A Practical Doctrine for Mathematical Applications

Henson Graves

Algos Associates
2829 West Cantey Street
Fort Worth, TX 76109 United States

Abstract: A doctrine, in the sense of Jon Beck, is outlined for representing and reasoning about mathematical applications. The doctrine is a two category whose objects are axioms sets and whose morphisms are functors. The use of this doctrine for developing and reasoning about axiom sets corresponds closely to informal practice, but differs from textbook development. An application axiom set is specified by a signature and formulae in the language of the signature. Each application axiom set uses a base language with term constructions from topos theory. The axioms are Horn rule axioms. These rule axioms sets generate a topos as their deductive closure. A First Order Logic is used to express the axioms, but extends standard presentations in that terms are allowed to have decidable preconditions for being well-formed. Constructions such as composition of maps are defined as functions terms. The axiom sets are represented as tuples within the 2-category doctrine. The 2-category is a meta logic for operating on axiom sets and maps between them. The doctrine is also a specification for a class of software tools for developing and analysing axiom sets that represent applications.

Keywords: axiomatics, category theory, engineering models, doctrines, ontology, topos theory,type theory

1. Introduction

An approach to axiomatics for science and engineering, as well as mathematics, in the tradition of William Lawvere, Lawvere (2005), and Andrei Rodin, Rodin (2014) is outlined. The approach uses a doctrine to represent axiom sets and transformations between them. The use of a doctrine, in the sense of Jon Beck, as a foundation for axiomatics accords well with a statement by Jaakko Hintikka on the subject of axiomatics, Hintikka (2011). Hintikka notes that axiomatics requires logic in two different senses. First there is the logic that is being used in an axiomatic theory itself, in deriving theorems from axioms. Secondly, the logic for the metatheory of the axiom sets used to obtain results are about the theories. The objects of the doctrine outlined are axiom sets and the maps are logical functors between axiom sets. The logic of the doctrine is the meta logic for the axiom sets. This doctrine can be interpreted in the category of categories. Its significance for engineering and science applications is that it is a specification for a class of software tools for developing and analysing axiom sets that represent applications.

The axiomatic representation of scientific theories has a long history. However until recently axiomatizations have not played a significant role in everyday engineering and science applications. This is changing. The development of models, in the engineering sense, as descriptions of systems under design or analysis,is increasingly becoming the authoritative information source in application development. There are well developed modelling languages and tools for authoring and analysing models with widespread industry use. While engineering models are not generally presented as axiom sets, there are results embedding engineering models in logic Berardi et al. (2005), Graves & Bijan (2011). These results mean that engineering models can viewed as axiom sets with the potential for using automated reasoning to solve application problems. Currently, model developers often make assumptions which are not explicitly formalized within their models. However, this implicit knowledge can be formalized within axiom sets which extend the informal models. Formalizing implicit knowledge as part of models as axiom sets becomes increasingly important. System analysis based on models which have undergone validation are now often taken as legal determinates of system behaviour. In the following engineering models will be identified as axiom sets within a logic. The issue is what kind of logic is adequate to capture the intent of these models and to operate on the axiom sets as objects.

Many everyday modelling applications including vehicle and molecular structure design and analysis require a language with an expressiveness comparable to set theory. Patrick Suppes, Suppes (2002) has suggested that no standard first order theory is sufficient for science applications as physics can not be represented in these theories. He suggests using set theory. Set theory is not well suited for automated reasoning, as the axioms are not very algebraic and extensive use of existential quantification is made. Set theory, however, is not the only choice of language with sufficient expressiveness. An elementary topos has the expressiveness needed and are very algebraic. However, elementary topos theory also employs full first order logic with existential quantification, which is also not conducive to automated theorem proving. Topos theory axioms can be modified to make them suitable for automated reasoning.

The axioms for an elementary topos, with one exception, subobject classification, are readily put in a rule form suitable for automatic reasoning. The axioms for products, equality, subtypes, and power types are all known to have rule forms, Lambek & Scott (1986). A modification of the subobject classification axioms can be given as Horn rules in a first order language. The replacement axiom is described briefly in Section 2.2. The axioms for the language constructions generate a topos as their deductive closure. Application axioms in the form of rules also generate a topos which is a quotient of the free topos generated by the language axioms. This axiomatic formalism can be used for embedding models in a variety of modelling languages including SysML Friedenthal et al. (2006) and Description Logics Baader et al. (2003).

