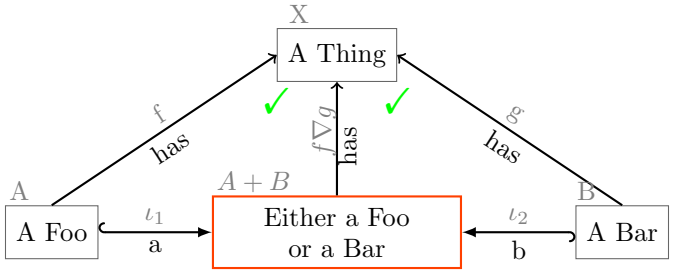
In each of these cases, you are presented with a type (colors, equipment, and queue) and an exhaustive collection of different things that could be members of the type.

The coproduct generalizes the concept of unions of options. A coproduct of two types A, and B, is a type $A + B$. It contains all the possible values of A, and all the possible values of B. Every coproduct also comes with a pair of aspects ($A \xrightarrow{\iota_1} A + B$, and $B \xrightarrow{\iota_1} A + B$) which inject a values of A and B into the coproduct respectively:



6.1 Universal Aspect of Coproducts

Coproducts also have the universal aspect, that given any type X and any pair of aspects $A \xrightarrow{f} X$ and $B \xrightarrow{g} X$, then there is a unique aspect $A + B \xrightarrow{f \nabla g} X$ that makes facts in the following olog hold:

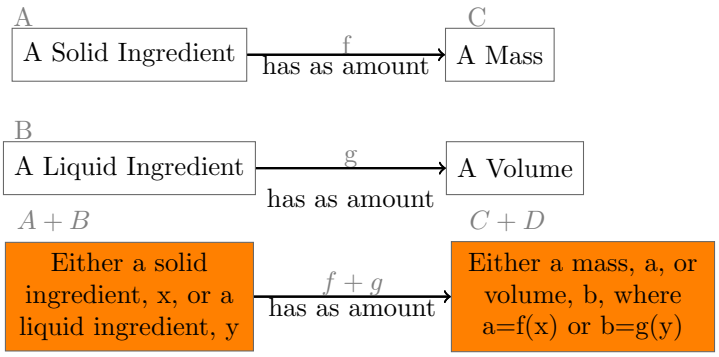


where the universal aspect $A + B \xrightarrow{f \nabla g} X$ applies f to an instance if it is in A, and g if it is in B.

Whenever you use the universal aspect of a coproduct, the fact is implicitly understood and does not need to be explicitly stated on the olog.

6.2 Applying aspects to case of a coproduct

Sometimes you would like to apply a different function to each case of a coproduct, to obtain a new coproduct. For example if you wish to represent an aspect from ingredients in a recipe to their amounts, where the quantities are measured in units of volume if the ingredient is a liquid, and units of mass otherwise. You might create the following olog:



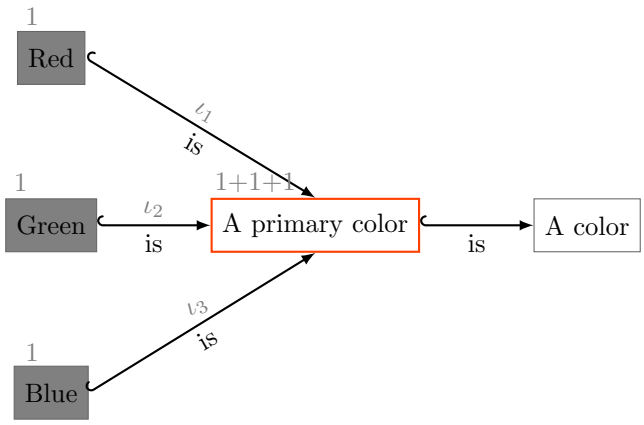
The aspect, $f+g$ is called the "cocross" of f and g. It is uniquely determined by the functions f and g, and satisfies the following path equivalence: $f + g = (\iota_1 \circ f)\nabla(\iota_2 \circ g)$. By using the cocross function, this fact is implicitly understood, and need not be explicitly stated.

6.3 Larger coproducts

Like products, coproducts can be extended to more than two types. For example $A + B + C + D$ would be the type where each instance is either an element of A, B, C or D. Coproducts of more than two types will have the same number of injections as types in the sum $(\iota_1, \iota_2, \iota_3, ...)$.

