Knowledge representation in AI

Yan King Yin general.intelligence@gmail.com

October 22, 2018

Contents

Abstract

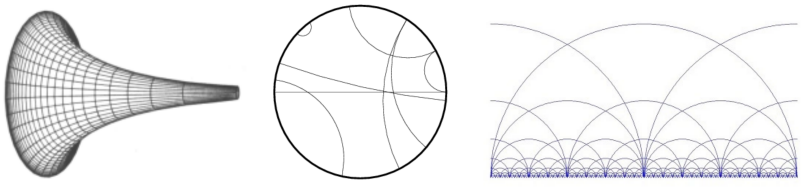
At the top level, an AI can be construed as a dynamical system moving in "cognitive space", trying to minimize a loss function (or maximize a reward). The control of such a system follows the Hamilton-Jacobi-Bellman equation. This framework of reinforcement learning does not offer sufficient structure to solve the AI problem effectively, but it provides a perspective from which subsequent development is possible.

The salient feature of human intelligence is reasoning ability, which is distilled as the theory of logic. We need to impose this logical structure on the AI system (ie, inductive bias). Topos theory provides the perfect abstraction tool to do this. Recently, Google / DeepMind proposed a reasoning architecture based on a neural network acting upon a graph memory. With our abstraction we may be able to improve upon the graph model with simpler and perhaps more efficient "geometric models". The categorical perspective makes it easier to see the problem's structure.

The abstract formulation of AI may lead to better variations. In this paper we look at 3 approaches, based on: A) genetic algorithm; B) neural networks; C) geometric models. This paper is written in a tutorial style as the mathematical background may be unfamiliar to many AI practitioners or researchers.

What is model theory?

For example, **hyperbolic geometry** can be "realized" by the following **models**:

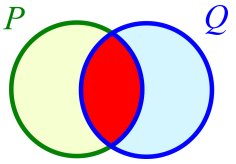


(1)

Models are not unique, there can be many models for a theory.

In mathematical logic, **model theory** studies the **duality** between **syntax** and **models**.

The most classic example is **Stone duality**, also familiar to all of us as "Venn diagrams":

$$P \wedge Q \quad \cong \quad \text{  } \tag{2}$$

Stone duality* refers to the duality between **Boolean algebras** and **topological spaces**.

Boolean algebra is the same as **propositional logic**, which concerns only with the truth and falsehood of propositions. For example, P = *"It is raining in New York"*, but we cannot "access" the internal constituents of the propositioon, such as *"rain"* and *"New York"*. The critical obstruction to strong AI is the **lifting** from propositional to first-order logic.

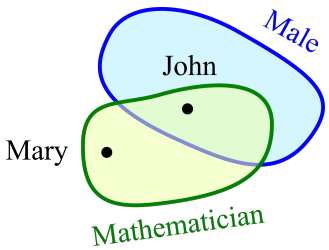
Roughly speaking, classical logic consists of 2 key notions:

- **propositions** are things that can be assigned **truth values**
- each proposition is a **relation** between **objects**

By an informal argument, any idea that is *expressible* in natural language, must be structurally essentially equivalent to predicate logic, which we know from the study of linguistics. So there is no *concievable* way for us to invent an alternative logic that is drastically different from classical logic, with the exception of relatively simple syntactic transformations.

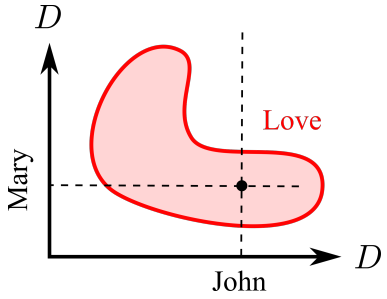
Alan Turing (1912-1954), being very ahead of his time, **circumcribed** the problem of AI by formulating the form of all computable functions, ie, Turing machines, later shown to be equivalent to λ-calculus and combinatory logic. Figure (3) shows the family of major logical structures.

First-order logic can be modeled by **sets** and their **elements**. For example
John $\in$ Male, John, Mary $\in$ Mathematician



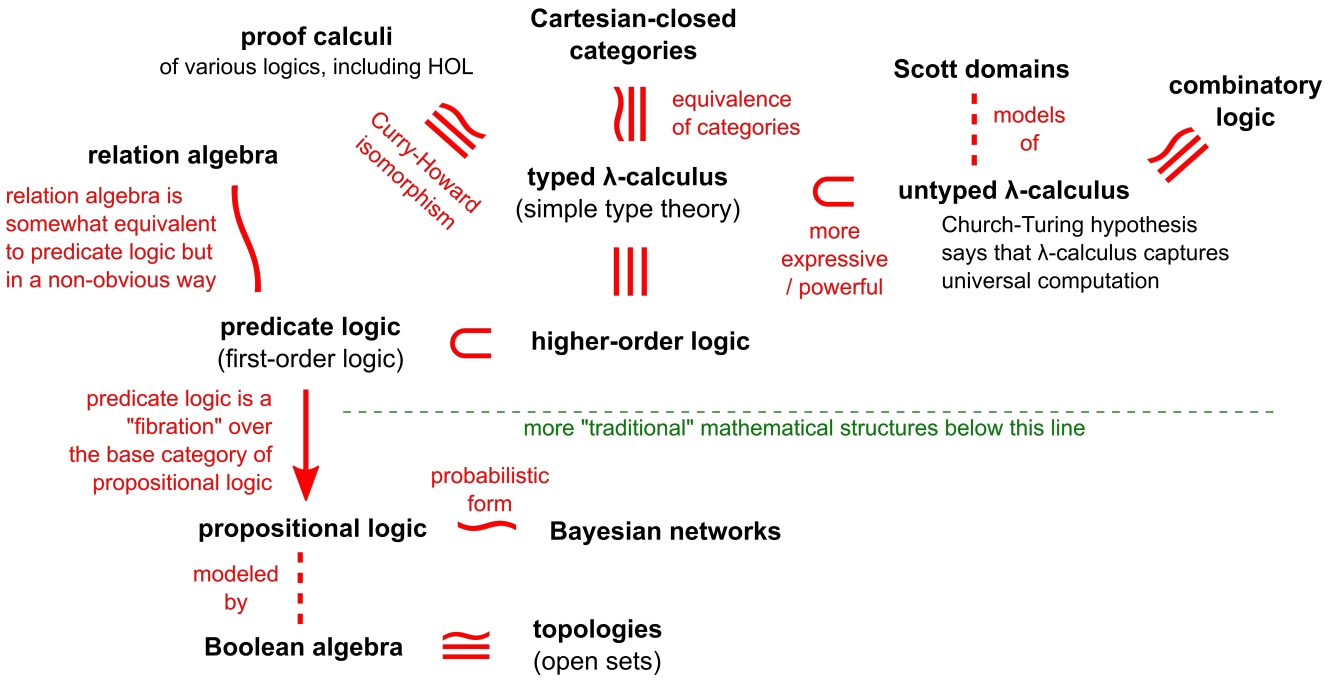
(4)

Whereas **relations** between first-order **objects** in a domain D are represented by **subsets** of the Cartesian product $D \times D$, eg:



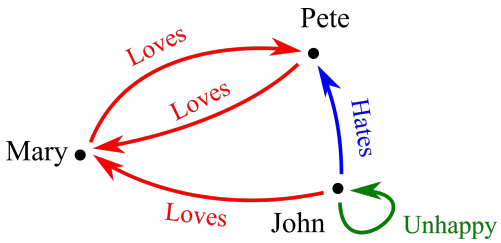
(5)

*Marshall Stone (1903-1989), American mathematician who contributed to real analysis, functional analysis, topology and the study of Boolean algebras.



(3)

For computer science people, you may be more familiar with **relation graphs** or **knowledge graphs** such as this one:



(6)

This is a **directed multi-graph**, or **quiver**. Quiver is an important structure in algebraic representation theory. The category of quivers $\mathscr{Q}$ is a **topos** (this will be introduced below), basically a topos is a structure that has the requirements to be a first-order logic **model**.

According to [Grilliette 2017], a hyper-graph is not a topos, a multi-graph is not a topos, but their **directed** versions are toposes.

Note also that the above graph is somewhat misleading because, strictly speaking, in a topos, **objects** are **types** and **morphisms** are **terms**. So "love" should be a map from the set of people to itself. In our graph this is broken down into individual relations.

Perhaps it is most important to understand that the morphism $A \to B$ in a topos is supposed to mean $A \subset B$ or $A \in B$ in set theory. For example $\text{John} \in \text{Men}$ and $(\text{John}, \text{Mary}) \in \text{Love}$.

The above knowledge graph can be easily converted into a collection of **logic formulas**:

```
Loves(John, Mary)
Loves(Pete, Mary)
Loves(Mary, Pete)
Hates(John, Pete)
Unhappy(John)
```

(7)

So, logic and graphs are basically **equivalent**.

Hypergraphs and simplicial complexes

If an edge of a graph can contain any number of vertices, then we have a **hyper-graph**. In other words, each edge of the hypergraph $\in \wp(V)$, V is the vertex set. It can also be said that the hypergraph is the **subset system** of V. For logic, the benefit is that there can be relations over relations.

Hypergraph can 1-1 correspond to topological **simplicial complexes**, and its homology and cohomology can be studied. Simplicial complexes can also 1-1 correspond to **square-free monomial ideals**. Square-free means that the exponent of x_i can only be 0 or 1. The latter is the scope of **combinatorial commutative algebra**. For the time being, I don't know if these associations are useful. See [Brown 2013], [Miller and Sturmfels 2005] for details.