The logic used to formalize the topos-based axiom sets is a species of First Order Logic. However, the first order axiomatization allows function terms to have definedness conditions. Operations such as map composition are represented as first order function symbols which have preconditions for being well-defined. The use of rule axioms for the topos language constructions logic satisfies this criteria. Formulae are generalized rules (Horn clauses with a single consequent where the clauses are atomic formulae which include negation only for equality). This form of axioms is sufficient to represent a wide class of applications and the reasoning can be performed efficient inference systems Graves & Bijan (2011). By extending standard First Order Logic for conditionally defined term constructions models of a theory within a category coincides with functors defined for a theory. Using a conditional logic provides a unification between models of theories and a funtors. Interpretations of function terms with definedness conditons are required to be defined only which their definedness conditions are satisfied. Mappings between the axiom sets are functors interpreted within a category of toposes.

In application practice, as well as mathematics practice, one generally works with multiple axiom sets (models in the sense of science and engineering). In mathematics axioms are often specified for a concept, such as a group, in First Order Logic with equality. A group is then defined as any structure which satisfies the axioms. The development of the theory of groups, for example, Bourbaki (1948), takes groups as objects, classifies them, and constructs mappings between them. This development is usually done informally in set theory. Modelling in product development is similar. One might develop design model and refine this model into an implementable design which meets requirements criteria. For product design models axioms are added and modified to achieve an implementable design. The development process, including design and analysis, generally makes use of multiple models of physical components to be incorporated into a design and physics models which are used in models of the environment in which a product is designed to operate. Application development tools generally not only allow a author to develop a model, but operate on multiple models and map between models. The objects are the axiom sets and the models are logical functors. The logic of the doctrine extends the logic of the axiom sets to represent axiom sets as tuples which contain the signature and axioms.

Mathematics and engineering development take place in a language in which reasoning is done within the theory of an axiom set, and where axiom sets can be packaged as modules which can be operated on and reasoned about. By including in each application axiom set the language constructions and axioms for a topos, one obtains a common ontology in which to integrate and combine multiple axiom sets. The use of a base axiom set for applications is an extension of the capability of engineering modelling tools to provide a base collection of data type constructions and operations. The application axiom sets generate a topos as the deductive closure of the combined axioms. Axiom sets, by including the topos signature and axioms, can use as much mathematical structure language as needed. Axioms can be given for concepts such as a group, a finite state machine, or representations of engineering and science models. The application axioms together with the ontology axioms can be used to do elementary reasoning within the ontology. Reasoning within an axiom set can be done using natural deduction proof rules and term rewriting for equality.

The packaging of axiom sets as objects and functors as morphisms yields a doctrine. As a doctrine the axiom sets are represented as tuples which contain the topos signature and axioms, as well as symbols and axioms for the particular application. This doctrine of axiom sets and functors is practical, not only in the sense that it conforms to mathematical practice, but is also suitable for implementation as an interactive computational system. The doctrine corresponds to an abstract specification for a model/axiom set software development tool. It elaborates features found in many current model development tools. A detailed presentation of the formalism can be found in Graves & Bijan (2011). Axiom sets can be developed for product designs within engineering modelling tools and automatically exported to reasoning engines. Reasoning both within an axiom system and about axiom systems can be used to solve common engineering problems. The result is an integrated framework suitable for interactive development within a theorem proving and proof checking computational system.

2. Engineering models as axiom sets

An engineering or science model, such as an aircraft design model or a model for the human heart, can be expressed as axiom sets in a language with the map and type constructions of topos theory. Application models are often informally represented as directed graphs. The nodes of the graphs are types, such as the type of a vehicle engine, and the arrows are associations, such as the association of an instance of a vehicle type with its engine. Formal modelling languages such as SysML which have a graphical syntax use this kind of primitives. In SysML each association has a domain and codomain type. For example, a design specification for a vehicle may use types for components such as engines and wheels. The structure of a vehicle is described by associations such as the specification that each vehicle has an engine and has four wheels all connected in a specific way. In detailed models component attributes specify form, fit, function, and material composition. The models may also specify constraints on interactions between components, for example, the maximum pressure permitted on oil lines. When these constraints are violated the model becomes inconsistent in the sense that their are no valid implementations.