6.4 Enumerations

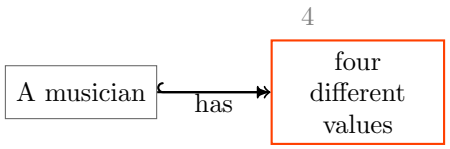
Sometimes you would like to create a type that contains only specific named instances, for example: red, green and blue are the primary colors. This can be achieved using the olog below:



By using the coproduct (1+1+1) you ensure that red, green and blue are the only primary colors, and that they are all distinguishable from each other. By the use of the injective arrow between "A primary color" and "A color" you ensure that the primary colors also map to distinct colors.

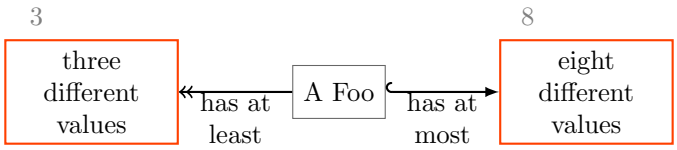
6.5 Types with A Specific Number Of Elements

Sometimes you would like to ensure that a type has a specific number of distinct values. For example, you would like to ensure that the number of musicians in a quartet is exactly four. From section 1, you know that you can use a bijective aspect to create a correspondence with a set containing the desired number of elements:



Where 4 is a finite set with exactly four elements in it. The finite set for any cardinal number can be constructed from the coproduct of multiple copies of the terminal object. You can indicate that you are using a finite set by labeling your type with a cardinal number. Each finite set is equipped with injections one,two,.. etc to make referring to its elements easier.

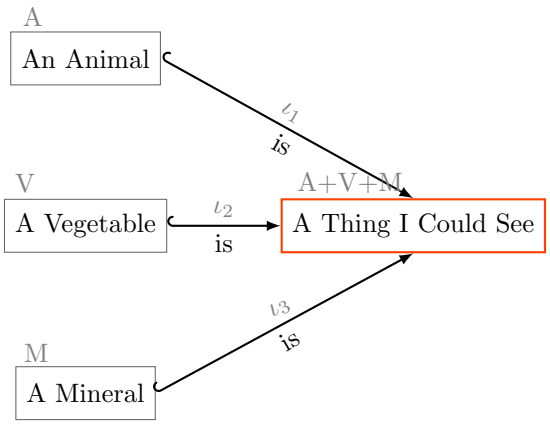
You can also construct ologs with finite sets and injections and surjections to indicate that type has a minimum or maximum number of distinct values. For example the olog below:



captures the belief that there are between 3 and 8 possible values for the type Foo.

6.6 Union Types

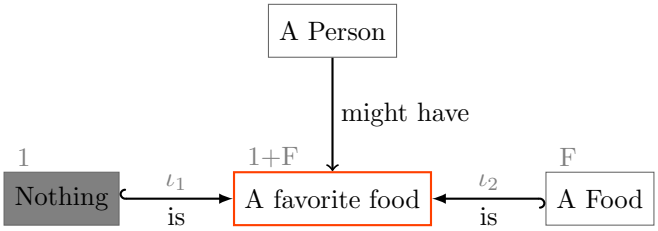
Consider the case where modeling a popular children's game, and you would like to create a type that represents things that you could see. You might construct the olog below:



Structurally this olog looks very similar to the example in the previous section. The main difference here is that the types "An Animal", "A Vegetable" and "A Mineral" need not be the terminal object – there may be many instances of each. The use of the coproduct ensures that animals, vegetables and minerals are all distinct things that I could see, and they are all distinguishable from each other.