A collection of logic formulas is called a logic **theory**. A collection of algebraic equations is called an **algebraic theory**.

For example, you can have the following logic formula ("unrequited love $\Rightarrow$ unhappy"):

$$\forall x, y.\ \text{Loves}(x, y) \wedge \neg\text{Loves}(y, x) \rightarrow \text{Unhappy}(x) \tag{8}$$

This formula contains the universal quantification $\forall$, so it is not part of the model. Logically, only a collection of **ground sentences** (formulas without variables) can form a model, such as (7) .

Fractal structure in the model space

A formula in a logic **theory** can cause many **new** vertices and edges to appear in the model. This is the study of model theory. In some cases, the model space will have an "infinitely subdivided" fractal structure.

For example, each natural number $n \in \mathbb{N}$ has its successor $S(n)$. The existence of this function results in a series of **infinite** vertices in the model space:

$$\bullet \quad \bullet \quad \bullet \quad \bullet \quad \bullet \quad \bullet \quad \bullet \quad \bullet \quad \bullet \ldots\ldots \tag{9}$$

If we add this **rule**:

$$\forall n \in \mathbb{N}. \quad S(n) \geq n \tag{10}$$

then it immediately generates an infinite number of relations:

$$\bullet \xleftarrow{\geq} \bullet \xleftarrow{\geq} \bullet \xleftarrow{\geq} \bullet \xleftarrow{\geq} \bullet \xleftarrow{\geq} \bullet \xleftarrow{\geq} \bullet \xleftarrow{\geq} \bullet \ldots\ldots \tag{11}$$

Although, in **common-sense intelligence**, this kind of infinite structures are rare; usually their structures are "shallow".

Partial models in an AI system

The knowledge representation of classical logic-based AI is split into two parts: **rules** and **facts**. The former are the formulas with $\forall$ **variables**, the latter are ground sentences. Rules are stored in KB and facts are stored in **working memory**. The former is a **theory**, the latter can be thought of as some **"partial" models**. The reason for saying partial is because it does not represent the entire model. In fact, the model is a huge structure that cannot be stored in any physical system. AI or the brain can only store certain theories and partial models. The key issue of AI is how to find a good syntax structure to make theory-learning faster and more efficient.

A concise formulation of strong AI

This section describes the mathematical structure of a strong AI system as concisely as possible, so that mathematicians without AI background can understand it.

Brief review of neural networks

In short, a neural network is a **parametrized** function with **universal approximation** ability.

A typical **neural network** is:

$$F(\vec{x}) = \sigma(W_1 \sigma(W_2 ... \sigma(W_L \vec{x}))) \tag{12}$$

(**weights** matrix; total # layers)

Its set of **parameters** is $\Theta = \{W^{\ell}_{i,j}\} \in \mathbb{R}^m$, where $m = \#$ weights.

The purpose of **machine learning** is to find the optimal Θ^* subject to an objective function. In other words, machine learning = **optimization**, which is one of the most basic problems in applied mathematics.

During training, it is given a set of data points (F is a neural network):

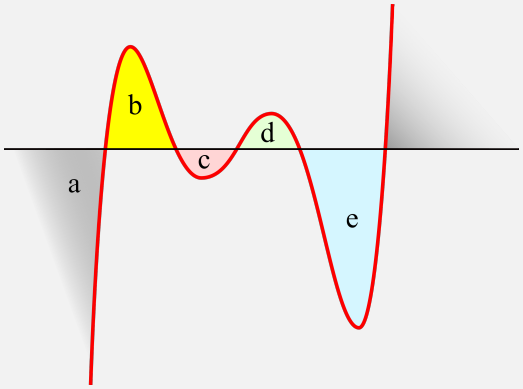
$$\boxed{\text{training examples}} \quad e \xrightarrow{F} a \quad \boxed{\text{answers}} \tag{13}$$

The error for each answer is ϵ, and the objective function J is the sum of the errors over many iterations. We want to optimize J by calculating the gradient of J w.r.t. Θ:

$$\nabla_\Theta J := \frac{\partial J}{\partial \Theta} \tag{14}$$

Of course, this is the famous **back-propagation** algorithm, which is actually gradient descent.

If the sigmoid function σ is replaced by a polynomial function σ , the overall network function would be a composite polynomial whose **degree** is the product of the degrees of every layer. In other words, the total degree grows **exponentially** as #(layers) grows. As the polynomial degree is the number of **zero-crossings**, it can be regarded as the number of possible **classifications** discernable by the network. For example, a quintic polynomial can distinguish up to 5 classes:



(15)

In other words, the classifying power of NN increases exponentially as #(layers).

Consider an NN whose inputs and outputs are **binarized** and of the same size, ie, it maps $\{0,1\}^n \to \{0,1\}^n$. The number of such distinct discrete functions is $(2^n)^{2^n} = 2^{n2^n}$, this number is **doubly exponential** in n, and is so large that in practice it could be regarded as ∞. Yet an NN is able to approximate this function space using $L \cdot n^2$ weights. The **hierarchical** organization of weights is what gives the NN its *"unreasonable effectiveness"*.

The bottleneck of strong AI is in the **learning algorithm** being too slow.

What is special about an AI learning algorithm is that it performs optimization over **logic formulas** instead of the usual $\mathbb{R}^n$:

Passage from optimization over $\mathbb{R}$ to optimization over $\mathscr{L}$

$$\Theta \in \mathbb{R}^n \quad \rightsquigarrow \quad \Theta \in \mathscr{L} \tag{16}$$

Where $\mathscr{L}$ is some kind of (*eg* first-order predicate) **logical syntax**, and Θ is a set of logic formulas. The solution to this optimization problem Θ^* is an optimal logic theory.

The top-level architecture of the system is reinforcement learning, a.k.a. dynamic programming. It is a special case of optimization. It can also be called control theory. The **system** it controls is:

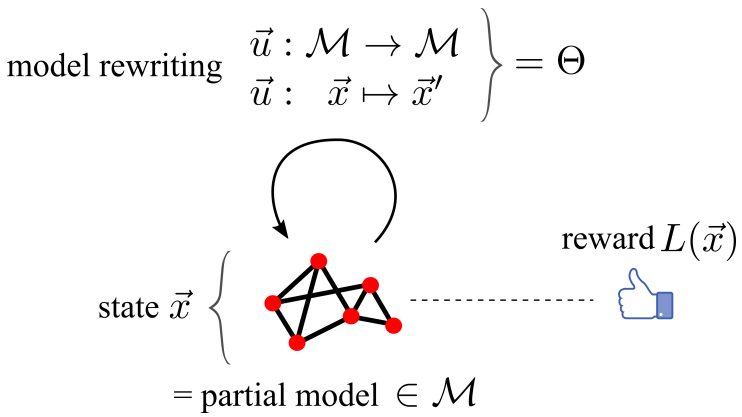
$$\vec{x}_{t+1} = \vec{f}(\vec{x}_t, \vec{u}_t) \tag{17}$$

$\vec{x}$ is the system's **state**, $\vec{u}$ is called the **control** or action. In AI, $\vec{x}$ is the **position** in "cognitive space", and $\vec{u}$ is the "thinking" steps. We want to control $\vec{u}$ so that the system reaches the maximum value of **rewards** over the long run:

$$\boxed{\text{total rewards}} \quad J = \sum_t L(\vec{x}) = \int L dt \tag{18}$$

where $L \in \mathbb{R}$ is the **instantaneous reward** at position $\vec{x}$. Due to historical reasons (from analytic mechanics), L is called the **Lagrangian**, its unit is energy (but the sign is reversed, now it measures the penalty), and note that $\vec{x}$ is in "cognitive space" which is not the same as physical space. I prefer to write the differential version as it is easier to remember.

The system operates as follows:

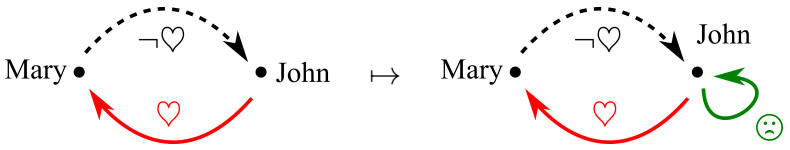


(19)

$\vec{u}$ and $\vec{f}$ coincide, its function is to **rewrite** $\vec{x}$:

$$\vec{f}(\vec{x}, \vec{u}) \equiv \vec{u}(\vec{x}) \tag{20}$$

For example, the logic rule "$\Rightarrow$" performs the rewriting of the following sub-graph:



(21)

This is the **state transition** $\vec{u} : \vec{x} \mapsto \vec{x}'$, which can also be regarded as the **logical inference** $\vec{u} : \vec{v} \vdash \vec{x}'$, where $\vec{u}$ is the rewriting function or logic rule.

Optimization of J acts over Θ (ie $\vec{u}$). Note that $\vec{u}$ belongs in a function space or the space of logic formulas, which is very different from the traditional optimization over $\mathbb{R}$.

In classic AI, $\vec{u}(\vec{x}) = \vec{f}(\vec{x})$ is equivalent to the deduction $\vec{x} \vdash \vec{x}'$, also known as **forward-chaining** (deduce logical conclusions). In classic AI this job is handled by an **inference engine**, which consists of 2 algorithms: **unification** (= rule matching) and **resolution** (= proof search). These operations are not visible in our framework as they are absorbed into $\vec{u}$ or $\vec{f}$.