What can we say about the purpose of an engineering model and how a model can be used to solve problems? In the engineering of mechanical and biological products models are used to analyse the structure and behaviour of a system of interest. In the case of a simple molecule such as H_2O one may not think any special machinery is needed. However, when designing molecules for drugs or analysing them we may need to determine if a molecule contains instances of a particular substructure such as a carbon ring. For an aircraft we may want to determine what components can draw current from the electrical system. Many engineering

problems translate into the question of whether an axiom set is consistent. A model can be used to answer these kinds of questions, provided we are certain the model precisely describes the systems of interest.

How can we be certain that the model describes the systems of interest and only those systems? This is a meta theoretical question in sense mentioned by Hintikka, as it is a result about the model. For structural models we may be able to answer this question when, for example, we can prove that all of the realizations of the axiom sets have the same structure. In the case of H_2O this means that all realizations of the axioms within a logical model, i.e., individual water molecules, have exactly 2 hydrogen atoms and 1 oxygen atom connected by bonds in a specific way. For aircraft and other mechanical products extensive methodology and procedures are employed to validate models. This topic is also meta theoretic.

2.1. Examples

Three examples are used to motivate the kind of logic needed to represent engineering and science models as axiom sets, one mathematical, one computer science, and one molecular science.

2.1.1. Water

Consider building an axiom set for the class of H_2O molecules. An individual molecule has three atoms, an oxygen atom and two hydrogen atoms. The oxygen atom is connected to each of the hydrogen atoms with a bonding relation. An axiom set whose logic models (valid interpretations) consist of one or more H_2O molecules can be given using the signature below.

$$\begin{aligned} H_2OMolecule =< H_2O, Hydrogen, Oxygen, \\ hasHydrogen1 : H_2O \rightarrow Hydrogen, \\ hasHydrogen2 : H_2O \rightarrow Hydrogen, \\ hasOxygen : H_2O \rightarrow Oxygen, \\ covalentbond1 : Image(hasOxygen) \rightarrow Image(hasHydrogen1) \\ covalentbond2 : Image(hasOxygen) \rightarrow Image(hasHydrogen2) \end{aligned} \tag{1}$$

The signature uses two kinds of terms, which we call maps and types. The symbols H_2O, $Hydrogen$, and $Oxygen$ are types. The symbols such as $hasHydrogen2$ and $covalentbond1$ are maps. An expression such as $hasOxygen : H_2O \rightarrow Oxygen$ specifies a domain and range type for the map. The map associates water molecules with oxygen atoms. An individual water molecule is represented as a map which has its domain a special type, One, called the terminal type. Note that the domain and ranges of the covalent bonds are not atomic types, but use as their domain and range types images of the part maps. The range type of a water molecule is the type H_2O. The expression $Image(hasOxygen)$ is a type construction is used to represent the oxygen molecules which as part of a water molecule. The hydrogen atoms which occur within an H_2O molecule is a subclass of hydrogen atoms.

Axioms can be given to express that atom types are disjoint which we write as

$$Oxygen \perp Hydrogen. \tag{2}$$

An axiom that says that the same hydrogen atom in a molecule cannot be used by both part relationships

$$hasHydrogen1 \neq hasHydrogen2. \tag{3}$$

Other conditions can be given to ensure that any molecule in a model has the expected structure of exactly three atoms connected in the specific way. While the axioms for the H_2O molecule have only been indicated, they can all be given as Horn rules.

We have given a signature and some of the axioms for the H_2O molecule axiom set, but we haven't said exactly what logic the axioms are expressed in. In engineering parlance the H_2O molecule is a use case to help us determine the requirements for a suitable logic. The notation used looks very much like that used for sets and functions in mathematics. However, the physical interpretation of types and maps is quite different. For example, the type $Oxygen$ is not interpreted as a collection of atoms, but corresponds to our ability to recognize that a map has $Oxygen$ as codomain type. This kind of interpretation is much more in accord with Lawvere's concept of functorial semantics. Lets look briefly at two more examples, groups and finite state machines. Both occur in mathematics and in applications.