6.7 Optional Values

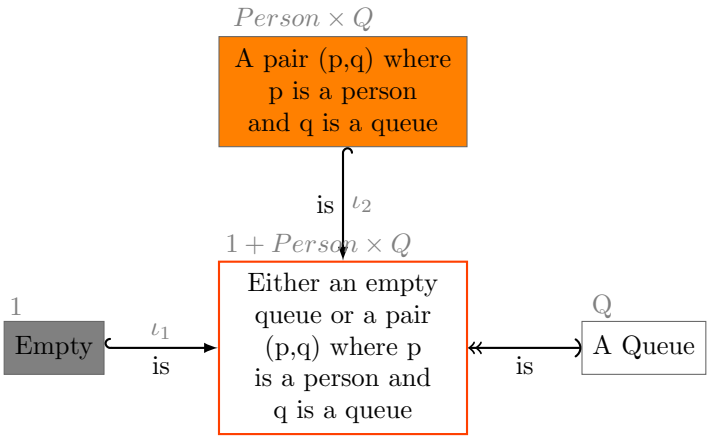
Sometimes you would like to create an aspect between two types the expresses the concept of "A might have a B". This can be achieved using the pattern below:



By use of the coproduct, you are able to construct a type that has as instances, all possible foods and one additional distinct and distinguishable element that represents nothing. The aspect "might have" maps each person to either their favorite food, or to the element representing nothing.

6.8 Recursive Types

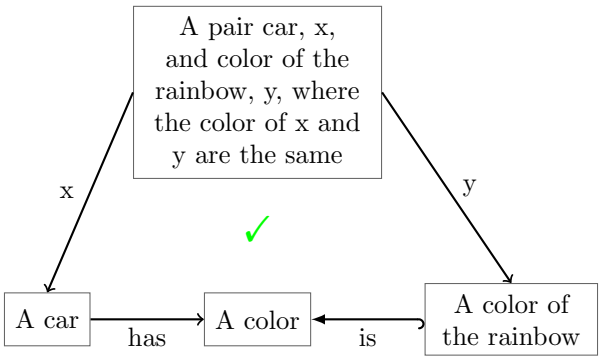
In the introduction to this section, I used the example of a queue being "either an empty queue, or a person standing at the head of a queue". This can be made explicit using coproducts:



The use of both the product and the coproduct in this olog allows us to describe the type queue in terms of itself. The magic of this olog is the aspect from "A Queue" into the coproduct which is both injective and surjective. It "ties" the recursive knot implied in the prose description of a queue.

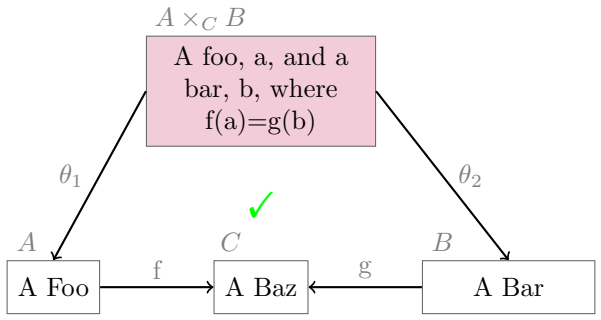
7 Patterns Involving Pullbacks

In the previous sections, you have used facts to ensure that any instances of a type must satisfy a path equivalence relation. For example the statements below:



The problem with this representation is that you frequently want all the possible pairs that satisfy the fact, not just some of them. As diagramed, an expert can choose to endorse any subset of the pairs which satisfy the fact, (including none at all).

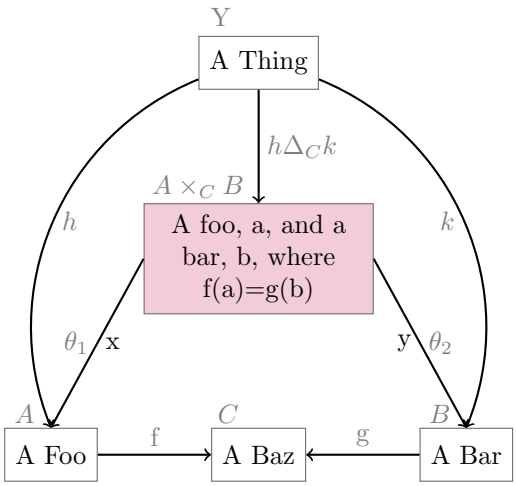
Pullbacks generalize the concept of a pairs that satisfy a fact. Specifically, a pullback of two aspects $A \xrightarrow{f} C$ and $B \xrightarrow{g} C$ is a new type which you represent with the symbol $A \times_X B$. The pullback is the largest subset of the product of A and B that makes the path $A \times_C B \xrightarrow{\theta_1} A \xrightarrow{f} C$ equal $A \times_C B \xrightarrow{\theta_2} B \xrightarrow{g} C$, where θ_1 and θ_2 are the projection functions for the elements of the pullback.