Hamilton-Jacobi-Bellman equation

For experts in applied mathematics, the following theory is quite standard. It merely sets up the AI system under the framework of reinforcement learning, which otherwise has no substantial content.

The definition of Hamiltonian in analytical mechanics is:

$$H = L + \frac{\partial J}{\partial \vec{x}} \vec{f} \tag{22}$$

Similarly, the Hamiltonian of the **discrete** system can be defined as:

$$H = L + J(\vec{f}(\vec{x})) \tag{23}$$

Pontryagin[†] 's **maximum principle** gives the condition for the optimal solution:

$$H^* = \inf_u H \quad \text{or} \quad \nabla_{\vec{u}} H^* := \frac{\partial H^*}{\partial \vec{u}} = 0 \tag{24}$$

We can define an operator T that acts on J:

$$TJ := \inf_u H \tag{25}$$

Then Bellman's optimality condition can be expressed as the following fixed-point equation [Bertsekas 2013]:

$$J^* = TJ^* \tag{26}$$

It is said that the Hamilton-Jacobi method leads to the solving of partial differential equations, which is not particularly efficient. In practice, the Pontryagin minimum principle is more useful.

There is a gradient $\nabla_{\vec{u}}$ in (24), so it would be useful to find a derivative for $\vec{u}$. If the problem is continuous optimization, one can use non-smooth analysis. The minimum requirement is that there is a norm (*ie*, Hilbert space or Banach space) in the domain, then we can employ techniques such as proximal gradient.

One question is: why hasn't the keyword "**symplectic**" appeared in current AI literature on reinforcement learning, while all existing techniques are **statistical** in nature? I suspect that it may be due to either: 1) the time-step being discrete; or 2) the dimensionality of the state space being too high (though this dimensionality may not be *intrinsic* to the problem, *ie*, it may be possible to embed into lower dimensions).

[†]Lev Pontryagin (1908-1988) Soviet mathematician. Blind since the age of 14, he made major discoveries in a number of fields including algebraic topology and differential topology.

Plan 0: geometric models

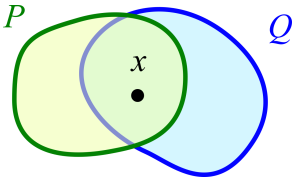
first-order syntax $\mathscr{L}$ is unnecessary;
we only need the ability to **rewrite** models (27)

There can be many variations on this theme, for example:

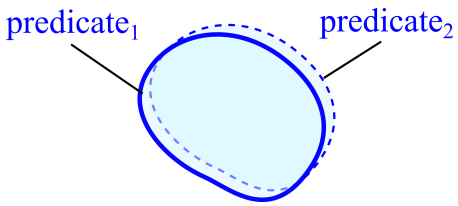
$$\begin{array}{ccc} \text{logical syntax } \mathscr{L} & & \text{graph re-writer} \\ \downarrow & \approx & \downarrow \\ \text{partial model } \mathscr{M} & & \text{graph model} \end{array} \tag{28}$$

Consider a kind of what I call **geometric models**, as typically exemplified by Venn diagrams:

$$P(x) \wedge Q(x) \quad \cong \quad \tag{29}$$



The motivation for introducing geometric models: Since we need to perform search among logic formulas, it may be desirable to represent formulas such that they can continuously "morph" into each other:



(30)

In classical logic this is not possible. For example, Love(x, y) and Like(x, y) may be similar in meaning, but the transition from one to the other requires a discrete "jump".

We propose to use **points** to represent first-order objects (*eg* "John") and **regions** (*ie*, point sets, open sets) to represent predicates and relations (*eg* "Love"). It is possible to use a *dual* model in which points and regions are swapped.

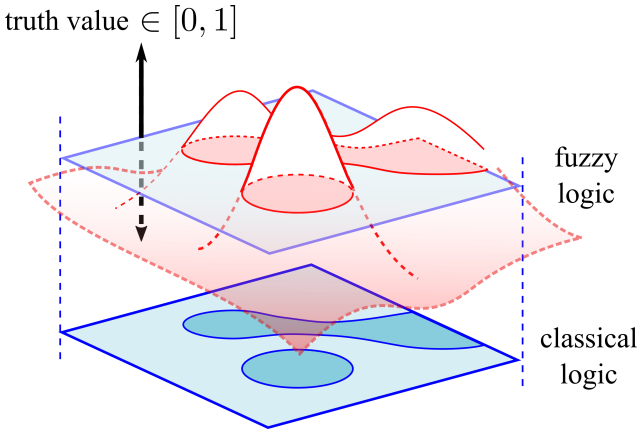
A caveat is that the points and regions must be *labelled* entities. For example, it is not sufficient to know that the point (John, Mary) belongs to a relation, but it matters whether that relation is Love or Hate. In other words, the *identity* of the region matters.

Previously I made the mistake of forgetting the labels associated with geometric regions, thus falsely simplifying the algorithm and claimed success. It turns out that the presence of labels forces us to handle the rewriting function on the **syntactic** level, and the geometric shapes of models do not seem to make an essential difference (to my disappointment). So this idea needs re-consideration.

This representation scheme can be summarized as follows:

$$\begin{aligned} \text{model } \mathscr{M} &= \text{geometric model of set theory} \\ \text{rewriter} &= \text{functor } \mathscr{M} \to \mathscr{M} \end{aligned} \tag{31}$$

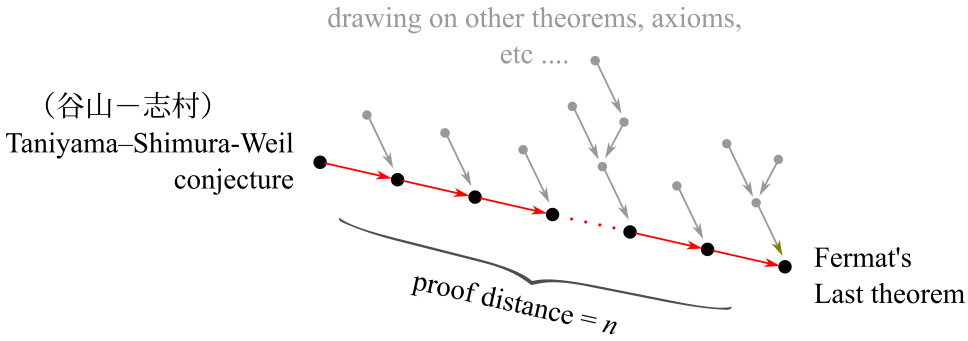
But we need a model of set theory that can be continuously changed, maybe it can be achieved with some kind of fuzzy topology? E.g:



(32)

The boundary of regions that need to be defined is 1 degree higher.

But here's the problem: Logically, the semantic distance between two expressions can be defined as the number of proof steps required to derive from one formula to another (it also depends on the contents of KB), but according to the halting theorem that Turing proved in 1936, this distance is **incomputable**. ‡ In other words, it is impossible in the space of logic formulas to have an exactly defined semantic metric. Our $\vec{u} = \Theta$ is analogous to logic formulas; If we can define a gradient ∇_Θ on Θ, it seems to lead to contradiction. Can **fuzzy topology** be able to approximate the semantic metric? This depends on whether the fuzzy model-rewriting function space Θ is **metrizable** or not (*cf* [Goubault-Larrecq 2013]). Further investigation is needed.



(33)

The "plateau problem": From [Bergadano and Gunetti 1996]: Researchers have discovered that, during the inductive learning of logic formulas, it may happen that in the body of the correct clause, the gain remains low and constant (on a plateau) until one final important literal concludes the computation and brings the gain to a local maximum. An example is the following target clause:

```
append(X,Y,Z) :- list(X), head(X,X1), tail(X,X2),
append(X2,Y,W), cons(X1,W,Z).
```

(34)

The following main analysis of plan A and B (they are already feasible). Plan A uses discrete optimization directly, so no metric space is required. Plan B uses a neural network whose weights $\in \mathbb{R}$ is a contiguous space, but this neural network acts on the graph, which is still a discrete structure.

‡The logic semantic distance is similar to **Kolmogorov complexity** but not exactly the same. Kolmogorov complexity is incomputable but can be approximated [Li and Vitanyi 2008].

Plan A: genetic algorithm

abandon gradient descent (35)

$$\begin{aligned} \text{model } \mathscr{M} &= \text{symbolic logic formula} \\ \text{rewriter} &= \text{symbolic logic formula + classical AI logic engine} \end{aligned} \tag{36}$$

GA is very suitable for the **discrete** search space. It is compatible with the logical structure. There is no theoretical obstructions on this route.

First you need a logic-based **rule engine**, which is responsible for forward-chaining, which is completely the scope of classic AI. For example, the classic Soar architecture [Carnegie-Mellon University] is a rule-base engine.

The genetic algorithm's population is composed of individual logical rules, but the winner is not a single rule, but a set of rules (the highest score of N). This is called **cooperative co-evolution**(COCO).

Inputs and outputs are logic formulas, which are actually easier to handle.

The whole system is still based on reinforcement learning, but you don't need to do RL directly, because those rules are actually **actions**, and the probabilistic strength of each rule is like the Q value in Q-learning.

[I have not had time to explore the practical theory of COCO.]

Plan B: neural network / deep learning

Most of the time is to solve the problem of how to implement the classic logic engine with NN, especially the problem of variable substitution. Finally, the crux of the problem is the lack of short-term memory mechanism.

The solution is to use graph as **memory system** and use neural network for graph re-writing, which is the graph neural network proposed by Google / DeepMind. This is a "hybrid" architecture.