2.1.2. Groups

For the development of group theory, as opposed to the axiomatization of groups, one generally starts with a signature which contains function symbols for the identity element, element composition, and for the inverse of an element. For example one might start with

$$\begin{aligned} \mathcal{G} =< G, \\ id : G \to G, \\ \circ : (G, G) \to G, \\ inv : G \to G, \\ id \circ g = g \circ id, ... > . \end{aligned} \tag{4}$$

At this point we note only that the axiom set is represented as a tuple whose components are expressions which use syntax familiar from set theory and topos theory.

2.1.3. Finite state machines

A finite state machine is usually presented using language constructions familiar from set theory. For example, a deterministic finite state machine is generally represented as a quintuple

$$\mathcal{F} = (\Sigma, S, F, s_0, \delta), \tag{5}$$

where Σ is the input alphabet (a finite, non-empty set of symbols). S is a finite, non-empty set of states. s_0 is an initial state, an element of S. F is the set of final states, a (possibly empty) subtype of S. δ is the state-transition function: $\delta : S \times \Sigma \to S$ (for a nondeterministic finite automaton δ is $\delta : S \times \Sigma \to \mathcal{P}(S)$ where $\mathcal{P}(S)$ is the power type of S. This description is really little beyond the signature. The description for a finite state machine can be made more precise by adding typing information for the symbols in the signature. The representation

$$\mathcal{F} =< \Sigma, S, F, s_0, \delta, s_0 : One \to S, \delta : S \times \Sigma \to S, F \sqsubseteq S > \tag{6}$$

expands the quintuple to provide typing information for the symbols in the quintuple. The behavior of a FSM is often described by rules. For example, a FSM which has two states, on and off and rules which turns itself off when it is on and conversely can be specified using notation for a case construction (coproduct) with:

$$\begin{aligned} (\Sigma, \{on, off\}, s_0 = off, \\ \delta = caseof(on \to on := off, off \to off := onn), F = on, off) \end{aligned} \tag{7}$$

Note that the specification for a FSM uses additional language constructions for Cartesian product and a subtype relation $\sqsubseteq$. Informally the axiom set specifies a class of FSM. With the addition of the $on - off$ axioms a further specialized subclass is specified.

2.2. Axiomatics

From the examples, we have an indication that the paradigm of axiom set as building block works for engineering and science applications, as well as mathematics. Before addressing the packaging axiom sets as objects and transformations between them as morphisms, i.e., as a doctrine, lets go back to informal mathematics used within an axiom set to see how these concepts can be formalized in a form suitable for automated reasoning in science and engineering.

The language constructions of the examples include composition of maps, tuples, product types, the image type, and subtype relations. Additional examples use other constructions familiar from mathematics. As noted in the introduction, these language constructions can be given axioms that can be used to derive consequences. The axioms for the language constructions are derived from topos theory. The logic used present axioms for the language constructions as well as represent the application axioms is first order, but differs from standard presentations in that term constructions which have decidable definability preconditions are admitted. This extension enables the axiomatics to conform much more closely to informal practice, but in a well defined sense, the extension is hygienic. For computational tractability the axiom sets are restricted to rule axioms.

As an example of how axioms for maps and types can be given, consider that informal mathematics and mathematical applications often use operations which are not totally defined. For example, composition of maps is well-formed when ranges and domains match appropriately. This is expressed as an axiom

$$f : A \to B, g : B \to C \Rightarrow f.g : A \to C \tag{8}$$

where $f.g$ is the notation for the composition of f and g. Composition terms are interpreted as being defined only when the interpretation of their definition conditions is true. In standard FOL operations represented as function symbols are required to be total in their interpretations. This constraint makes first order axiomatization of a directed graph or a category difficult as the partially defined operations have to be represented as relations. However, these axioms can be expressed in a form in which new function terms do not depend for their introduction on preconditions that require provability of something. This condition has been called Tarski's requirement for the Hygiene of Formal Systems. First order reasoning works normally in this context and is less messy than representing the map constructions as function relations, or using a logic with explicit partial functions. Proofs use standard natural deduction inference rules.

Axioms which satisfy the hygiene conditions can be given for the language constructions from topos theory. Existential axioms from topos theory are replaced by function constructors. The preconditions for well-definedness are decidable. Axioms may require proving preconditions in order to make use of the conclusion. The axioms for the map and type term constructions are Horn clauses with a single consequent where all of the clauses are atomic or negation of map equality. By extending the signature of the base and axioms for the language constructions, axioms sets for application can be given. These axioms are also in the form of generalized rules. In addition predicates on the symbols of the signature are allowed. Reasoning using rule axioms is computationally efficient.