Whenever you use the pullback or its projections, the fact is implicitly understood, and need not be indicated on the olog.

7.1 Universal Aspect of Pullbacks

Pullbacks also have the universal aspect. Given any fact that equates two paths $Y \xrightarrow{h} A \xrightarrow{f} C$ and $Y \xrightarrow{k} B \xrightarrow{g} C$, then there is a unique aspect from Y to $A \times_C B$ such that the facts in the following olog hold:



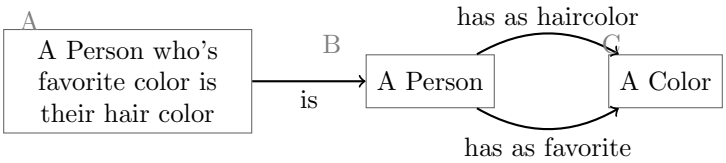
$$f \circ h = g \circ k$$
$$h = \theta_1 \circ (h\Delta_C k)$$
$$k = \theta_2 \circ (h\Delta_C k)$$

The universal aspect takes each element y of Y to the element (h(y),k(y)) in the pullback.

Whenever you use the universal aspect of a pullback, the facts are implicitly understood, and do not need to be indicated on the olog.

8 Patterns Involving Equalizers

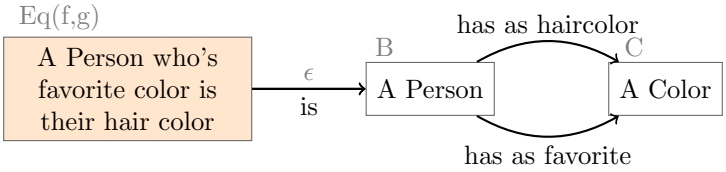
Sometimes you want a subset of a type which satisfies a fact. This could be captured with the pattern:



has as haircolor $\circ$ is = has as favorite $\circ$ is

The problem with this representation is that you frequently want persons that satisfy the fact, not just some of them. As diagramed, an expert can choose to endorse any subset people which satisfy the fact, (including none at all).

Equalizers generalize the subset of a type that satisfy a fact. Specifically, an equalizer of two aspects $B \xrightarrow{f} C$ and $B \xrightarrow{g} C$ is a new type which you represent with the symbol $Eq(f,g)$. The equalizer is the largest subset of the B that makes the path $Eq(f,g) \xrightarrow{\epsilon} B \xrightarrow{f} C$ equal $Eq(f,g) \xrightarrow{\epsilon} B \xrightarrow{g} C$, where $Eq(f,g) \xrightarrow{\epsilon} B$ maps each element of the equalizer to the corresponding element of B.

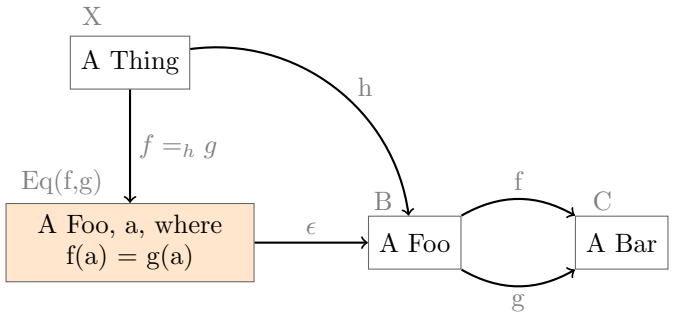


has as haircolor ∘ is = has as favorite ∘ is

When you use the equalizer, the fact is implicitly understood, and does not need to be explicitly indicated on the olog.

8.1 Universal Aspect of Equalizers

Equalizers have a universal aspect. Given any type X and any aspect $X \xrightarrow{h} B$ such that the paths $X \xrightarrow{h} B \xrightarrow{f} C$ and $X \xrightarrow{f} B \xrightarrow{g} C$ are equal, then there is a unique aspect $X \xrightarrow{f=_h g} B$ such that the facts of the following olog are satisfied:

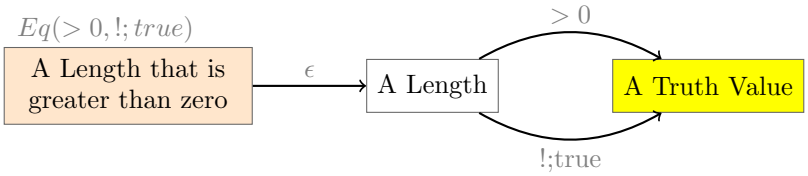


$$h = \epsilon \circ (f =_h g)$$

Whenever you use the universal aspect of an equalizer, the facts are implicitly understood, and do not need to be indicated on the olog.