NNs have difficulty handling substitutions

Consider the logic rule that was mentioned in the previous section ("Love is not happy"):

$$\forall x, y.\ x \heartsuit y \wedge \neg y \heartsuit x \to \text{☹} x \tag{37}$$

The **predecessor** (antecedent) of this rule must be established, and must be <u>*a* **equal**, twice appearing, *b* **equal**</u>:

$$a \heartsuit b \wedge \neg b \heartsuit a \tag{38}$$

Also, to produce the correct **postsent** (consequent), you need to pass *a* **copy** from the previous:

$$a \heartsuit b \wedge \neg b \heartsuit a \to \text{☹} a \tag{39}$$

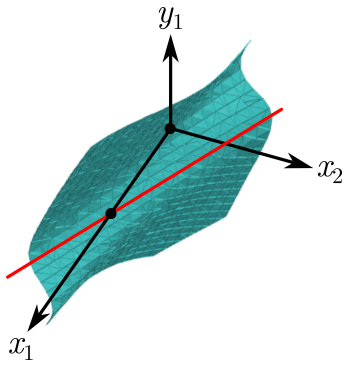
These two actions (**compare** and **copy**) are hard to do with neural networks. But they are the essence of variable substitution and the trouble of predicate logic. In other words, it is difficult to accomplish these two actions in one breath with a monolithic end-to-end neural network:

$$a\heartsuit b \wedge \neg b\heartsuit a \longrightarrow \text{☹}a \tag{40}$$

First consider the copy problem of **poster**. For simplicity, assume that the logical variable z corresponds to some component of the input vector $\vec{x}$, such as x_i. The purpose of Copy is to copy x_i to the y_j position:

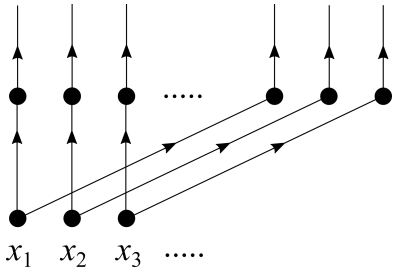
$$\vec{F}: \ (x_1, ..., x_i, ..., x_n) \mapsto (y_1, ..., y_j, ..., y_n) \tag{41}$$

This requires the function surface of the neural network to pass through some diagonal lines, as shown below:



(42)

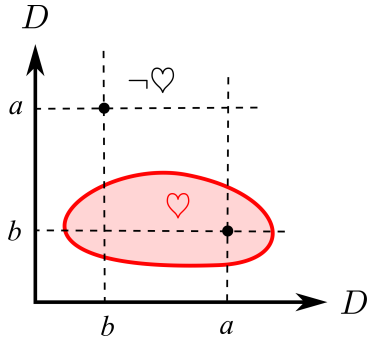
The following is a simple copier neural network (ownership weight = 1, other weights = 0 not shown):



(43)

A neural network with input dim = n and output dim = $2n$, if fully connected, needs to train $2n^2$ weights. But I haven't had time to test how long it takes to train a multi-layer neural network to learn this action.

Second, consider the establishment of **frontware**, a viable **geometric image** is such a [§]:



$$\Rightarrow \text{☹}a \tag{44}$$

[§]this is just one of many possible representations, but it seems that any "geometric" form of representations has similar problems. . Unless we consider representations with the characteristics of "procedural"? The following will discuss ....

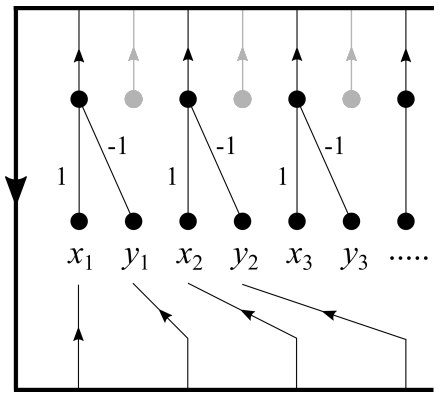
$\in$ is easy to solve with a neural network, for example $\heartsuit$ can be defined as a neural network:

$$\heartsuit(\vec{x}) = \begin{cases} 0 & \vec{x} \notin \text{region} \\ 1 & \vec{x} \in \text{region} \end{cases} \tag{45}$$

But even then, there is still a pattern matching (comparison) problem:

$$\begin{aligned} \vec{p_1} = (\vec{a}, \vec{b}) \quad &\in \quad \heartsuit \\ \not\Downarrow \\ \vec{p_2} = (\vec{b}, \vec{a}) \quad &\in \quad \neg\heartsuit \end{aligned} \tag{46}$$

The following is a simple comparator simulated with the RNN neural network (all = 0 weights are not shown):



(47)

After iterate n times, the leftmost output will be the true or false value of $\vec{x} \stackrel{?}{=} \vec{y}$. Assuming the input dimension is $2n$, you need to train $(2n)^2$ weights.

Conclusion: Based on the above analysis, it seems not very difficult to simulate copier and comparator with NN, but in fact, these "components" are used in conjunction with short-term memory, and the entire architecture is still unknown.

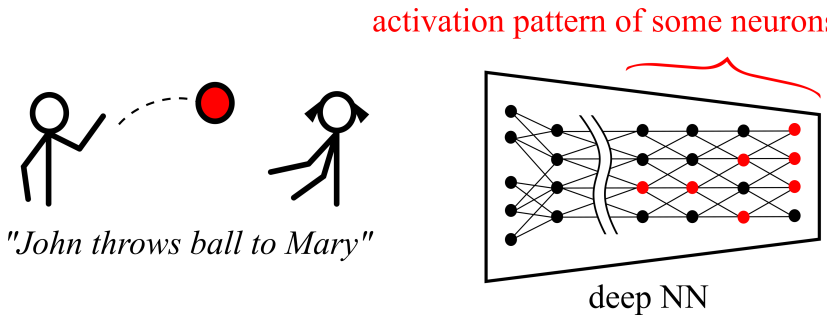
"Distributive" representations

The meaning of **Distributive representation** is: Suppose there is a vector indicating that the output of the neural network has $n = 10$ granule neurons:

$$\vec{x} = (x_1, x_2, ..., x_{10}) \tag{48}$$

In binary, each $x_i \in \{0, 1\}$, $\vec{x}$ can represent the concept of 10 "**one-hot**" respectively. But if you use distributive representation, these 10 bits can express up to $2^n = 1024$ different states/concepts. But in fact, the conjunctions of one-hot features are not different from distributive representations if they are treated as different states. So, the representation of a neural network can be said to be $\mathbb{R}^n$ vector , or as n coordinates of the n-dimensional manifold.

For example, *"John throws ball to Mary"* This image, after processing such as CNN, can get a **distributed knowledge representation**:



(49)

This can be understood as: a "neat" proposition is broken down into a number of small propositions, for example:

$$\text{A B} \Longleftrightarrow \begin{cases} \text{A flings his arm } \wedge \\ \text{ball leaves A's hand } \wedge \\ \text{ball flies in mid-air } \wedge \\ \ldots \end{cases} \tag{50}$$

These two sides are **logically equivalent**. The details of "The ball is red" cannot appear on the right side, because this detail is not left **implication**. But there can be "the ball is usually round". in other words:

$$\boxed{\text{"neat" proposition}} \quad p \Longleftrightarrow \begin{cases} q_1 \wedge \\ q_2 \wedge \\ \ldots \wedge \\ q_n \end{cases} \quad \boxed{\text{distributed propositions}} \tag{51}$$

Conclusion: Unknown What is the impact of distributed knowledge representation on AI? For the time being, our architectures are equally applicable to distributive and neat logic, except for the increase in the number of propositions and the part of the "visual nerve".

NNs lack short-term memory mechanism

Consider the image of "White Cat Chasing Black Cat":

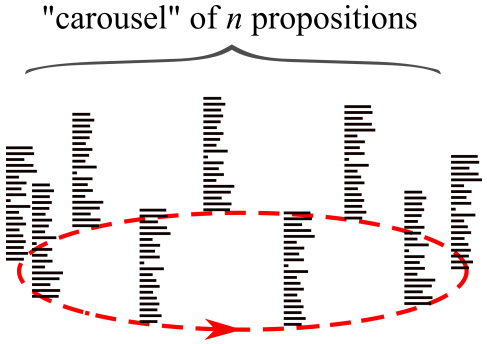
(52)

The concept of "cat" needs to appear **twice**, but the feature corresponding to "cat" in the neural network is only **set** (unless there are two duplicate modules that can represent any concept, but it is wasteful) . In other words, the current CNN does not have the ability to "traverse" the field of view; it cannot distinguish and describe the **relationships** between objects.

It's hard to imagine how a "monolithic" neural module (such as feed-forward NN or RNN) can do this. It seems necessary to express the proposition as a series of concepts of **time sequence**, some kind of **short-term memory** (**STM**, short-term memory).

I was a little surprised to find that the current neural network does not have the mechanism of **short-term memory**, and "short-term" means that the time-scale is shorter than the time when the weights change. For example, I tell you a string of numbers (such as a phone number), you can remember it in the brain, but this mechanism seems to have no research in the current artificial neural network, perhaps some models in computational neuroscience, but for the time being I don't know. Without such an STM, it is difficult to simulate symbolic logic. In other words, strong artificial intelligence cannot be achieved.

For example, using NN to implement a **dynamic memory**, when it receives a new element, it will **compare** to other elements in the memory, and it has the **copy** function. For example, the following time series mechanism like "Rotating Trojan" (each ▮▮▮ represents a distributive vector):



(53)

In short, it's obviously cumbersome to simulate STM purely with NN.