This form of axioms for a category has the following consequence. An interpretation only requires composition and other constructors to be defined, when the interpretation of the definedness conditions are met. The definition of model for an axiom set coincides with the definition of a functor from the axiom set to a category. The models of axiom sets are functorial models. This suggests that not only that one should view fol logic as a special case of FOL with definability preconditions, but that standard FOL needs to accommodate category theory.

Axioms for products, sums, subtypes, exponentials, and powers can be given with decidable predefinition rules. The notation for the product type constructor is:

$$(X1, X2) \tag{9}$$

and the tuple map constructor is:

$$< f, g > \tag{10}$$

The axiom below gives the precondition for a tuple to be well-formed.

$$f : Z \to Y, g : Z \to X \Rightarrow < f, g > : Z \to (Y, X) \tag{11}$$

2.2.1. Subobject classification

Of the topos axioms only the subobject classification axiom is not readily expressed in a rule form. The use of this axiom is critical for expressing axioms in engineering and science applications. For example, in a model for a vehicle with types $Vehicel$, $Engine$, and a map $p : Vehicle \to Engine$ which associates an engine to a vehicle, one wants the map p to have the property that it doesn't assign the same engine to two distinct vehicles. To prove properties about the design model in this situation one needs to know that for monics there is an isomorphism from the image of the monic back to its domain. A replacement subobject classification axiom is used here builds on a property found in set theory, but not in an arbitrary topos.

The subobject classification axioms are rule axioms with equational conclusions which use hygienic map constructors in that they yield maps with well-defined types. The two constructors used are a characteristic map constructor $char_f$ and a factorization constructor $fac_{h,f}$ which is used to factor a map $h : T \to X$ through a monic $m : Y \to X$ when $char_m(h) = true$. This factorization constructor is in addition to the subtype factorization constructor. The axioms, which are called monic classification axioms, imply that each type X has a canonical subobject structure, $incl_{char_f}$ determined by the characteristic maps $char_f X \to \Omega$.

$$f : Y \to X \Rightarrow char_f : X \to X, \tag{12}$$

$$char_f(f) = true \tag{13}$$

$$f : Y \to X, p : X \to \Omega,$$
$$p(f) = true, char_f(incl_p) \Rightarrow p = char_f \tag{14}$$

$$f : Y \to X, h : T \to X \Rightarrow fac_{h,f} : T \to Y \tag{15}$$

$$m : Y \to X, h : T \to X, monic(m), char_m(h) = true \Rightarrow$$
$$h = m(fac_{h,m}) \tag{16}$$

$$incl_{true_X} = id_X \tag{17}$$

$$c1 : X \to \Omega, c2 : X\{incl_{c1}\} \to \Omega \Rightarrow incl_{char_{incl_{c1}(incl_{c2})}} = incl_{c1}(incl_{c2}) \tag{18}$$

$$c1 : B \to \Omega, c2 : D \to \Omega \Rightarrow incl_{(c1(proj1) \wedge c2(proj2)} = incl_{(incl_{c1}, incl_{c2})} \tag{19}$$

$$char_{f^*} = \exists! x.\epsilon(x, id_{Pow(X)}) \wedge char_f(x) \tag{20}$$

A consequence of the axioms is that each type has a canonical family of subobjects, corresponding to the subsets of a set in set theory. The monic classification axiom enables the image of a monic $m : Y \to X$ to be characterized as a subtype of X. The axiom uses the constructor $char_f : Y \to X$ and a constructor $m^{-1} : Im(m) \to Y$. When m is monic, m^{-1} is a map $m^{-1} : Im(m) \to Y$ a map which provides an isomorphism of the domain of m with its image.

2.2.2. Topos generated by an axiom set

By including the language construction axioms in a module, the deductive closure is a topos. The deductive closure of an axiom set (modules) $\mathcal{M}$ is a topos $\mathcal{T}(\mathcal{M})$. To generate the topos we use the topos signature and axioms $\mathcal{A}$ is augmented with $\mathcal{M}$. The theory of $\mathcal{M}$ is the collection of formulae derivable from the application axioms together with the topos axioms using the full intuitionistic "conditional" first order logic. The axioms for an axiom set generate a topos with a natural numbers type when provably equal map terms are identified. The first order theory generated by deductive closure of an axiom set is a topos with a form of canonical subobjects. An interpretation of the signature of the axiom set coincides with the definition of a functor from the theory generated by the axiom set.