8.2 Universally Quantified Predicates using Equalizers

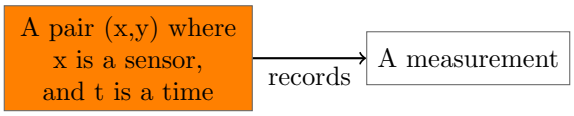
You can extend the pattern that you used in section 4.1 to construct the largest possible subset of a type that satisfies a predicate. For example:



Unlike section 4.1, the use of the equalizer ensures that you have the largest subset of lengths that satisfy the predicate (> 0). No additional facts other than those implied by the use of the equalizer are required.

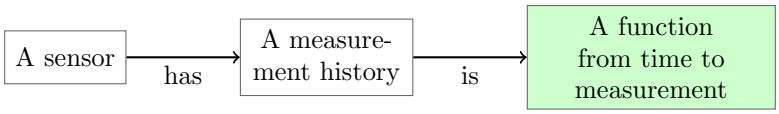
9 Patterns Involving Exponentials

Sometimes the most natural way to encode a property of a type is as a function. For example temperature sensor should have a relationship between time and the temperature that was measured. Using products, you could represent this as an olog:



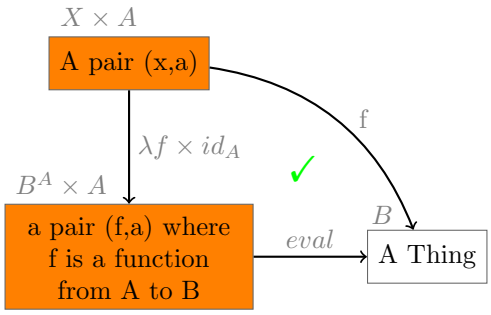
This is somewhat unsatisfactory, because you would like to talk about the history of a sensor being an aspect of the sensor.

Exponentials generalize the concept of function spaces. Given pair of types A and B an exponential object is the set of all possible functions from A to B. The exponential object is denoted by B^A and has a unique aspect $B^A \times A \overset{eval}{\to} B$ that takes each (f,x) of the source type returns f(x). With exponentials, you can capture the idea of a sensor and its measurement history in an olog:



9.1 Universal Aspect of Exponentials

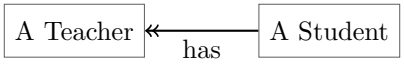
Exponentials have a universal aspect. Given any type X and any aspect $X \times A \overset{f}{\to} B$ there is a unique aspect $X \overset{\lambda f}{\to} B^A$ such that the facts of the following olog are satisfied:



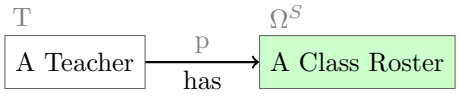
Whenever you use the exponential, the evaluation function its universal aspect, the facts are understood, and do not have to be explicitly stated in the olog.

9.2 Unordered Collections as Exponentials

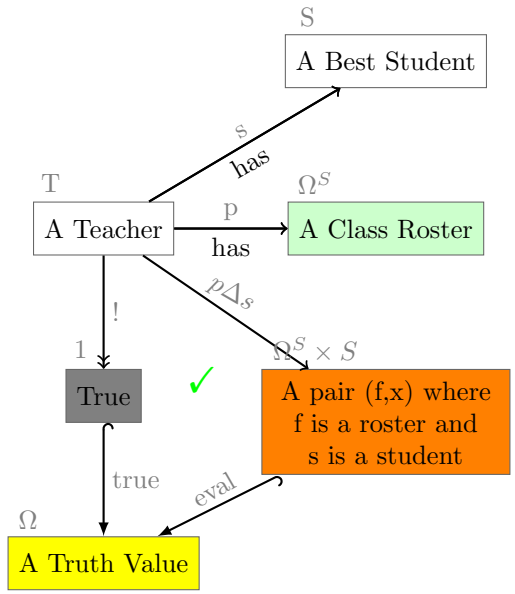
Sometimes you would like to capture the idea that a type has a collection of objects associated with it. For example, "A Teacher has a collection of students". You could represent this using the olog below:



This is somewhat undesirable because a student may have more than one teacher. Using an exponential, you can give every teacher a predicate that membership test that is true for each of their students:



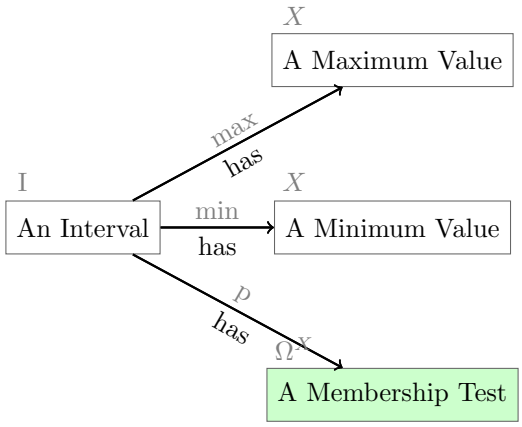
If you want to ensure that each teacher has a student, you may need to introduce some additional machinery. For example, you could give each teacher an aspect to their top student, and then introduce an fact that ensures that the student is in the class:



The left path of the fact maps every possible teacher to the truth value true; the right hand path applies the class membership function to the teacher's top student. Requiring the two paths to be equal is equivalent to requiring that the teacher have at least one student.

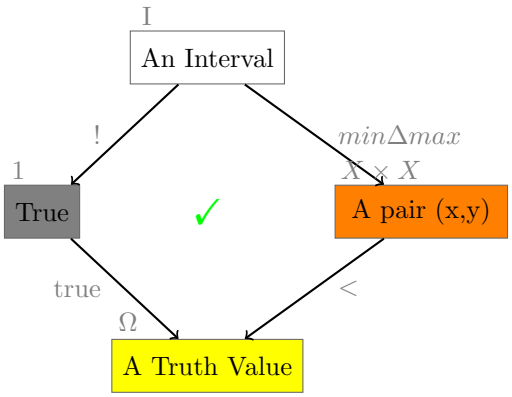
9.3 Intervals

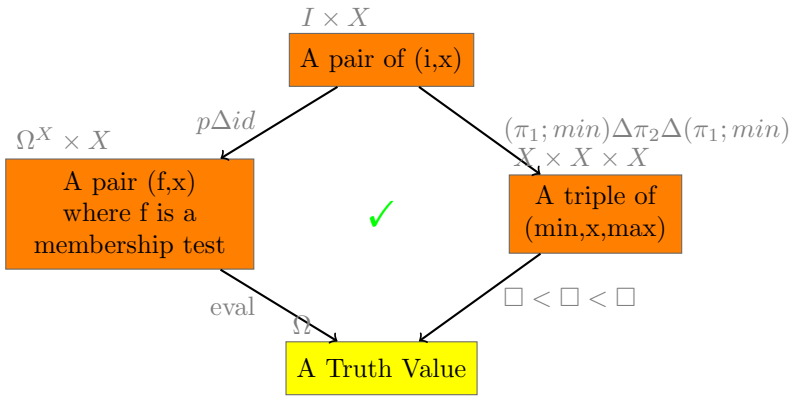
Modeling an interval (A range of values between a maximum and a minimum) can be performed with the help of exponentials. You start by declaring an interval to be a type which has a maximum and a minimum value, and a membership function:



Next you need to introduce some facts to make sure that everything makes sense:

from top to bottom, the facts ensure that (1) the maximum value is greater than the minimum, (2) all members of the interval are between the maximum and minimum value.





For the first fact, the left path is a constant function to the value "true". The rightmost path compares the min and max value for the interval and returns true if the min is less than the max. Equating the paths amounts to stating that for any interval the maximum is greater than the minimum.

For the second fact, the rightmost path uses the membership test to evaluate if a value is in the interval. The rightmost path compares the same value to the minimum and maximum and returns true only if falls between the bounds. Equating the two paths ensures that the membership function is true if and only if a value falls between the maximum and minimum bounds of the interval