Graph NNs

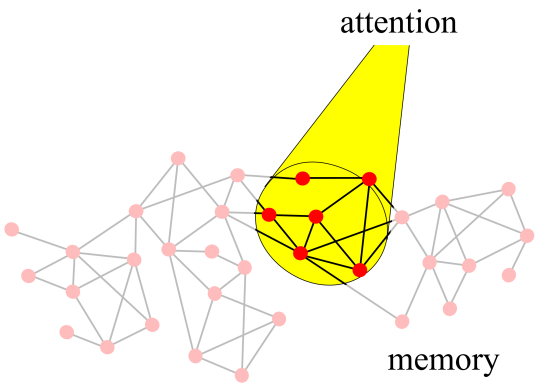
Use a **graph** for memory storage
(including long-term and short-term memories) (54)

When adding a new memory unit, the same nodes will be matched into one. In other words, the matching step is solved by the traditional symbolic method, and the problem is left to the neural network.

$$\begin{aligned} \text{model } \mathscr{M} &= \text{graph} \\ \text{rewriter} &= \text{deep NN} = \text{graph neural network} \end{aligned} \tag{55}$$

Many thanks to the graph network paper [Battaglia et al. 2018] published by Google / DeepMind in June 2018, Peter Battaglia and 26 collaborators surveyed the development of graph network. The graph network they proposed is closer to some physical systems such as springs and spheres, rather than the first-order model, but essentially the same.

Usually the model is too large, you need to use the attention mechanism to select a fragment of it, and then "present" to handle the neural network:



(56)

This attention mechanism is somewhat different from the attention in the current deep learning or in the specific details, but basically the same concept.

The neural network input $\vec{x}$ requires an embedding like this:

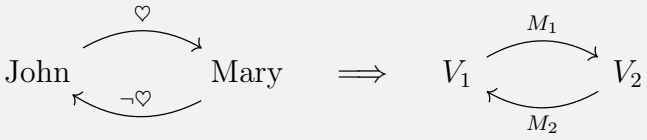
$$\boxed{\text{graph}} \quad \xrightarrow{\text{embed}} \quad (x_1, ..., x_m) \quad \boxed{\text{vector}} \tag{57}$$

But graph is not a "linear" structure [¶], it seems quite difficult to represent the graph structure as a vector (this may be the delay of the graph neural networks) There are reasons for the breakthrough).

[¶] linear means symbolic form, for example, tree can be represented as a linear line

Quiver representations

In representation theory one studies **quiver representations**, which turns **vertices** into vector spaces and **edges** into linear transformations between vector spaces. For example:



(58)

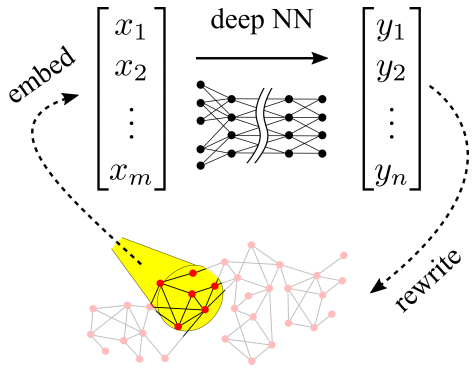
where V_1, V_2 are vector spaces, $M_1, M_2 \in GL(\mathbb{R})$ are matrices.

The vector spaces V_i can be stacked up vertically to form a single big vector space V. Then all the relations fit into an endomorphism $V \to V$.

Under **change of basis** of the vector spaces V_i, two matrices M and M' may turn out to be the same linear transformation. Therefore, we need to consider their **invariant form** or **moduli**. Representation theory is concerned with the decomposition of M into irreducible components. The **Dynkin diagrams** that arise in this decomposition are the same as those found in Lie algebra classification. If the quiver is not a Dynkin diagram then the quiver is "wild" and is generally difficult to study. Even a simple quiver can be of wild type.

Each quiver defines a **path algebra** whose elements are the paths in the quiver. In logic these paths correspond to logical **relations** and their composition.

The overall operation is like this:



(59)

This deep NN can be either CNN or RNN because they are currently very successful in natural language understanding/translation, but they deal with linear sequence inputs. Or you can split the memory sub-graph into linear elements (that is, individual relationships/propositions), and you can't use global variable references (in other words, variable matching has been done). Then the variable copying output by NN is also externally processed. In other words, in combination with NN and **graph rewriting** in the "hybrid" way. I still haven't seen a good solution for the embedding details in the picture, but DeepMind / Google's research is very close.

Note: Although the parameter space $\Theta \in \mathbb{R}^N$ of NN is **continuous**, the rewriting rules it learns are **discrete** because graph itself is a discrete structure. .

Add a point: attention mechanism to traverse memory graph, in other words a graph search algorithm, this part can be combined with NN into the same module (there are many RNN architectures now).

The rest of this paper are some background mathematical knowledge...

What is topos theory?

Topos theory can be understood as a generalization of set theory under the influence of category theory.

An **elementary topos** ("elementary" as in "elements" of a set) is a **Cartesian-closed category** (CCC) with a **sub-object classifier** Ω.

A Cartesian-closed category is, roughly speaking, where one can form arbitrary:

- **Cartesian products** $A \times B$, and
- **exponentiation** B^A

where $B^A \simeq A \to B$ is the class of all functions from A to B.

The sub-object classifier Ω allows a topos to handle **subsets** as in set theory. In the category **Set**, Ω is $\{\top, \bot\}$, representing True and False.

For example, "Love" is a **relation** within $D \times D$, where D is the universe set of all "people". One can think of $D \times D$ as the "full" relation, then Love $\subset D \times D$ is its **subset**. Thus sub-objects enable elementary toposes to model relation algebra and first-order logic. A standard textbook on topos theory is [Goldblatt 1984, 2006].

Some history

"The development of elementary topoi by Lawvere and Tierney [around 1960-70] strikes this writer as the most important event in the history of categorical algebra since its creation..... It is not just that they proved these things, but that they dared to believe them provable." — Peter Freyd.

"In a sense logic is a special case of geometry." — Bill Lawvere.

Around 1963, the concept of topos came independently from several different sources:

- Alexander Grothendieck's sheaf theory in algebraic geometry
- F William Lawvere's categorical re-formulation of set theory
- Paul Cohen's invention of forcing (to resolve the **Continuum Hypothesis**)

A useful aspect of topos theory is that it allows for a great variety of different **models** in contrast to ZFC for which it is fairly hard to construct different models. Thus topos theory helped Cohen to construct forcing models for ZFC where the Continuum Hypothesis fails.

Sheaf means: Objects defined on some open sets V_i, they are consistent on the overlap $V_i \cap V_j$, which can be "collate". The situation is like some charts in differential geometry. . After World War II, Leray, then Cartan, defined sheaf using the open sets method. Later Lazard defined sheaf with étale, which is the main motivation for the topos theory. For example, a pre-sheaf on a category $\mathscr{C}$ can be defined as a **functor**:

$$\widehat{\mathscr{C}} : \mathscr{C}^{\text{op}} \to \mathbf{Set} \tag{60}$$

That is, a "layer" set-valued function on the category $\mathscr{C}$. It is "functor" so it handles those collating. This functor is contravarient so there is op. Grothendieck Applying sheaf to topology (cohomology), then Jean-Pierre Serre found that it can also be used in algebraic geometry. They and other collaborators wrote the 1623-page masterpiece "SGA IV", which greatly influenced the development of algebraic geometry, resulting in In 1974 Deligne solved the Weyl conjecture. But I am not familiar with algebraic geometry for the time being, so it is not clear what Grothendieck did.... See the book [MacLane and Moerdijk 1992] for details.

In the topology space, the complement of an open set U is closed and not necessarily open, so if it is confined within the open sets, the "negation" of U should be defined as "the interior of its complement' '. This causes U's "**double negative**" not necessarily equal to U, in other words, the algebra of open sets follows **intuitionistic logic**, such an algebra is called a **Heyting algebra**. (cf. ibid.)

There are two kinds of morphisms between Topoi: **geometric morphisms** and **logical functors**. The former maintains "geometry", the latter maintains a logical type theory, so there is a definition of **elementary topos**. The latter is characterized by its **sub-object classifier** Ω.

Sub-object classifier

This commutative diagram defines the sub-object classifier Ω:

$$\begin{array}{ccc} A & \xrightarrow{\ f\ } & D \\ {\scriptstyle !}\downarrow & \lrcorner & \downarrow{\scriptstyle \chi_f} \\ 1 & \xrightarrow{\ true\ } & \Omega \end{array} \tag{61}$$

Pulling $true$ back along χ_f yields the set $\{x : \chi_A(x) = 1\}$ which is just A.

The arrow χ is called the **classifying** arrow of the sub-object A; It can be thought of as taking exactly the part of D that is A to the "point" t of Ω.

We form a collection, the sub-objects of D:

$$\mathrm{Sub}_{\mathscr{C}}(D) = \{[f] : f \text{ is a monic with } \mathrm{cod} f = D\} \tag{62}$$

If $\mathscr{C}$ has pullbacks (of monomorphisms along arbitrary morphisms in $\mathscr{C}$) then $\mathrm{Sub}_{\mathscr{C}}(D)$ extends to a functor $\mathrm{Sub}_{\mathscr{C}} : \mathscr{C}^{\mathrm{op}} \to \mathbf{Set}$ by putting $\mathrm{Sub}_{\mathscr{C}}(f)([m]_{\sim}) = [f^*m]_{\sim}$ where f^*m is the pullback of m along f. (??) [Streicher 2006] p.79.