3. Doctrine

In this section we look at the properies of the logic used to represent axiom sets as objects. The logic of the doctrine reflects the operations used to develop the models (axiom sets). Application development tools generally not only allow a author to develop a model, but operate on multiple models and map between models. The doctrine formalism abstracts the capabilities found in engineering model development tools. The logic of the doctrine, which has axiom sets as objects and functors between axiom sets as maps, is a typed logic whose types form a cartesian closed category. The types are meta types derived from the sorts of the axiom set logic. The task of constructing a model (axiom set) to represent to represent a design specification such as that for a vehicle consists of combining and assembling models into a unified model of the vehicle and its operating environment. This unified model is then refined to be have the degree of detail and precision required for the application. Of course combining models sometimes requires model modification. Ideally combining models consists of unification, i.e., finding a model which specializes the model being combined . For example, one might start with an initial vehicle model with place holders (type variables)for components such as the engine and drive train. Separately developed component models are integrated by binding the type terms of the component models to the vehicle model component variables. The use of a common base language with the topos language constructions reflects the common assumptions within the application domain.

3.1. Objects of the doctrine

Axiom sets are the molecular unit for the doctrine approach. The application axiom sets include a base signature and axioms for the term constructions of this base signature. Recall the finite state machine description:

$$\mathcal{F} =< \Sigma, S, F, s_0, \delta, s_0 : One \to S, \delta : S \times \Sigma \to S, F \sqsubseteq S > \tag{21}$$

The tuple $\mathcal{F}$ is a actually a pattern description for a FSM. The symbols Σ, S, F are type variables rather than type constants. This will be made precise in Section 3.2.1 where we also determine the product type of which a concrete FMS is an instance.

The base language of term constructions use product, sum, exponential, and subtypes. The commonality of axiom sets is that they use a signature with two sorts Map and $Type$. The base ontology uses these two sorts with a collection of map and type symbols, a collection of function symbols used as constructors for the map and type terms, and a collection of predicate symbols. The type symbols include One the terminal type, $Null$ the empty type, Ω the truth value type, and N the natural numbers type. The map and type constructors are first order function symbols. The signature also contains atomic predicates such as $\sqsubseteq$ and $\in$ $monic, function, \simeq, \perp$ have

definedness restrictions. The formula $X \sqsubseteq Y$ is defined only when X is instantiated as a type. The axioms for map constructions provide antecedent conditions for a term to be well-formed.

3.2. The Doctrine as a category of axiom sets and functors

The metalanguage we have been using to talk about axiom sets axiom sets such as $\mathcal{G}$ and $\mathcal{F}$ mirrors the language used by a computational system for authoring, and reasoning about these axiom sets. A computational system which has axiom systems as its objects, works in much the same way as a word processing software which has text files as its objects. The computational system allows users to create and edit axiom sets. When an axiom set is created the system instantiates the axiom system with the base signature and axioms. Users may add to this axiom set using the base signature and any additions. The metalanguage used by the computation system, which need not be formalized here, has as its objects axiom sets,the theories they generate, and transformations between these theories. This language and its computation and reasoning rules are simply a precise form of the informal metalanguage used in this discussion.

3.2.1. Metatypes

As noted an axiom set is a tuple whose elements contain the symbols in the signature followed by the axioms. The use of tuples indicates that the metalanguage has Cartesian products. The types of the doctrine are meta types constructed from the sorts for Map, $Type$, and from a meta type for $Formula$. The meta type constructions are used to distinguish the different kinds of maps, types, and formulae that occur in an axiom set. The notation of a letter such as X preceded by a : is used for a metatype. The instance relation also uses :. For example, the expression $f :: T$ assets that f is an instance of the meta type $: T$. The meta types are constructed from type symbols, type variables, and the base type constructions. The meta type constructions can be specified by a grammar. For example, the grammar production rules

$$: Type :=: TypeSymbol | : TypeVar | : ProdType | : Subtype | : PowerType \tag{22}$$

$$: ProdType := (: Type, : ProdType) \tag{23}$$

specify a part of the grammar for meta types. Similarly the map meta type grammar specification includes:

$$: Map :=: MapSymbol | : Comp | : Tuple | : Relation \tag{24}$$

In addition to the grammar for maps and types, a grammar would be used to specify the formulae.

$$: Rule := : Atom | Conjunction \Rightarrow Atom \tag{25}$$

However, for this discussion we distinguish subsets of map and type symbols using Boolean connectives to define subtypes of Map and $Type$. These types are expressed with set builder notation. For example, for a signature with map symbols $p1, p2, p3, b1, b2$ we use the predicate $f \in \{b1, b2\}$. For example, production rules for the rule formulae include:

$$D = \{a :: MapSymbol | a : One \to \mathbf{G}\} \tag{26}$$

Similarly,

$$: Map\{\delta : S \times \Sigma \to S\} \tag{27}$$

is a subtype of : Map. For the meta product types, i.e., products of meta types we use the parenthesis notation where the entries are meta types. For example,

$$(: Type, : Rule) \tag{28}$$

is the product of the meta types : $Type$ and : $Rule$. We also use the computer science notational convention

$$(S :: Type, R :: Rule) \tag{29}$$

where : $Type$ and : $Rule$ are meta-types and S and R are names for the projection maps for the product. The finite state machine axiom set tuple is an instance of the product type

$$\begin{aligned}(\Sigma :: Type, S :: Type, F :: Type\{F \sqsubseteq S\}, s_0 :: Map\{s_0 : One \to S\},\\ \delta :: Map\{\delta : S \times \Sigma \to S\})\end{aligned} \tag{30}$$

We are identifying Σ, S, F as projection maps. The notation : $Type\{F \sqsubseteq S\}$ specifies a subtype of : $Type$.

3.2.2. Coalescing models and functors

The use of a first order logic in which decidable term formation rules are allowed enables the category concept of functor to coalesce with the first order concept of model. A first order model for an axiom set $\mathcal{A}$ is a mapping $I : \mathcal{A} \to \mathcal{C}$ where $\mathcal{C}$ is a category. The mapping I sends terms of the two sorts, map and $type$ to maps and objects in the category. The constructor symbols are mapped to operations in the category. The values of the constructors define operations on the maps and types in the closure of the image of the axioms. These operations define a topos structure for on this subcategory of the target category.

4. Conclusion

The topos formalism and its doctrine for axiom sets have been developed to represent application models as axiom sets and to integrate reasoning with application model development in the context of model development tools. Integration of modelling with formal reasoning has great potential in everyday application design and analysis use. This quest to integrate application modelling tools with logic is only the latest stage in the long history of axiomatic representation of scientific theories. One might ask why this task has proven so difficult, and why a framework such as described above will be successful. The primary difficulty for the axiomatic approach, in this author's opinion, has been in finding a suitable language in which to represent application models. The language constructions needed based on the author's experience with building and analysing application models is that a language with constructions such as is found in set theory is required. Further if automated reasoning is to be practical one needs a formula language restricted to something such as Horn rules.

4.1. Base language as ontology

In this doctrine formalism all of the axiom sets use the common base language for a topos. This base language provides a common base ontology. As an ontology it is sufficient to represent a variety of mathematics and a variety of applications which use classical physics. Many of the engineering models use additional axiom sets for physical axioms such as optics, electro magnetics, and gravity. However, it is possible that base ontologies with other properties may be needed for applications such as those which have quantum mechanical properties.

4.2. Comparisons

Are there other candidates for a base ontology? Type theory is the only other formalism with comparable expressiveness. Type theory has played a role in the foundations of software development comparable to that suggested for topos theory in science and engineering. As is well known the two formalisms are closely related, Lambek & Scott (1986). According to the definition of a type theory given there, the topos formalism can also be considered as a type theory. In the author's experience the use of both a domain and codomain type for terms works better for the engineering and science applications. One other formalism, Description Logic, Baader et al. (2003) is worth mentioning as it is a logic-based formalism and has received considerable application use.