The pullback condition is equivalent to requiring the contra-variant sub-object functor,

$$\mathrm{Sub}_{\mathscr{C}}(-) : \mathscr{C}^{\mathrm{op}} \to \mathbf{Sets} \tag{63}$$

(which acts by pullback) to be **representable**, ie:

$$\mathrm{Sub}_{\mathscr{C}}(-) \cong \mathrm{Hom}_{\mathscr{C}}(-, \Omega). \tag{64}$$

The required isomorphism is just the pullback condition stated in the definition of a sub-object classifier [Awodey 2006] p.175.

A contra-variant (or co-variant) presheaf $F : \mathscr{C}^{\mathrm{op}} \to \mathbf{Set}$ is called **representable** if it is isomorphic to $\mathrm{Yon}_{\mathscr{C}}(I) = \mathscr{C}(-, I)$ (or $\mathscr{C}(I, -)$) for some $I \in Obj\ \mathscr{C}$.

Example. If $\mathscr{C}$ is a monoid M (considered as a category) then Ω consists of all right ideals in M, *ie*, subsets I of M such that $x \in I$ and $y \in M$ implies $xy \in I$, and $\Omega(x)(I) = \{y \in M | xy \in I\}$. [Streicher 2006] p.83. (??)

Example. If $\mathscr{C}$ is a group then according to the above example, the truth value object Ω of the topos $\widehat{\mathscr{C}}$ of G-actions has $\{G, \varnothing\}$ as underlying set because G and $\varnothing$ are the only right ideals in G which, moreover, are left invariant by all actions of group elements.

Fibration

[Jacobs 1999]

Predicates on sets can be organized in a category called **Pred**:

- **Objects** are pairs (I, X) where $X \subseteq I$ is a subset of a set I; in this situation we consider X as a predicate on a type I, and write $X(i)$ for $i \in X$ to emphasize that an element $i \in I$ may be understood as a free variable in X. When I is clear from the context, we sometimes write X for the object $(X \subseteq I)$.
- **Morphisms** $(I, X) \to (J, Y)$ are functions $u : I \to J$ between the underlying sets satisfying:

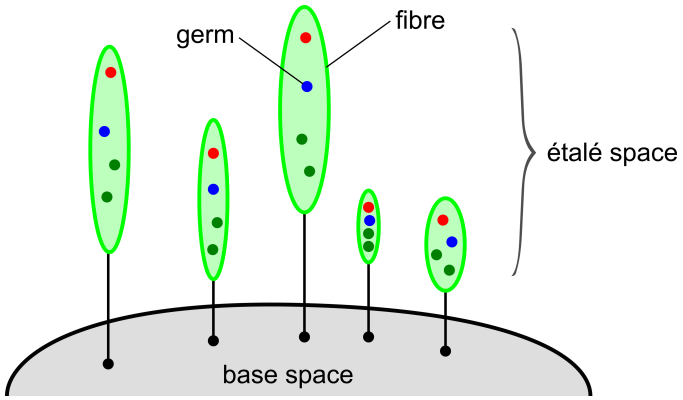
$$X(i) \text{ implies } Y(u(i)), \text{ for each } i \in I. \tag{65}$$

  Diagrammatically, this condition on such a function $u : I \to J$ amounts to the existence of a necessarily unique (dashed) map:

$$\begin{array}{ccc} X & \longrightarrow & Y \\ \downarrow & & \downarrow \\ I & \xrightarrow{u} & J \end{array} \tag{66}$$

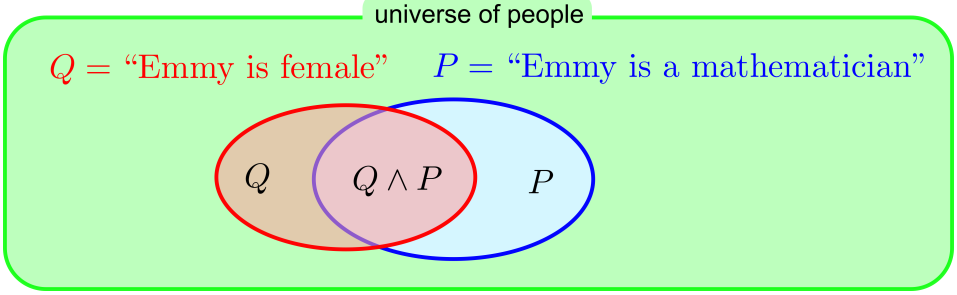
  indicating that u restricts appropriately.

Illustration of a **fibration**:



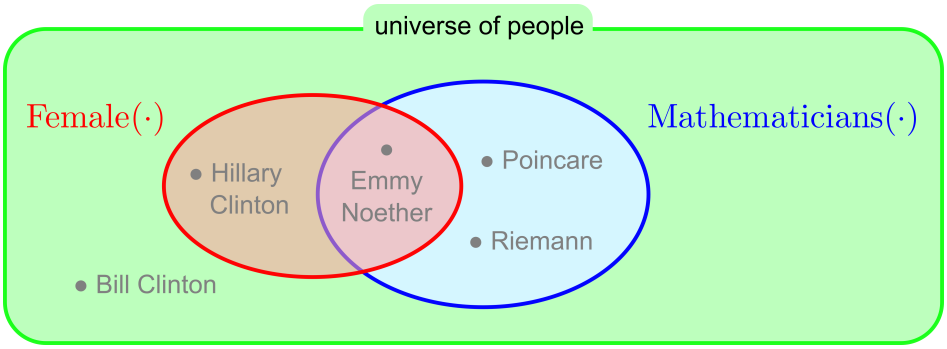
(67)

Propositional logic is a kind of **topological** (open set) structure:



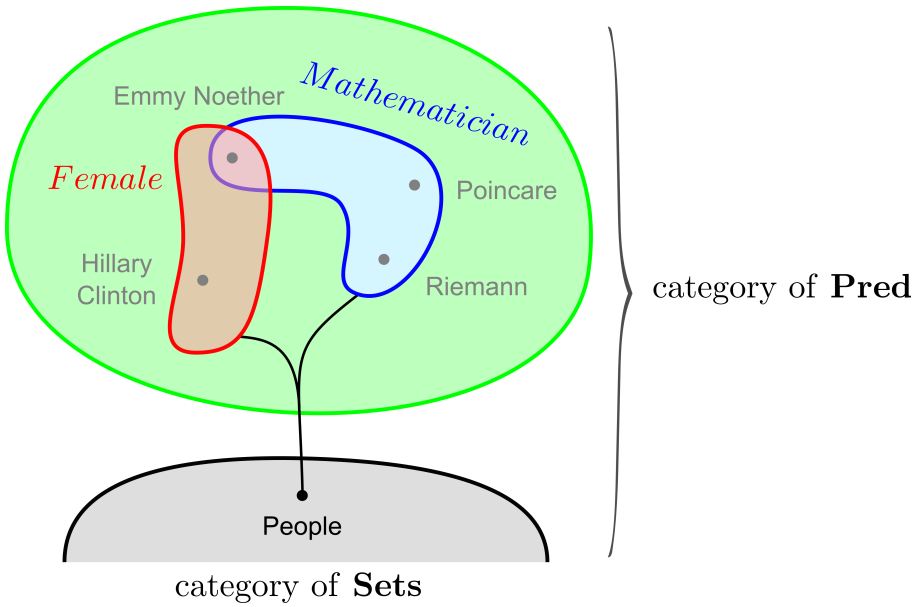
(68)

Inside a proposition, there is the **predicate** structure



(69)

Predicate logic could be regarded as a **fibration** over propositional structure, denoted as $\begin{smallmatrix}\textbf{Pred}\\ \downarrow \\ \textbf{Set}\end{smallmatrix}$:



(70)

Quantifications are adjoints

A pair of monotone functions $f : X \to Y$ and $g : Y \to X$ between pre-ordered sets determines an **adjunction**, if $\forall x \in X, y \in Y$:

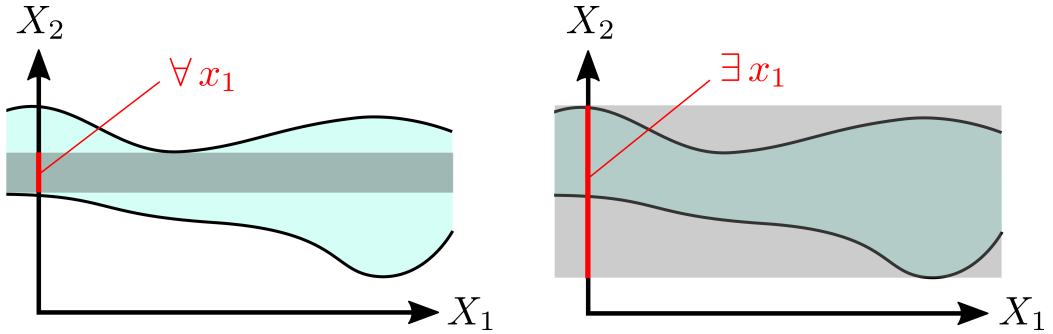
$$f(x) \leq y \text{ in } Y \quad \Leftrightarrow \quad x \leq g(y) \text{ in } X \tag{71}$$

in which case we write $f \dashv g$ and say "f is left-adjoint to g", or "g is right-adjoint to f". (Mnemonic: the *left* adjoint appears on the *left* of $\leq$ and *vice versa*)

Lawvere discovered the following adjunctions:

$$\exists \dashv * \dashv \forall \tag{72}$$

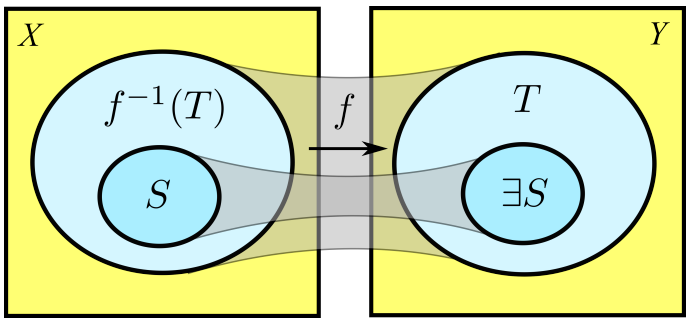
In cylindric algebra, the quantifiers $\forall$ and $\exists$ can be interpreted as **projections** to a component (X_2) of the domain



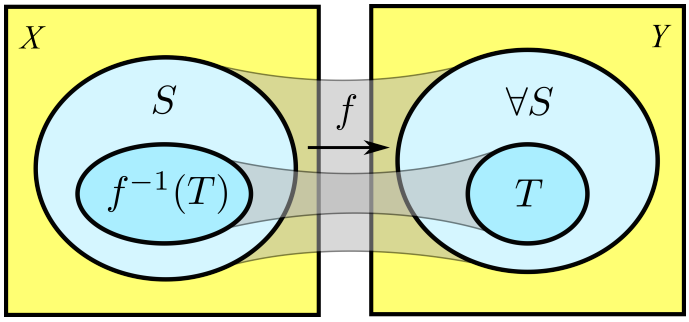
(73)

The following 2 formulas are from [Abramsky and Tzevelekos 2011]. For all $S \subseteq X, T \subseteq Y$:

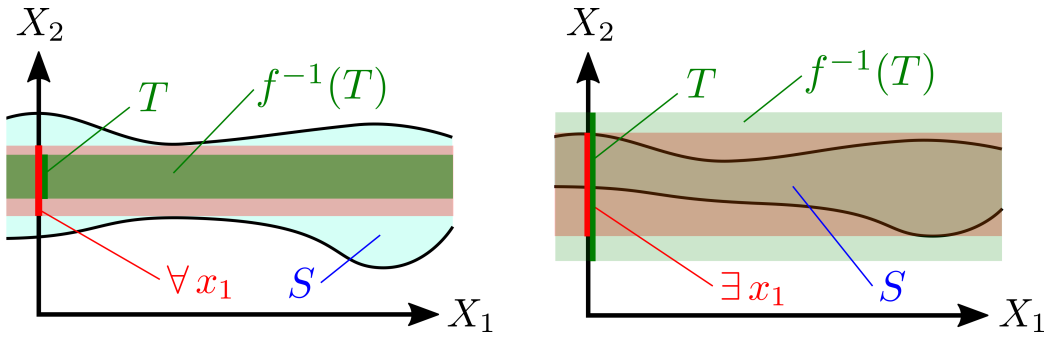
$$S \subseteq f^{-1}T \iff \exists_f S \subseteq T \tag{74}$$



$$f^{-1}T \subseteq S \iff T \subseteq \forall_f S \tag{75}$$



They can be interpreted in a "cylindrical" way: Let $X = X_1 \times X_2$ and $Y = X_2$. Then f is a projection $X_1 \times X_2 \to X_2$:



(76)

Logic in a topos

In (higher-order) predicate logic we are concerned with sequents of the form:

$$\Gamma \rhd \Phi \vdash \phi \tag{77}$$

where Φ is the **assumption**, ϕ the **conclusion**, and Γ is a **variable context** and specifices which free variables are allowed in Φ and ϕ. For example:

$$\overbrace{x : \mathbb{N}, y : \mathbb{N}}^{\text{variable context}} \rhd \ \mathrm{odd}(x) \wedge \mathrm{odd}(y) \vdash \mathrm{even}(x + y) \tag{78}$$

For every **type** α in the signiture we have an object $[\![\alpha]\!]$ of $\mathscr{C}$.

For every **proved term** $\Gamma \vdash M : \alpha$, we have a morphism in $\mathscr{C}$:

$$[\![\Gamma \vdash M : \alpha]\!] : [\![\Gamma]\!] \to [\![\alpha]\!] \tag{79}$$

For $A \in \mathrm{Obj}\, \mathscr{C}$, the set of **predicates** in $A = \mathrm{Sub}_{\mathscr{C}}(A) \cong \mathscr{C}(A, \Omega)$, with a partial order $\leq_A$ on it.

Categorical semantics

Categorical semantics is model theory expressed in category theory.

We have the following duality:

- Every algebraic / logic theory Th has a **classifying category** $Cl(Th)$
- Every category $\mathscr{C}$ with finite products has an **internal language** $Th(\mathscr{C})$

with the equivalences:

$$\begin{aligned} Cl(Th(\mathscr{C})) &\simeq \mathscr{C} \\ \text{and} \quad Th &\simeq Th(Cl(Th)) \end{aligned} \tag{80}$$

According to nLab: The **MitchellBénabou language** is a particularly simple form of the internal language of an elementary topos $\mathscr{E}$. In [Streicher 2006] it is show that an arbitrary elementary topos $\mathscr{E}$ can interpret **higher-order intuitionistic (constructive) logic**.

The category $Cl(Th)$ has:

- objects = finite lists of types, *eg*, $\vec{\alpha} = [\alpha_1, ..., \alpha_n]$
- morphisms from source $\vec{\alpha}$ to target $\vec{\beta} = [(\Gamma|M_1), ..., (\Gamma|M_m)] : \vec{\alpha} \to \vec{\beta}$

where $(\Gamma|M)$ is an equivalence class of pairs (Γ, M) of contexts and raw terms:

We define a **category of models Mod**$(Th, \mathscr{C})$:

- objects = models of Th in $\mathscr{C}$
- morphisms = homomorphisms of models of Th in $\mathscr{C}$

and we define another category **FP** (for finite product) that has:

- objects = finite-product preserving functors
- morphisms = natural transformations

then we can define a **modelling functor** Ap_G:

$$Ap_G : \ \mathbf{FP}(Cl(Th), \mathscr{D}) \to \mathbf{Mod}(Th, \mathscr{D}) \tag{81}$$

[Caramello 2018] [Goldblatt 1984, 2006]. *Conceptual mathematics* [Lawvere and Schanuel 1997, 2009] [Lawvere and Rosebrugh 2003].

Different logics can be defined by **proof theory** (which studies *syntactic* rules of deduction):

algebraic logic	no additional rules
Horn logic	finite $\wedge$
regular logic	finite $\wedge$, $\exists$, Frobenius axiom
coherent logic	finite $\wedge$ and $\vee$, $\exists$, distributive axiom, Frobenius axiom
geometric logic	finite $\wedge$, infinitary $\vee$, $\exists$, infinitary distribution axiom, Frobenius axiom
first-order intuitionistic logic	all finitary rules except law of excluded middle
first-order classical logic	all finitary rules

(82)

For example, **algebraic theory** means: It has only one relation $=$, and all axioms are of the form $s = t$.

There are also examples of these deduction rules:

$$\boxed{\wedge \text{ rule}} \quad \frac{\Phi \vdash \Psi,\ \Phi \vdash \chi}{\Phi \vdash (\Psi \wedge \chi)} \tag{83}$$

$$\boxed{\exists \text{ double rule}} \quad \frac{\Phi \vdash_{\vec{x},y} \Psi}{\exists y\, \Phi \vdash_{\vec{x}} \Psi} \tag{84}$$

$$\boxed{\text{Frobenius axiom}} \quad \Phi \wedge \exists y\, \Psi \vdash_{\vec{x}} \exists y\, (\Phi \wedge \Psi) \tag{85}$$

Tarski's model theory "corresponds" the first-order syntax to the **structure** of the set theory. As a result, different logical syntaxes correspond to different structural categories:

categories with finite products	algebraic logic
Cartesian categories	Cartesian logic
regular categories	regular logic
coherent categories	coherent logic
geometric categories	geometric logic
Heyting categories	first-order intuitionistic logic
Boolean coherent categories	first-order classical logic

(86)

The term "Geometric" logic comes from **geometric morphisms**, which is a kind of morphism between 2 toposes that are particularly "natural" to define, similar to **continuous maps** between topological spaces.