Description Logics use the paradigm that models are axiom sets. Description Logics with extensions to rule logics have been employed with partial success to represent applications in science and engineering as axiom sets,Magka et al. (2012). Description Logic representations of associations use relations, Baader et al. (2003). However, equivalently these associations can be described by maps whose codomain is a power type. By using a two sorted rule logic framework one gets the benefit of the Description Language constructions, as well as, having an additional expressiveness of variables and term constructions. Thus, avoiding problems inherent in using Description Logic and its extensions for structural modelling, Graves & Horrocks (2008). As noted in there DLs cannot be used to axiomatize a molecular structure such as cyclobutane which always has a ring of carbon atoms. At least one tree shaped structure will be consistent with the axioms. A logic programming formalism described in Magka et al. (2012) called Description Graph Logic Programs (DGLPs) has been suggested as an approach to remedy deficiencies of DGDL. DGLP does not contain an explicit representation of the graph structures used in the descriptions and does not permit classification of graph theoretic structures. DGLP places the burden of modelling on identifying the functions which represent the graph structure and on producing the collection of graph orderings. Description logics even with the logic programming extensions can be represented within the topos framework.

4.2.1. Doctrines

In application practice, as well as mathematics practice, one generally works with multiple axiom sets (engineering models). For product design axioms are added and modified to achieve an implementable design. The development process, including design and analysis, generally makes use of multiple models of physical components to be incorporated into a design and physics models which are used in models of the environment in which a product is designed to operate. Application development tools generally not only allow a author to develop a model, but operate on multiple models and map between models. Such an application development tool can be described as implementing a doctrine, in the sense of Jon Beck. The informal concept has been described as like a theory, except to be interpreted in the category of categories. The objects are the axiom sets and the models are logical functors. The logic of the doctrine extends the logic of the axiom sets to represent axiom sets as tuples which contain the signature and axioms. Analysis including reasoning is done within the context of a specific theory. An axiom set is developed for the application at hand. The application theory generated by the axiom set may include broader scientific theories and may be refined or changed as the process plays out.

By formalizing the topos constructions as rule axioms within Conditional first order logic the natural definition of model for the axiom set coincides with the definition of a functor from the axiom set. The models of axiom sets are functorial models. The natural definition of model for a module is a strict logical functor, as all of the topos map and type constructions are preserved. All of these models use the same basic map and type constructions. The concept of doctrine,

provides a framework for the development of axiom sets. The doctrine of axiom sets has a natural interpretation in the category of categories. For this doctrine, model of an axiom set is morphism in the 2-category The doctrine in which a theory lies specifies the structure in which models of that category can be internalized.

References

Baader, F., Calvanese, D., McGuinness, D., Nardi, D., & Patel-Schneider, P. F. (Eds.). (2003). *The description logic handbook: Theory, implementation and applications*. CUP. ((2nd edition, 2007))

Berardi, D., Calvanese, D., & De Giacomo, G. (2005). Reasoning on uml class diagrams. *Artificial Intelligence*, *168*(1), 70–118.

Bourbaki, N. (1948). L'Architecture des mathématiques. In F. Le Lionnais (Ed.), *Les grands courants de la pensée mathématique* (pp. 40–41). Paris: Hermann.

Friedenthal, S., Moore, A., & Steiner, R. (2006). Omg systems modeling language (omg sysml) tutorial. In *Incose intl. symp.*

Graves, H., & Bijan, Y. (2011). Using formal methods with sysml in aerospace design and engineering. *Annals of Mathematics and Artificial Intelligence*, *63*(1), 53–102.

Graves, H., & Horrocks, I. (2008). Application of owl 1.1 to systems engineering.

Hintikka, J. (2011). What is the axiomatic method? *Synthese*, *183*(1), 69–85.

Lambek, J., & Scott, P. S. (1986). *Introduction to higher order categorical logic*. Cambridge: Cambridge University Press.

Lawvere, F. W. (2005). An elementary theory of the category of sets (long version) with commentary. *Repr. Theory Appl. Categ*, *11*, 1–35.

Magka, D., Motik, B., & Horrocks, I. (2012). Modelling structured domains using description graphs and logic programming. In *The semantic web: Research and applications* (pp. 330–344). Springer.

Rodin, A. (2014). Axiomatic method and category theory. *Synthese Library*, *364*.

Suppes, P. (2002). *Representation and invariance of scientific structures*. CSLI publications Stanford.