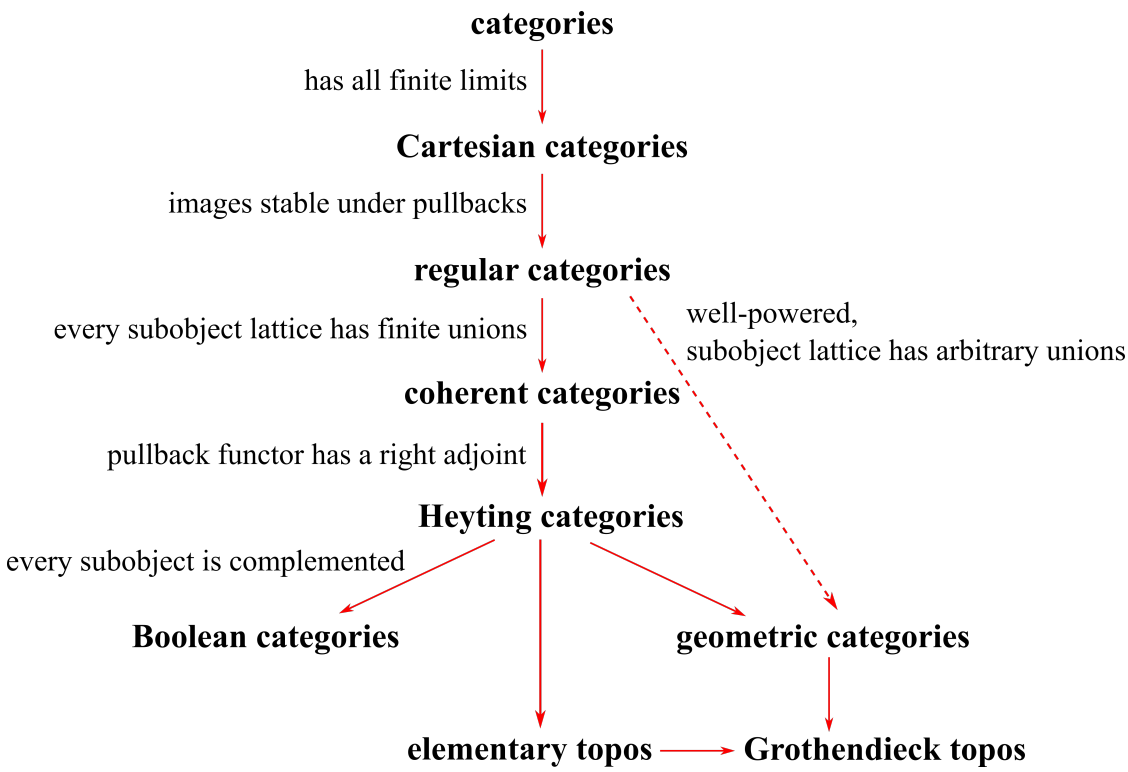
A **geometric morphism** between 2 elementary toposes, $f : \mathscr{C}_1 \to \mathscr{C}_2$, is a pair of functors of the form:

$$\mathscr{C}_1 \underset{f_*}{\overset{f^*}{\leftarrow\!\!\rightarrow}} \mathscr{C}_2 \tag{87}$$

such that f^* is **left exact** and left adjoint to f_*.

A functor is called **left exact** if it preserves all finite limits (*ie*, limits of all finite diagrams). Dually, a **right exact** functor preserves all finite co-limits.

I made the following diagram from [Caramello 2018] Ch.1:



(88)

Grothendieck's idea of topos is a *stronger* notion than elementary topos.

Fuzzy logic

From §5.2.4 of [Bělohlávek, Dauben, and Klir 2017].

In any topos one may define arrows which play the role of **truth functions** of logical connectives on Ω. In this sense, each topos "internalizes" a logic. It is a striking fact that, in general, Ω becomes a Heyting algebra and thus the internal logic of topoi is intuitionistic logic.

An early attempt of a category of fuzzy sets is $\mathbf{S}(L)$ proposed by Goguen. For a complete residuated lattice L, $\mathbf{S}(L)$ has as **objects** the pairs $\langle U, A\rangle$ where U is a set and A an L-set of U, *ie*, $A : U \to L$, *ie*, a subset of U fuzzified by L; and as **morphisms** $R : \langle U, A\rangle \to \langle V, B\rangle$ the L-relations $R : U \times V \to L$ satisfying $\bigvee_{u\in U} A(u) \otimes R(u, v) \leq B(v)$, along with $\circ$-composition. This is later shown to be a **quasi-topos**.

In 1981 Eytan described a category of **Heyting-algebra-valued sets**, similar to Goguen's $\mathbf{S}(L)$, and claimed that it forms a topos, which was disproved by Pitts the next year.

Graphs

The category of graphs is a functor category:

$$\mathbf{Graphs} = \mathbf{Sets}^{\Gamma} \tag{89}$$

where Γ is a category pictured as follows:

$$\bullet \rightrightarrows \bullet \tag{90}$$

It has exactly 2 objects and 2 distinct arrows. It follows from this that **Graphs** is Cartesian-closed [Awodey 2006] p.143.

Homotopy type theory

[Rodin 2014]

2 continuous maps:

$$A \underset{g}{\overset{f}{\rightrightarrows}} B \tag{91}$$

between 2 spaces A, B from **Top** are called **homotopical** when there exists a homotopy between them. A homotopy h between 2 continuous maps $f, g : A \to B$ in **Top** is a continuous map $h : A \times [0, 1] \to B$ such that $h(0) = f$ and $h(1) = g$.

By identifying all homotopic maps in **Top** one gets the **homotopy category hTop**.

Quillen in 1967 axiomatized homotopy theory as the **model category**. Awodey and Warren showed that every model category admits an internal language, which is a form of Martin-Löf type theory.

Voevodsky provided this inductive definition:

1. A space A is called **contractible** (a.k.a. a space of h-level 0) when there is a point $x : A$ connected by a path with each point $y : A$ in such a way that all these paths are homotopic.
2. We say that A is a space of h-level $n + 1$ if for all points x, y the path spaces $\mathrm{Path}_A(x, y)$ are of h-level n.

Then we have these hierarchical levels:

- **Level 0:** up to homotopy equivalence there is just one contractible space that we call "point" and denote pt
- **Level 1:** up to homotopy equivalence there are 2 spaces at this level: the **empty space** $\varnothing$ and the **point** pt. We call them **truth values**. We also refer to types of this level as **properties** and **propositions**. Propositional logic lives at h-level 1
- **Level 2:** Types of this level are characterized by the property that their path spaces are with empty or contractible. So such types are disjoint unions fo contractible components (points), or in other words **sets** of points. This will be our working notion of set available in this framework.
- **Level 3:** Types of this level are characterized by the property that their path spaces are sets (up to homotopy equivalence). These are obviously (ordinary flat) **groupoids** (with path spaces hom-sets)
- **Level 4:** Here we get 2-groupoids

  $\vdots$
- **Level n + 2:** n-groupoids

Domain theory

Both λ-calculus and combinatory logic are formalisms for expressing arbitrary **functions**. If the domain of the whole function is D and the number of functions by $D \to D$ is $|D^D|$, $|D^D|$ must be greater than $|D|$ according to Cantor's theorem, even if D is infinite. In other words, λ-calculus and combinatory logic are unlikely to have models. This conclusion is very disturbing. But in 1971, the problem was solved by Dana Scott and C Strachey, creating **domain theory**.

 [Vickers 1989] [Goubault-Larrecq 2013].

Scott's solution is to give domain D endow with a **Scott topology** and then only consider $D \to D$'s **continuous function**. The latter is a small number, so it avoids Cantor's theory.

Scott worked on Boolean-valued models of set theory and the idea of replacing Boolean by **Heyting algebras**. His considerations lead to topoi structures and the concept of **Heyting-valued sets** which play an important role in the development of fuzzy topos theory.

 From [**Pitts1991**] §4-5:

There is a correspondence between logic theories and pre-ordered sets, tracing back to the correspondence between classical propositional logic and Boolean algebras.

Lawvere's dictum: *"categories are theories and models are functors"*

..... [To be continued] 

Conclusion and further questions

- We saw that the model structures are "grounded" in the sense that they don't contain variables. Does this restriction result in a sub-category of toposes? What is the categorical characterization of such structures?
- Does Lawvere quantification generalize to cases where the domain may not be Cartesian products or where f is not a projection function? This may allow for more variations of geometric models.
- Fuzzy logic and its relation to topos theory.

References

Abramsky and Tzevelekos (2011). Introduction to categories and categorical logic. In: New structures for physics. Ed. by Coecke. Chap. 1.

Awodey (2006). Category theory.

Battaglia et al. (2018). Relational inductive bias, deep learning, and graph networks. In: URL: `https://arxiv.org/pdf/1806.01261.pdf`.

Bělohlávek, Dauben, and Klir (2017). Fuzzy logic and mathematics – a historical perspective.

Bergadano and Gunetti (1996). Inductive logic programming - from machine learning to software engineering. MIT.

Bertsekas (2013). Abstract dynamic programming.

Brown (2013). Discrete structures and their interactions. CRC Press.
Caramello (2018). Theories, sites, toposes – relating and studying mathematical theories through topos-theoretic 'bridges'.
Goldblatt (1984, 2006). Topoi – the categorical analysis of logic.
Goubault-Larrecq (2013). Non-Hausdorff topology and domain theory – selected topics in point-set topology. Cambridge new mathematical monographs 22.
Grilliette (2017). A Functorial Link between Quivers and Hypergraphs. In: URL: https://www.researchgate.net/publication/305787097_A_Functorial_Link_between_Quivers_and_Hypergraphs.
Jacobs, Bart (1999). Categorical logic and type theory. Elsevier.
Lawvere and Rosebrugh (2003). Sets for mathematics. Cambridge.
Lawvere and Schanuel (1997, 2009). Conceptual mathematics. Cambridge.
Li and Vitanyi (2008). An introduction to Kolmogorov complexity and its application (3rd edition). Springer.
MacLane, Saunders and Ieke Moerdijk (1992). Sheaves in geometry and logic – a first introduction to topos theory. Springer.
Miller and Sturmfels (2005). Combinatorial commutative algebra. GTM 227.
Rodin, Andrei (2014). Axiomatic method and category theory.
Streicher (2006). Domain-theoretical foundations of functional programming. World Scientific.
Vickers (1989). Topology via logic.